REWRITING
SOLVING
PROVING

January 28, 2006

Claude Kirchner and Hélène Kirchner

REWRITING SOLVING PROVING

This is a preliminary version

We begin writing this book in the early 90. By lack of time, we did not really finish it, but since we were asked by several colleagues who wanted to use some of its contents, we make it available *as it is.* All comments on any part of this work are very welcome.

Authors: Claude Kirchner and Hélène Kirchner

LORIA, INRIA & CNRS
Campus scientifique
615, rue du Jardin Botanique
BP 101
54602 Villers-lès-Nancy CEDEX
FRANCE

E-mail: `Kirchner@loria.fr`
Web: `http://www.loria.fr/~ckirchne`
Web: `http://www.loria.fr/~hkirchne`

Acknowledgments

This document is still under development. It has been used, generaly in part, as a support for D.E.A. and Master lectures in Nancy and several other national or international schools.

Several parts of the document are based on joined works of the authors with other persons: especially with Jean-Pierre Jouannaud and Christophe Ringeissen for some chapters on unification, and with Jean-Luc Rémy for the part on parameterization.

We would like to warmly thank all our colleagues and students for their remarks and constructive criticisms. All remaining flaws remain of course ours.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This is not yet an introduction,
we just give some pointers to related surveys and books

The concept of term rewriting system emerges already half a century ago in the study of computational processes. The *ambda*-calculus played a crucial role in mathematical logic for formalizing the computability notion and can be seen as a first term rewriting system.

Since then it has been used in simplification based theorem proving, constraint solving, study and implementation of sequential or parallel computations, design of functional languages as well as to combine and integrate logic programming with functional programming, definition of language semantics, as a logical framework, etc...

The reader could also refer to the following main surveys: Gérard Huet and D. Oppen [HO80], Jan Willem Klop [Klo90a], Nachum Dershowitz and Jean-Pierre Jouannaud [DJ90a, DJ91] David Plaisted [Pla93] Jürgen Avenhaus and Klaus Madlener [AM90].

Text books on term rewriting written in English are now available: one by Franz Baader and Tobias Nipkow [BN98] that makes a nice overview of the main rewrite based techniques, and more recently a very complete one [“T02], written under the name Terese by Marc Bezem, Jan Willem Klop, Roel de Vrijer, Erik Barendsen, Inge Bethke, Jan Heering, Richard Kennaway, Paul Klint, Vincent van Oostrom, Femke van Raamsdonk, Fer-Jan de Vries, Hans Zantema.

Part I

Terms, Logics and Algebras

Chapter 2

First order logic and equational logic

This chapter contains the main notions from logic and universal algebra which are useful in this book. Since we are mainly concerned with the logic of equality, we define the relevant notions of formulas, models and deduction systems.

For an extensive presentation of these notions, the reader can refer to [Bir35, Grä79, BM67, Coh81, Hen77, Wec92], to [Orc90a, Orc90b] for a good survey of mathematics notions used in theoretical computer science and [Del86] for an introduction to deduction systems. We assume well known the usual notions of elementary set theory.

2.1 Deduction systems

In order to define the logics and proof theories we are interested in, we shortly present the notion of deduction systems that will be intensively used.

Definition 2.1 A *Deduction system* is a 4-tuple $(\Sigma, \Phi, A, \mathcal{R})$ where:

- Σ is a countable *alphabet*,
- Φ is a set of *formulas* $\{\phi_0, \phi_1, \ldots\}$ that is a decidable language over Σ,
- A is a set of *axioms* $\{a_0, a_1, \ldots\}$ that is a decidable subset of Φ,
- $\mathcal{R}$ is a finite set of *inference rules* $\{r_0, r_1, \ldots, r_n\}$ that are computable predicates over Φ.

Infinite sets of axioms are allowed and specified by axiom schemes. An inference rule is written as:

$$\begin{array}{lll} \textbf{Name} \quad \phi_0 \ \ldots \ \phi_{n-1} & \vdash & \phi_n \\ & & \textsf{if } \mathit{Condition} \end{array}$$

and means:

"Given $\phi_0 \ \ldots \ \phi_{n-1}$ deduce ϕ_n if *Condition* holds."

A *derivation* d of the conclusion $c \in \Phi$ from the *premises* $P = \{p_0, \ldots, p_{n-1}\} \subseteq \Phi$ is a finite nonempty sequence $(d_0, \ldots, d_m)$ such that $d_i \in \Phi$, $d_m = c$ and either $d_i \in A$ (d_i is an axiom), or $d_i \in P$ (d_i is a premise), or d_i has been obtained by applying some inference rule in $\mathcal{R}$ to a set $\{d_j, \ldots, d_k\}$ of formulas such that $d_j, \ldots, d_k \in d$ and $j, \ldots, k < i$. This is written as:

$$p_0, \ldots, p_{n-1} \vdash c.$$

A *theorem th* is a derivation from the empty set of premises, written:

$$\vdash th.$$

A derivation of a theorem is called a *proof*.

For a deduction system, the set of all proofs is decidable. If the set of all theorems is decidable, the deduction system is said *decidable*. If the set of all theorems is undecidable but semi-decidable, the deduction system is said *semi-decidable*. If the set of all theorems is not even semi-decidable, the deduction system is said *undecidable*. An algorithm that computes a decidable set of theorems is called a *decision procedure* for the deduction system.

2.2 First-order terms

We give here various point of views of what terms and their basic operations are.

2.2.1 Terms as strings

First-order terms are built on a vocabulary of function symbols and variable symbols. Let us first consider them as strings.

Definition 2.2 Let $\mathcal{F} = \cup_{n \geq 0} \mathcal{F}_n$ be a set of symbols called *function symbols*, each symbol f in $\mathcal{F}_n$ has an *arity* which is the index of the set $\mathcal{F}_n$ it belongs to, it is denoted $\text{arity}(f)$. Elements of arity zero are called *constants* and often denoted by the letters $a, b, c, \ldots$. It is always assumed that there is at least one constant. Occasionally, prefix or postfix notation for $\mathcal{F}_1$ and infix notation for $\mathcal{F}_2$ may be used. $\mathcal{F}$ is often called a set of ranked function symbols or a (unsorted or mono-sorted) *signature*. Given a (denumerable) set $\mathcal{X}$ of *variable* symbols, the set of (first-order) *terms* $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the smallest set containing $\mathcal{X}$ and such that the string $f(t_1, \ldots, t_n)$ is in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ whenever $arity(f) = n$ and $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for $i \in [1..n]$.

The set of variable-free terms, called *ground* terms, is denoted $\mathcal{T}(\mathcal{F})$. Terms that contain variables are said *open*. A term is *linear* if each of its variable occurs only once in it.

Example 2.1 Assume that $\mathcal{F} = \{f, a\}$ with $arity(f) = 2$ and $arity(a) = 0$. The term $f(a, a)$ is ground, $f(x, f(a, x))$ is not linear but $f(x, f(y, z))$ is.

Notation: Variables are denoted by the letters x, y, z, terms by the letters $l, r, g, d, p, q, s, t, u, v, w$.

2.2.2 Terms as trees

A term t may be viewed also as a *finite labeled tree*, the leaves of which are labeled with variables or constants, and the internal nodes of which are labeled with symbols of positive arity.

Definition 2.3 A *position* (also called occurrence) within a term t is represented as a sequence ω of positive integers describing the path from the *root* of t to the root of the *subterm* at that position, denoted by $t|_\omega$. A term u has an *occurrence* in t if $u = t|_\omega$ for some position ω in t.

The notation $t[s]_\omega$ emphasizes that the term t contains s as subterm at position ω. In some cases, ω may be omitted.

We use $\wedge$ for the empty sequence (denoting the empty path to the root) and v, ω for others. Positions are ordered in the following way: $\omega_1 \leq \omega_2$ if there exists ω_3 such that $\omega_1\omega_3 = \omega_2$. If two positions ω_1 and ω_2 are incomparable, this is denoted $\omega_1 \bowtie \omega_2$.

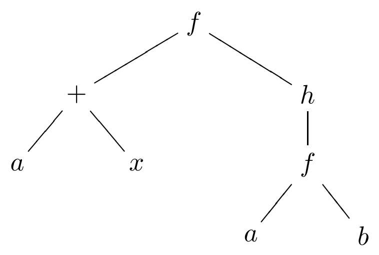
2.2.3 Terms as mappings

A term can also be viewed as a partial mapping from the monoid of positive naturals $(\mathbf{N}^*, .)$ with neutral element $\wedge$ to the set of ranked function symbols $\mathcal{F}$. $\mathcal{D}om(t)$ the domain of such a mapping, is the set of positions in t, also called *domain* of t; it should be non empty and closed under prefix, i.e. if $\omega \in \mathcal{D}om(t)$, then all prefixes of ω are belonging to $\mathcal{D}om(t)$. The size $|t|$ of the term t is the cardinal of $\mathcal{D}om(t)$. The number of nodes in the term t labeled with the symbol f is denoted $|t|_f$. $\mathcal{V}ar(t)$ denotes the *set of variables in t*. $\mathcal{G}rd(t)$ is the set of *non-variable positions in t*. By extension, the mapping of empty domain is called the *empty term* and denoted Λ.

Example 2.2 The term $t = f(a + x, h(f(a, b)))$:
— is the following mapping:

$\wedge$	$\mapsto$	f
1	$\mapsto$	$+$
1.1	$\mapsto$	a
1.2	$\mapsto$	x
2	$\mapsto$	h
2.1	$\mapsto$	f
2.1.1	$\mapsto$	a
2.1.2	$\mapsto$	b

— and is represented under a tree form as follow:



Exercice 1 — Define the notions of subterm and replacement on terms using their definition based on mapping.
Answer: a faire

In order to emphasize that $t[s]_\omega$ has been obtained by subterm replacement, the notation $t[\omega \hookleftarrow s]$ is also used sometimes to denote that in t the subterm $t_{|\omega}$ has been replaced by s.

Definition 2.4 The subterm relationship, denoted by $\trianglelefteq_{sub}$, is defined by $s \trianglelefteq_{sub} t$ if s is a subterm of t. The term s is a *proper subterm* also called a *strict subterm* of t if $s \trianglelefteq_{sub} t$ and $s \neq t$, which is denoted $\triangleleft^{sub}$.

2.2.4 Terms as functions

To any term t and to any subset $V = \{x_1, \dots, x_n\}$ of $\mathcal{V}ar(t)$ we can associate a function denoted $t(x_1, \dots, x_n)$ from $\mathcal{T}(\mathcal{F}, \mathcal{X})^n$ to $\mathcal{T}(\mathcal{F}, \mathcal{X})$ such that:

$$\begin{array}{rccl} t(x_1,\dots,x_n): & \mathcal{T}(\mathcal{F},\mathcal{X})^n & \to & \mathcal{T}(\mathcal{F},\mathcal{X}) \\ & t_1,\dots,t_n & \mapsto & t[x_i \hookleftarrow t_i]_{i=1,\dots,n} \end{array}$$

In particular, taking $V = \mathcal{V}ar(t)$ shows that any term t can be considered as a function.

Two extensions of the concept of term are useful: infinite terms and sorted terms.

2.2.5 Infinite terms

Infinite terms are simply infinite labeled trees or can be defined, following Section 2.2.3 as partial mappings having an infinite domain [CR80, AN80, Cou80, Cou83]. Infinite labeled trees with finitely many different subtrees are called *rational trees.*

2.2.6 Sorted terms

Sorted terms are obtained when function symbols, variables and terms are categorized into classes, called *sorts.*

Definition 2.5 A *many-sorted signature* denoted by Σ is given by a (denumerable) set of *sorts* S and a (denumerable) set of ranked function symbols $\mathcal{F}$. A function symbol f with *arity* $w = s_1, \dots, s_n \in S^*$ and *co-arity* (or value sort) s is written $f : w \mapsto s$.

Variables are also sorted and $x : s$ means that variable x has sort s. The set $\mathcal{X}_s$ denotes a set of variables of sort s generally supposed to be denumerable and $\mathcal{X} = \bigcup_{s \in S} \mathcal{X}_s$ is the set of many-sorted variables.

Many-sorted terms are built on many-sorted signatures and classified according to their sorts.

Definition 2.6 The set of terms of sort s, denoted $\mathcal{T}(\Sigma, \mathcal{X})_s$ is the smallest set containing $\mathcal{X}_s$ and any constant $a : s$ such that $f(t_1, \dots, t_n)$ is in $\mathcal{T}(\Sigma, \mathcal{X})_s$ whenever $f : s_1, \dots, s_n \mapsto s$ and $t_i \in \mathcal{T}(\Sigma, \mathcal{X})_{s_i}$ for $i \in [1..n]$.

The set of *many-sorted terms* $\mathcal{T}(\Sigma, \mathcal{X})$ is the family $\{\mathcal{T}(\Sigma, \mathcal{X})_s | s \in S\}$.

2.3 Substitutions

2.3.1 Definitions and elementary properties

A *substitution* is an application on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ uniquely determined by its image of variables. It is thus written out as $\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ when there are only finitely many variables not mapped to themselves. The application of a substitution $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ to a term is recursively defined as follows:

1. if t is a variable x_i for some $i = 1, \dots, n$, then $\sigma(t) = t_i$,
2. if t is a variable $x \neq x_i$ for all $i = 1, \dots, n$, then $\sigma(t) = t$,
3. if t is a term $f(u_1, \dots, u_k)$ with $u_1, \dots, u_k \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $f \in \mathcal{F}$, then $\sigma(t) = f(\sigma(u_1), \dots, \sigma(u_k))$.

Example 2.3 For example, applying $\sigma = \{(x \mapsto f(a, z))\}$ on the term $t = g(a, g(x, x))$ results in the term $\sigma(t) = g(a, g(f(a, z), f(a, z)))$.

Notation: We are mainly denoting substitutions by the Greek letters $\alpha, \beta, \gamma, \sigma$.

Definition 2.7 The *domain* of a substitution σ is the set of variables that are not trivially mapped to themselves:

$$\mathcal{D}om(\sigma) = \{x | x \in \mathcal{X} \text{ and } \sigma(x) \neq x\}$$

and the set of variables introduced by σ is called its *range*, defined by:

$$\mathcal{R}an(\sigma) = \bigcup_{x \in \mathcal{D}om(\sigma)} \mathcal{V}ar(\sigma(x))$$

When $\mathcal{R}an(\sigma) = \emptyset$, σ is called a *ground substitution.* When $\rho(s)$ is an instance of s that belongs to $\mathcal{T}(\mathcal{F})$, ρ is called a *ground instantiation* of s. The substitution whose domain is empty is denoted Id.

We denote by $\sigma_{|W}$ the *restriction* of the substitution σ to the subset W of $\mathcal{X}$, defined as follows:

$$\begin{array}{rcll} \sigma_{|W}(x) & = & \sigma(x) & \text{if } x \in W \\ & = & x & \text{else.} \end{array}$$

Restrictions extend to sets: if $\mathcal{S}$ is a set of substitutions, then:

$$\mathcal{S}_{|W} = \{\sigma_{|W} | \sigma \in \mathcal{S}\}.$$

The *composition of substitutions* α and β is denoted by $\circ$ or . or simply by the juxtaposition $\beta\alpha$ and defined as usual for mapping as $\beta\alpha(x) = \beta(\alpha(x))$. The set of all substitutions of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is denoted $Subst^{\mathcal{F}\mathcal{X}}$ or $Subst$ when $\mathcal{F}$ and $\mathcal{X}$ are clear from the context.

The addition of two substitutions σ and ρ can also be defined when their domains are disjoint ($\mathcal{D}om(\sigma) \cap \mathcal{D}om(\rho) = \emptyset$):

$$\begin{array}{rl} (\sigma + \rho)(x) = & \sigma(x) \text{ if } x \in \mathcal{D}om(\sigma) \\ & \rho(x) \text{ if } x \in \mathcal{D}om(\rho) \end{array}$$

The following elementary properties of substitutions are both basic and useful.

Proposition 2.1

1. The composition of substitutions is associative.
2. For all subset of variables V of $\mathcal{X}$, for all term t and for all substitution σ:
$$\mathcal{V}ar(t) \subseteq V \Rightarrow \sigma(t) = \sigma_{|V}(t).$$
3. For all substitutions σ and σ' and for all term t,
$$\sigma(t) = \sigma'(t) \Leftrightarrow \sigma_{|\mathcal{V}ar(t)} = \sigma'_{|\mathcal{V}ar(t)}.$$

Idempotent substitutions are quite important since they enjoy useful properties that make the definition of concepts much simpler and proofs easier. In particular we will see that for unification one can restrict without loss of generality to idempotent unifiers.

Definition 2.8 A substitution σ is *idempotent* if $\sigma.\sigma = \sigma$.

The idempotent substitutions can be characterized nicely from their domain.

Lemma 2.1 The following properties are equivalent:

1. σ is idempotent,
2. $\mathcal{R}an(\sigma) \cap \mathcal{D}om(\sigma) = \emptyset$.

Proof: If the range and the domain of a substitution are disjoint then clearly the substitution is idempotent. Conversely, assume the σ is idempotent. Then suppose that there exists $x \in \mathcal{R}an(\sigma) \cap \mathcal{D}om(\sigma)$. By definition of the range, there exists y such that $\sigma(y) = t[x]$ for some context $t[]$. Then we have $\sigma\sigma(y) = \sigma(t[x]) \neq t[x]$ since $x \in \mathcal{D}om(\sigma)$, and this contradicts the hypothesis that σ is idempotent. $\Box$

Definition 2.9 Let ξ be the finite substitution $\{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$ where $y_1, \ldots, y_n$ are distinct variables. Then ξ is called a *permutation* if $\mathcal{R}an(\xi) = \mathcal{D}om(\xi)$ and a *renaming* if $\mathcal{R}an(\xi) \cap \mathcal{D}om(\xi) = \emptyset$.

We can now prove that permutations are *invertible* substitutions i.e. satisfy $\xi.\xi^{-1} = Id = \xi^{-1}.\xi$.

Lemma 2.2 A substitution is invertible iff it is a permutation.

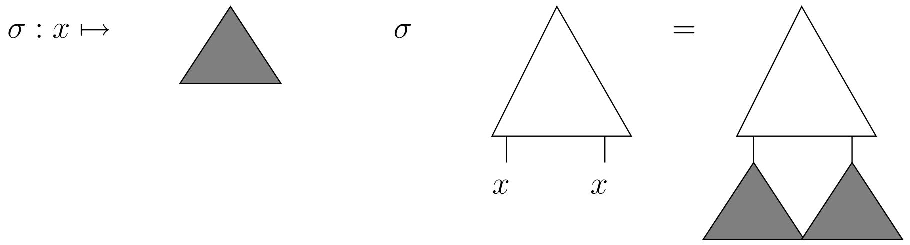
Proof: If $\xi = \{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$ is a permutation then $\mu = \{y_1 \mapsto x_1, \ldots, y_n \mapsto x_n\}$ is clearly its inverse, ie. $\xi.\mu = \mu.\xi = Id$.
Assume now that $\xi = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is a bijective substitution. If there exists $x \in \mathcal{D}om(\xi)$ such that $\xi(x) = f(\ldots)$ then $x = \xi^{-1}(f(\ldots))$ which is impossible. Thus $\mathcal{R}an(\xi) \subseteq \mathcal{X}$ and ξ should be of the form $\xi = \{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$. Since ξ is injective, all the y_i are distinct. Finally note that if $x \mapsto y \in \xi$ then $y \mapsto x$ belongs to ξ^{-1} and thus $\mathcal{R}an(\xi) \subseteq \mathcal{D}om(\xi^{-1}) \subseteq \mathcal{R}an(\xi)$, which proves that ξ is a permutation. □

Example 2.4 The substitution $\sigma = \{(x \mapsto f(a, z))\}$ is not a renaming, but $\xi = \{(x \mapsto y_1), (z \mapsto y_2)\}$ is one. The substitution $\mu = \{(x \mapsto y), (y \mapsto x)\}$ is a permutation, but not a renaming. Note that $\xi' = \{(y_1 \mapsto x), (y_2 \mapsto z)\}$ is *not* the inverse of ξ since $\xi.\xi'(x) = \xi(\xi'(x)) = \xi(x) = y_1$.

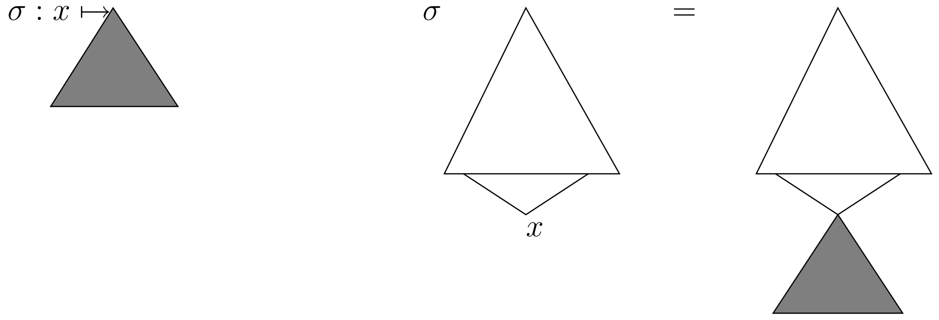
Lemma 2.3 If a finite substitution σ is injective and maps variables to variables then it is bijective.

Exercice 2 — Show that the last result is false if the substitution is not finite (i.e. $\mathcal{D}om(\sigma)$ is not finite).
Answer: Take $\mathcal{X} = \{x_i\}_{i \in \mathbf{N}}$ and $\sigma(x_i) = x_{2i}$. This substitution is injective but not surjective.

Representing term by trees leads to the following result of the application of a substitution.



When using a dag representation for terms, the substituted term may have a more compact form when it is not linear. In the previous example we get:



2.3.2 Term subsumption

Definition 2.10 A term t is an *instance* of a term s if $t = \rho(s)$ for some substitution ρ; in that case we write $s \leq t$ and say that s is *more general* than t or that s *subsumes* t. We call ρ a *match* from s to t. The relation $\leq$ is a quasi-ordering[1] on terms called *subsumption*, whose associated equivalence $\equiv$ and strict ordering $<$ are respectively called *subsumption equivalence* and *strict subsumption*.

Notice that the subsumption ordering is not stable by context: $s \leq t \not\Rightarrow f(t_1, \ldots, s, \ldots, t_n) \leq f(t_1, \ldots, t, \ldots, t_n)$, as shown by the following example: $x \leq a$ but $f(x, x) \not\leq f(x, a)$. It is not stable by substitution too: $s \leq t \not\Rightarrow \sigma s \leq \sigma t$ since $x \leq a$ but $(x \mapsto b)x \not\leq (x \mapsto b)a$.

Example 2.5 One can check that $f(x, y) \leq f(f(a, b), h(y))$.

Another useful ordering on terms is obtained as the combination of the subterm and the subsumption orderings.

Definition 2.11 The *encompassment ordering* denoted $\sqsubseteq$ is defined by $s \sqsubseteq t$ if a subterm of t is an instance of s i.e.

$$s \sqsubseteq t \Leftrightarrow \exists \sigma \in Subst, \exists p \in \mathcal{D}om(t), \sigma(s) = t_{|p}$$

The associated strict ordering is denoted by $\sqsubset$.

The main properties of the subsumption on terms are studied in Chapter 3. In particular, up to renaming, the subsumption ordering is well founded as we will show in Chapter 3.

[1]see definition on page 53

2.3.3 Substitution subsumption

Subsumption extends to substitutions as follows:

Definition 2.12 A substitution τ is an *instance* on $V \subseteq \mathcal{X}$ of a substitution σ, written $\sigma \leq^V \tau$, read σ is *more general* over V than τ, when:

$$\sigma \leq^V \tau \Leftrightarrow \exists \rho, \forall x \in V, \tau(x) = \rho(\sigma(x)).$$

This is also denoted $\tau =^V \rho\sigma$. The relation $\leq^V$ is a quasi-ordering on substitutions, also called *subsumption.* V is omitted when equal to $\mathcal{X}$.

Exercice 3 — Compare the substitutions $\sigma = \{(x \mapsto z), (y \mapsto z)\}$ and $\alpha = \{(x \mapsto y)\}$ respectively on $\mathcal{X}$ the set of all variables and on $\{x\}$.
Answer: We have $\alpha \leq^{\mathcal{X}} \sigma$ as well as $\alpha \leq^{\{x\}} \sigma$, since $\{y \mapsto z\}\alpha = \{x \mapsto z, y \mapsto z\}$ either on $\mathcal{X}$ or $\{x\}$.
But $\sigma \leq^{\{x\}} \alpha$ since $(z \mapsto y).\sigma = \alpha$ only on $\{x\}$ and not on $\mathcal{X}$.
Exercice 4 — Compare the substitutions $\{(x \mapsto f(y, a))\}$ and $\{(x \mapsto f(a, a))\}$ on $\mathcal{X}$ and $\{x\}$.
Answer: On $\mathcal{X}$ they are not comparable. On $\{x\}$ we have $\{(x \mapsto f(y, a))\} \leq^{\{x\}} \{(x \mapsto f(a, a))\}$.
In comparing substitutions, the restriction to the right set of variables is a quite important condition which has been sometime forgotten. The previous exercises show its importance. It is also emphasized in [SL90] and [Baa91].
Exercice 5 — Prove that when there is at least a symbol of arity two, the above definition of the subsumption ordering on substitutions is equivalent to the following one:

$$\sigma \leq^V \tau \Leftrightarrow \forall x \in V \quad \sigma(x) \leq \tau(x).$$

Answer: See [Hue76].

Similarly to terms, the equivalence $\equiv^V$ and the strict ordering $<^V$ on substitutions are respectively called *subsumption equivalence* and *strict subsumption.* It is easily seen that σ and τ are subsumption equivalent on V iff ρ is a one to one mapping from $Ran(\sigma_{|V})$ to $Ran(\tau_{|V})$.

Example 2.6 Note that $\{x \mapsto z, y \mapsto succ(z)\}$ and $\{y \mapsto succ(x)\}$ are subsumption equivalent on $\{x, y\}$ under the renaming $\{x \mapsto z\}$.

The following main property is a consequence of the well-foundedness of the subsumption ordering on terms. It does not hold in general for equational theories.

Theorem 2.1 *Up to renaming, the subsumption ordering on substitutions is well-founded.*

Proof: [Ede85] □

Extensive studies of substitutions can be found in [Hue76] and [Ede85].

2.4 Equational logic

2.4.1 Syntax

In the logic of equality, formulas are built from first-order terms and the equality predicate.

Definition 2.13 A pair of two terms $\{l, r\}$ is called an *equational axiom* or *equality* and denoted $(l = r)$, or an *equation* in which case it is denoted by $l =^? r$. The variables of an equational axiom are assumed to be universally quantified. When quantification must be explicit, it is written $(\forall X, l = r)$ where $\mathcal{V}ar(l) \cup \mathcal{V}ar(r) \subseteq X$.

2.4.2 Deduction system

From a set of axioms, new equalities can be deduced via inference rules. A deduction system for equational deduction is given in Figure 2.1.

Definition 2.14 Given a set of equational axioms E and a set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$, the *equational theory of* E, denoted $\mathcal{TH}(E)$, is the set of equalities that can be obtained, starting from E, by applying the inference rules given in Figure 2.1.

1. Reflexivity		$\vdash$	$(t = t)$
2. Symmetry	$(t = t')$	$\vdash$	$(t' = t)$
3. Transitivity	$(t = t'), (t' = t'')$	$\vdash$	$(t = t'')$
4. Congruence	$\{(t_i = t'_i) \mid i = 1, \ldots, n\}$	$\vdash$	$(f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n))$ if $f \in \mathcal{F}_n$
5. Substitutivity	$(t_1 = t_2)$	$\vdash$	$(\sigma(t_1) = \sigma(t_2))$ if $\sigma \in \mathit{Subst}$

Figure 2.1: The rules of equational deduction

Notation: We write
$$E \vdash s = t \text{ if } (s = t) \in \mathcal{TH}(E).$$

Exercice 6 — With the signature $\mathcal{F} = \{a, b, f, h\}$ where the arity are respectively 0, 0, 2, 1, describe the equational theory of $E = \{a = b\}$.
Answer: {a = a, b = b, a = b, h(a) = h(a), h(b) = h(b), h(a) = h(b), ...}

A more compact inference system is the so-called *replacement of equals by equals*:

Definition 2.15 Given a set E of axioms, we write $s \longleftrightarrow_E t$ if $s|_\omega = \sigma(l)$ and $t = s[\sigma(r)]_\omega$ for some position ω in $\mathcal{D}om(s)$, substitution σ and equality $l = r$ (or $r = l$) in E.

Example 2.7 If $E = \{a = b\}$ then $h(a) \longleftrightarrow_E h(b)$.
For $E = \{f(x, x) = x\}$ then $f(a, b) \longleftrightarrow_E f(f(a, a), b)$.

The *provability relation* of E is the reflexive transitive closure of the above symmetric relation, and is denoted $\overset{*}{\longleftrightarrow}_E$. It is a congruence relation.
Notation: The quotient of the set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ by $\overset{*}{\longleftrightarrow}_E$ is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{X})/E$.

The following theorem states the equivalence between the two inference systems for equality deduction.

Theorem 2.2 (Birkhoff) *[Bir35]*
$$E \vdash s = t \text{ iff } s \overset{*}{\longleftrightarrow}_E t.$$

2.4.3 Models

The models we are interested in are non-empty sets with operations, called algebras. In this section, and in most of this work, we only consider *mono-sorted algebras* which consist in only one sort: they are also called *unsorted algebras*.

Definition 2.16 For a given unsorted signature $\mathcal{F}$, for a set A and for a function symbol f of arity $n \geq 0$, an *interpretation* ι of f in A is a function $\iota(f)$ from A^n to A. For a set of function symbols $\mathcal{F}$, an interpretation ι of $\mathcal{F}$ in the set A is a mapping associating to each function symbol in $\mathcal{F}$ an interpretation in A.

For example if $A = \mathbf{N}$ and taking $\mathcal{F} = \{+, *, 0\}$ with arity$(+)$ = arity$(*)$ = 2 and arity(0) = 0 the following mappings are two interpretations:

$$\iota_1 = \begin{cases} + \mapsto +_\mathbf{N} \\ * \mapsto *_\mathbf{N} \\ 0 \mapsto 0_\mathbf{N} \end{cases} \quad \iota_2 = \begin{cases} + \mapsto *_\mathbf{N} \\ * \mapsto +_\mathbf{N} \\ 0 \mapsto 1_\mathbf{N} \end{cases}$$

Definition 2.17 For a set $\mathcal{F}$ of function symbols, an $\mathcal{F}$*-algebra* $\mathcal{A}$ is given by a non-empty set A (called the *carrier* of the algebra) together with an interpretation ι of $\mathcal{F}$. It is denoted $\mathcal{A} = (A, \iota(\mathcal{F}))$ and $\iota(\mathcal{F})$ is often written $\mathcal{F}_\mathcal{A}$ and the interpretation of a function symbol f written $f_\mathcal{A}$.

Example 2.8 A set may be regarded as an algebra with no operation.

Example 2.9 Let $\mathcal{F} = \{0, s, +\}$ with arities 0, 1, 2 respectively. Let us choose as carrier the set of natural numbers $\mathbf{N}$ and for interpretation of the functions: $0_\mathbf{N}$ which is the natural 0, $s_\mathbf{N}$ which is the successor function in the naturals and $+_\mathbf{N}$ which is the usual addition on naturals. Then $(\mathbf{N}, \{0_\mathbf{N}, s_\mathbf{N}, +_\mathbf{N}\})$ is an $\{0, s, +\}$-algebra.

For a set of function symbols $\mathcal{F}$ and a set a variable $\mathcal{X}$, take as carrier $\mathcal{T}(\mathcal{F},\mathcal{X})$ and for interpretation of each symbol in $\mathcal{F}$ the symbol itself i.e. $f_{\mathcal{T}(\mathcal{F},\mathcal{X})}(t_1,\dots,t_n) = f(t_1,\dots,t_n)$. Then $(\mathcal{T}(\mathcal{F},\mathcal{X}),\mathcal{F})$ is clearly a $\mathcal{F}$-algebra.

Definition 2.18 Given two $\mathcal{F}$-algebras $\mathcal{A} = (A,\mathcal{F}_{\mathcal{A}})$ and $\mathcal{B} = (B,\mathcal{F}_{\mathcal{B}})$, a mapping θ from A to B such that:

$$\forall f \in \mathcal{F}, \forall a_1,\dots,a_n \in A, \theta(f_{\mathcal{A}}(a_1,\dots,a_n)) = f_{\mathcal{B}}(\theta(a_1),\dots,\theta(a_n))$$

is called an *homomorphism* (an *endomorphism* if $\mathcal{A} = \mathcal{B}$). A bijective endomorphism is an *isomorphism.*

Example 2.10 A *trivial algebra* is an algebra $(A,\mathcal{F}_{\mathcal{A}})$ whose carrier has only one element. Then there is only one mapping from A^n to A for arbitrary n. Therefore all trivial $\mathcal{F}$-algebras are isomorphic.

Substitution on $\mathcal{T}(\mathcal{F},\mathcal{X})$ are endomorphisms of $\mathcal{T}(\mathcal{F},\mathcal{X})$, which justifies their fundamental property: for any terms $t_1,\dots,t_n \in \mathcal{T}(\mathcal{F},\mathcal{X})$ and symbol $f \in \mathcal{F}$:

$$\sigma(f(t_1,\dots,t_n)) = f(\sigma(t_1),\dots,\sigma(t_n)).$$

Definition 2.19 Let $\mathcal{A}$ be an algebra with carrier A and $\mathcal{X}$ a set whose elements are called variables. An *assignment* from $\mathcal{X}$ to $\mathcal{A}$ is a mapping ν from $\mathcal{X}$ to A.

An assignment ν from $\mathcal{X}$ to $\mathcal{A}$ extends to a morphism also denoted ν from $\mathcal{T}(\mathcal{F},\mathcal{X})$ to $\mathcal{A}$ by inductively defining the image by ν of a non-variable term t by: $\nu(f(t_1,\dots,t_n)) = f_{\mathcal{A}}(\nu(t_1),\dots,\nu(t_n))$.

A non-empty class $\mathcal{C}$ of $\mathcal{F}$-algebras is called a *variety* when it is closed under the operations of sub-algebra, homomorphic image, and direct product. A variety has the nice property of have a canonical representant called the free algebra.

Definition 2.20 Let $\mathcal{C}$ be a non-empty class of $\mathcal{F}$-algebras and $\mathcal{X}$ a set of variables. A *$\mathcal{C}$-free $\mathcal{F}$-algebra* (also simply called *free algebra*) over $\mathcal{X}$ is any $\mathcal{F}$-algebra $\mathcal{L} = (L,\mathcal{F}_{\mathcal{L}})$ such that:

- $\mathcal{L} \in \mathcal{C}$,
- $\mathcal{X} \subseteq L$,
- for any $\mathcal{F}$-algebra $\mathcal{A}$ in $\mathcal{C}$ and any assignment $\nu : \mathcal{X} \to \mathcal{A}$, there exists a unique homomorphism $\phi : \mathcal{L} \to \mathcal{A}$ such that ϕ and ν agree on $\mathcal{X}$, i.e. are such that $\forall x \in \mathcal{X}, \phi(x) = \nu(x)$.

This can be pictured as follows:

$$\begin{tikzcd}
\mathcal{X} \arrow[r, "\subseteq"] \arrow[d, "\nu"'] & \mathcal{L} \arrow[dl, dotted, "\phi"] \\
\mathcal{A} &
\end{tikzcd}$$

It is easy to see that, when it exists, a free algebra over a set $\mathcal{X}$ is unique up to an isomorphism.

Definition 2.21 An algebra $\mathcal{I}$ in a class $\mathcal{C}$ of $\mathcal{F}$-algebras is *$\mathcal{C}$-initial* (also simply called *initial algebra*) if for any algebra $\mathcal{A}$ in $\mathcal{C}$, there exists a unique homomorphism $\phi : \mathcal{I} \to \mathcal{A}$.

Note that by definition, the initial algebra can also be seen as the free algebra over the empty set of variables. When it exists, it is of course unique up to an isomorphism.

The first well known result of G. Birkhoff on this topics is that the free (and thus also initial) algebra of a variety always exists:

Theorem 2.3 *[Bir35] For any set of variable $\mathcal{X}$, the free algebra of any variety $\mathcal{C}$ always exists.*

Proposition 2.2 The $\mathcal{F}$-algebra $\mathcal{T}(\mathcal{F},\mathcal{X})$ is the free algebra over $\mathcal{X}$ of the class of all the $\mathcal{F}$-algebras.

Varieties can also be characterized by a set of axioms. Let us now introduce the notion of model and validity in the particular case of equality axioms i.e. universaly quantified equalities of the form $\forall \mathcal{V}ar(l,r), l = r$ and existentially quantified axioms i.e. $\forall(\mathcal{V}ar(l,r) \setminus V), \exists V, l = r$ where $V \subseteq \mathcal{V}ar(l,r)$. Notice that by skolemization, the latter can always be reduced to the former. The universal quantifier is often not explicitly mentioned.

Definition 2.22 An algebra $\mathcal{A}$ is a *model* of an axiom $s = t$, if for any assignment ν of the variables in s and t, $\nu(s) = \nu(t)$.

An algebra $\mathcal{A}$ is a *model* of an axiom $\exists V, s = t$, if for any assignment ν of the variables in s and t except those in V, there exists an assignment μ such that $\mu\nu(s) = \mu\nu(t)$.

Let E be a set of equality axioms. An algebra $\mathcal{A}$ is a *model* of E if it is a model of all the axioms in E. We also say that the equality $s = t$ is *valid* in $\mathcal{A}$, and this is denoted by $\mathcal{A} \models s = t$.

$\mathcal{M}od(E)$ denotes the set of models of E. The class of models of a set of equalities E is characterized by the notion of variety, as follow:

Theorem 2.4 *A class $\mathcal{C}$ of algebras is the class of models of a set of equalities E iff it is a variety, i.e. when it is closed under direct product, homomorphic image and sub-algebra.*

It can be proved that $\mathcal{T}(\mathcal{F}, \mathcal{X})/E$ is a model of E. Moreover,

Theorem 2.5 *(Birkhoff) [Bir35] For any set of axioms E, for any terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$,*

$$\mathcal{M}od(E) \models s = t \text{ iff } \mathcal{T}(\mathcal{F}, \mathcal{X})/E \models s = t \text{ iff } E \vdash s = t$$

Notation:

We write $s =_E t$ iff $\mathcal{M}od(E) \models s = t$. Thanks to Theorems 2.2 and 2.5, the notations $s =_E t$ and $s \overset{*}{\longleftrightarrow}_E t$ may be used interchangeably.

In the class of algebras that are models of E, the free algebra over $\mathcal{X}$ is (isomorphic to) the algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})/E$. In the class of algebras that are models of E, the initial algebra is (isomorphic to) the algebra $\mathcal{T}(\mathcal{F})/E$.

Example 2.11 Let $(G, +)$ be a commutative semi-group (see 2.4.7) and define for $g \in G$:

$$\begin{aligned} g^1 &= g \\ g^{n+1} &= g + g^n. \end{aligned}$$

Then for arbitrary, but fixed, natural number n,

$$f_n : g \mapsto g^n$$

is an endomorphism of G.

Term algebras are specially interesting because of their large representativity, but this is still limited to the class of term generated algebras:

Definition 2.23 A $\mathcal{F}$-algebra $\mathcal{A} = (A, \iota(\mathcal{F}))$ is *term generated* when for all elements a in A there exists a term t in $\mathcal{T}(\mathcal{F})$ such that $\iota(t) = a$.

All algebras are not term generated, for example if we consider $\mathcal{F} = \{+, *, succe, 0\}$, then the $\mathcal{F}$-algebra of the reals $\mathcal{R}$ where $+, *, succe, 0$ are interpreted with their usual meaning and which domain is the set of all reals $\mathbf{R}$ is not term generated since e.g. $\sqrt{2}$ can not be finitely expressed using only the operators $+, *, succe, 0$.

2.4.4 The subsumption ordering modulo

Definition 2.24 For a set of equational axioms A, the *subsumption ordering modulo A* on terms, denoted $\leq_A$, is defined as:

$$t \leq_A t' \Leftrightarrow \exists \eta \in Subst, \eta(t) =_A t'.$$

Exercice 7 — Prove, as asserted in the last definition, that $\leq_A$ is a preorder for any set of equational axioms A.
Answer: It is enough to check that $\leq_A$ is a transitive and reflexive relation.

As a particular case of this definition we get:

$$t \leq_\emptyset t' \Leftrightarrow \exists \eta \in Subst, \eta(t) = t'.$$

When there is no ambiguity, $\leq_\emptyset$ is also denoted $\leq$ since in this case we get the same definition as previously (Definition 2.10).

This preorder $\leq_A$ is not compatible with the term structure: namely it is false in general that if $t \leq_A t'$ and $f \in \mathcal{F}_n$ then $f(t_1, \ldots, t, \ldots, t_n) \leq_A f(t_1, \ldots, t', \ldots, t_n)$ for any terms $t_1, \ldots, t_n$.

We should also notice that substitutions are not monotonic mapping for the subsumption ordering:

$$t \leq_A t' \not\Rightarrow \sigma(t) \leq_A \sigma(t').$$

Exercice 8 — Illustrate with some examples the two remarks above.
Answer:

- $x \leq_\emptyset a$ but $f(x, x) \not\leq_\emptyset f(x, a)$.
- $x \leq_\emptyset a$ but $x.(x \mapsto b) \not\leq_\emptyset a.(x \mapsto b)$.

This preorder can be extended to substitutions in the following way; suppose that V is a subset of the set of variables $\mathcal{X}$ then:

$$\sigma \leq_A^V \sigma' \Leftrightarrow \exists\mu\, \forall x \in V\; \mu.\sigma(x) = \sigma'(x)$$

In this case the substitution σ is said *more general modulo* A than σ' on the set of variables V.
Exercice 9 — For the empty theory we have seen that $\sigma(t) = \sigma'(t) \Leftrightarrow \sigma_{|\mathcal{V}ar(t)} = \sigma'_{|\mathcal{V}ar(t)}$. (see Proposition 2.1). Give an example of an equational theory A such that this property is not satisfied when $=$ is replaced by $=_A$.
Answer: Assume that $A = \{f(y) = 0\}$ then the substitutions $\sigma = \{(x \mapsto a)\} and \sigma' = Id$ makes the term $f(x)$ A-equal: $\sigma(f(x)) = f(a) =_A f(x) = \sigma'(f(x))$.

Proposition 2.3

1. For all equational theory A, for all term t and for all substitution σ:

$$\mathcal{D}om(\sigma) \cap \mathcal{V}ar(t) = \emptyset \Rightarrow \sigma(t) =_A t.$$

The converse is true if $A = \emptyset$.

2. For all substitutions α, β and for all set of variables V_1 and V_2, if $\alpha \leq_A^{V_1} \beta$ and if $V_2 \subseteq V_1$ then $\alpha \leq_A^{V_2} \beta$

And idempotent substitutions have a simple interesting duplication property:

Lemma 2.4 If σ is an idempotent substitution then for all substitution α,

$$\sigma \leq_A \alpha \Leftrightarrow \alpha.\sigma =_A \alpha.$$

Proof: The implication $\Leftarrow$ is clear. Conversely, if $\sigma \leq_A \alpha$ then there exists ρ such that $\rho.\sigma =_A \alpha$ and thus:

$$\alpha.\sigma =_A \rho.\sigma.\sigma =_A \rho.\sigma =_A \alpha.$$

□

Exercice 10 — Give an example of an infinite strictly decreasing sequence of substitutions in an equational theory.
Answer: A faire

One should not confuse the subsumption equivalence modulo and equality modulo as shown as follows:

Example 2.12 Let f be an idempotent symbol, $A = \{x = f(x, x)\}$ then we have: $x \leq_A f(y, z)$ (using substitution $\eta = \{(x \mapsto f(y, z)\})$ and $f(y, z) \leq_A x$ (using substitution $\eta' = \{(y \mapsto x), (z \mapsto x)\}$), so that $x \equiv_A f(y, z)$ but of course $x \neq_A f(y, z)$.

2.4.5 Satisfiability

Following the mathematical usage, we use the words "equational axiom" or "equality" when the formula is supposed true. In this case, we are in particular interested in either:

1. specifying the properties that an algebra or class of algebras should satisfy, or
2. solving the problem of the validity of an equality formula in an algebra or class of algebras.

We use the word "equation" instead when we are interested in the problem of finding values to be given to the free variables appearing in the terms of the equation in such a way that the substituted equation becomes a valid equality.

Short name	Name	Definition
$A(f)$	Associativity	$f(f(x,y),z) = f(x,f(y,z))$
$C(f)$	Commutativity	$f(x,y) = f(y,x)$
$Dr(f,g)$	Right Distributivity	$f(g(x,y),z) = g(f(x,z),f(y,z))$
$Dl(f,g)$	Left Distributivity	$f(z,g(x,y)) = g(f(z,x),f(z,y))$
$D(f,g)$	Distributivity	$Dl(f,g) \cup Dr(f,g)$
$E(h,*)$	Endomorphism	$h(x*y) = h(x)*h(y)$
$UE(h,e)$	Unit Endomorphism	$h(e) = e$
$AE(h,*)$	Anti-endomorphism	$h(x*y) = h(y)*h(x)$
$H(h,*,+)$	Homomorphism	$h(x*y) = h(x)+h(y)$
$I(f)$	Idempotency	$f(x,x) = x$
$Iv(h)$	Involution	$h(h(x)) = x$
$InvR(*,e)$	Right Inverse	$x*i(x) = e$
$InvL(*,e)$	Left Inverse	$i(x)*x = e$
$Sr(f,g,h)$	Right Simplification	$f(g(x,y),h(y)) = x$
$Sl(f,g,h)$	Left Simplification	$f(h(y),g(y,x)) = x$
$Ur(*,e)$	Right Unit	$x*e = x$
$Ul(*,e)$	Left Unit	$e*x = x$
$Ar(*,e)$	Absorb Right	$x*e = e$
$Al(*,e)$	Absorb Left	$e*x = e$
$Cr(f)$	Right Commutativity	$f(f(x,y),z) = f(f(x,z),y)$
$Cl(f)$	Left Commutativity	$f(x,f(y,z)) = f(y,f(x,z))$
$F(f,g)$	Factorize	$f(x,f(y,z)) = f(g(x,y),z)$
$L(*)$	Lie brackets	$(x*y)*z = z*(y*x)$
$T(f,g)$	Transitivity	$f(g(x,y),g(y,z)) = f(g(x,y),g(x,z))$

Figure 2.2: Table of usual equational axioms

Definition 2.25 An equation $s =^? t$ is *satisfiable* in an algebra $\mathcal{A}$ if there exists an assignment ν of values to the variables of s and t for which $\nu(s) = \nu(t)$.

This definition is a particular case of the more general definition of an *equational problem.*

Definition 2.26 Let $\mathcal{F}$ be a set of function symbols, $\mathcal{X}$ be a set of variables, and $\mathcal{A}$ be an $\mathcal{F}$-algebra. An ($< \mathcal{F}, \mathcal{X}, \mathcal{A} >$-)*equational problem* is any set $P = \{s_i =^?_{\mathcal{A}} t_i\}_{i \in I}$ of equations, such that s_i and t_i are terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A *solution* of P is any homomorphism h from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ to $\mathcal{A}$ such that $\forall i \in I, h(s_i) = h(t_i)$, i.e. s_i and t_i are mapped to the same value in $\mathcal{A}$ by the homomorphism h.
Two equational problems P and P' are said to be *equivalent* if they have the same set of solutions.

We will investigate two cases, when $\mathcal{A}$ is the term algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})$ (syntactic unification) and when $\mathcal{A}$ is the quotient algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})/E$ (semantic unification), for some a priori given set E of equational axioms. As usual, we reserve the word "unification" for these two cases, and speak of "equations solving" otherwise.

2.4.6 Word problem

One of the simplest problem concerning equational logic is to decide, for a given equational theory $\mathcal{TH}(E)$, if two terms are E-equal or not.

Definition 2.27 Let $\mathcal{TH}(E)$ be an equational theory built on the term algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and t, t' be two terms. The *word problem* consists to decide if the equality $t = t'$ holds in $\mathcal{TH}(E)$, that is if $E \models t = t'$.

If the equational theory is recursive, i.e. if one can decide if a given pair of terms is an axiom or not, then the word problem is semi-decidable.

2.4.7 A theory directory

Let us first give in Table 2.2 the usual definition of commonly used equational axioms. Combination of these axioms give well known theories described in Table 2.3.
Here are some others well-known equational theories:

	Name	Definition
AG	Abelian Group	$A(*), C(*), InvR(*, e), Ul(*, e)$
QG	Quasi group	$Sl(., \backslash, h), Sl(., \backslash, h), Sr(., /, h), Sr(., /, h)$ with $h(x) = x$.
BR	Boolean Ring	$A(+), C(+), Ur(+, 0), InvR(+, 0),$ $A(*), C(*), Ur(*, 1),\ I(*),\ Dr(*, +)$
PA	Primal Algebras	Algebras such that every finitary function on the algebra can be expressed as a term [Grä79, Nip88]
	Minus	$AE(-, +), Iv(-)$
BST	Binary signed trees	$AE(-, +), Iv(-), Sr(+, +, -), Sl(+, +, -)$
FH	Fages & Huet	$Ul(*, e), f(x * y) = f(y)$
DlAU	Arnborg & Tiden	$Dl(f, g), A(g), Ur(f, e), Ul(f, e)$
CCC	Cartesian Closed Category	$A(*), C(*), Ur(*, 1), Ul(\Rightarrow, 1), Ar(\Rightarrow, 1), F(\Rightarrow, *), Dl(\Rightarrow, *)$

Figure 2.3: Table of some theories

Semi-groups: A *semi-group* is an algebra $(S, +)$ with a binary operation $+$ which is associative, i.e. satisfies:

$$(x + y) + z = x + (y + z).$$

A semi-group is Abelian or commutative if it satisfies in addition

$$x + y = y + x.$$

Monoids: A *monoid* is an algebra $(M, +, 0)$ with a binary operation $+$ and a nullary operation 0 that satisfies

$$\begin{aligned} (x + y) + z &= x + (y + z) \\ x + 0 &= x \\ 0 + x &= x \end{aligned}$$

where x, y, z are universally quantified variables.

Groups: A *group* $(G, +, i, 0)$ is an algebra with a binary operation $+$, a unary operation i and a nullary operation 0 that satisfies

$$\begin{aligned} (x + y) + z &= x + (y + z) \\ x + 0 &= x \\ 0 + x &= x \\ x + i(x) &= 0 \\ i(x) + x &= 0 \end{aligned}$$

where x, y, z are universally quantified variables. A group is Abelian or commutative if it satisfies in addition

$$x + y = y + x.$$

Rings: A *ring* is an algebra with binary operations $+$ and $*$, a unary operation i and nullary operations 0 and 1, such that $(R, +, i, 0)$ is an Abelian group, $(R, *, 1)$ is a semi-group, and

$$\begin{aligned} x * (y + z) &= (x * y) + (x * z) \\ (x + y) * z &= (x * z) + (y * z) \end{aligned}$$

where x, y, z are universally quantified variables. A ring is Abelian or commutative if it satisfies in addition

$$x * y = y * x.$$

Lattices: A *lattice* is an algebra $(L, \wedge, \vee)$, with binary operations $\wedge$ and $\vee$, such that

$$\begin{aligned} x \vee y &= y \vee x \\ x \wedge y &= y \wedge x \\ x \vee (y \vee z) &= (x \vee y) \vee z \\ x \wedge (y \wedge z) &= (x \wedge y) \wedge z \\ x \vee x &= x \\ x \wedge x &= x \\ x &= x \vee (x \wedge y) \\ x &= x \wedge (x \vee y) \end{aligned}$$

where x, y, z are universally quantified variables. The lattice is said distributive if it satisfies in addition

$$\begin{aligned} x \vee (y \wedge z) &= (x \vee y) \wedge (x \vee z) \\ x \wedge (y \vee z) &= (x \wedge y) \vee (x \wedge z). \end{aligned}$$

Boolean Algebras: A *boolean algebra* is an algebra $(B, \wedge, \vee, \neg, 0, 1)$, with binary operations $\wedge$ and $\vee$, a unary operation $\neg$ and two nullary operations 0 and 1, such that $(B, \wedge, \vee)$ is a distribute lattice and in addition

$$\begin{aligned} x \vee 1 &= 1 \\ x \wedge 0 &= 0 \\ x \vee (\neg x) &= 1 \\ x \wedge (\neg x) &= 0. \end{aligned}$$

Boolean Rings: A *boolean ring* $(B, \wedge, \oplus, \neg, 0, 1)$ is a commutative ring with identity such that

$$\begin{aligned} x \oplus x &= 0 \\ x \wedge x &= x \end{aligned}$$

where x is a universally quantified variable.

2.4.8 A morphological classification of theories

As one may have remarked in the section above, many theories share common properties based on the form of their axioms. We will now review the most commonly used classes of equational theories.

Let $s = t$ be an axiom of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. It is:

- *collapsing* if $s \in \mathcal{V}ar(t)$ and t is a proper term,
- *regular* if $\mathcal{V}ar(s) = \mathcal{V}ar(t)$,
- *subterm collapsing* if s is a proper subterm of t,
- *Permutative* if for all symbol f in $\mathcal{X} \cup \mathcal{F}$: $|s|_f = |t|_f$,
- *variable permutative* if it is a permutative axiom such that $\forall m \in \mathcal{D}om(s)\ s(m) \notin \mathcal{X} \Rightarrow s(m) = t(m)$, i.e. only variables are permuted.

Example 2.13 Idempotency ($f(x, x) = x$) is a classical example of collapse and regular axiom. The right inverse axiom ($x * i(x) = e$) is not regular and not collapse.

Now, an equational theory $\mathcal{TH}(E)$ is:

- *finitely presented* (also called *finitely generated*) if it admits a finite axiomatization,
- *finite* if all classes of terms under $\xleftrightarrow{*}_E$ are finite,
- *permutative* if every valid equality is permutative,

- *variable permutative* if all valid equality is variable permutative,
- *regular* if any presentation of $\mathcal{TH}(E)$ contains only regular axioms,
- *collapse-free* if no presentation of $\mathcal{TH}(E)$ contains a collapsing axiom,
- *Noetherian* (relatively to a quasi-ordering $>$) if all strictly decreasing chain of substitution $\sigma_1 > \sigma_2 > \dots$ is finite,
- *almost-free* (P. Szabo [Sza82] call them Ω-*free*) if

$$f(t_1, \dots, t_n) =_E f(t'_1, \dots, t'_n) \Rightarrow \forall i \in [1..n] \;\; t_i =_E t'_i,$$

- *simple* if no valid equality in $\mathcal{TH}(E)$ is subterm collapsing,
- *monadic* if all the symbols involved in the theory are of arity one.

The equational theory of a finite number of associative and commutative symbols is collapse-free, regular, permutative and finite but not variable permutative. Commutativity alone is variable permutative. Of course there exists theories that are finite but not permutative, take for example $E_1 = \{f(a) = g(b)\}$.

Let us give now the main characterizations of these theory classes.

Proposition 2.4 [BHSS90] An equational theory $\mathcal{TH}(E)$ is:

- permutative iff there exists a presentation of $\mathcal{TH}(E)$ consisting only of permutative axioms,
- regular iff there exists a presentation of $\mathcal{TH}(E)$ consisting only of regular axioms,
- collapse-free iff there exists a presentation of $\mathcal{TH}(E)$ without collapse axioms.

The main relations between the above theory classes are the following.

Proposition 2.5 [BHSS90]

1. Every permutative theory is finite.
2. Every finite theory is simple.
3. Every simple theory is regular and collapse-free.
4. Every almost-free theory is regular.
5. Every finite theory is Noetherian.
6. The converse of the above properties are false.

Given a class of equational theories $\mathcal{C}$, the *class problem* is to determine if a theory belongs to this class or not. The next results summarize the status of the class problems for the classes that we just have defined.

Theorem 2.6

1. *The class problem for permutative theories is decidable.*
2. *The class problem for regular theories is decidable.*
3. *The class problem for collapse free theories is decidable.*
4. *The class problem for finite theories is not decidable.*
5. *The class problem for almost-free theories is not decidable.*
6. *The class problem for simple theories is not decidable.*

Proof: The properties 1, 2 and 3 are an immediate consequence of Proposition 2.4.

Point 4 has been proved in [Rao81]. It is also a consequence of the fact, proved in [NOR85], that it is undecidable if a theory $\mathcal{TH}(E)$, such that E is a finite Church-Rosser semi-Thue system, admits any infinite congruence class.

The points 5 and 6 are proved in [BHSS90]. □

Exercice 11 — [BHSS90] Let E be the theory defined by the following monadic term rewriting system:

$$\begin{array}{lllll} g_1(f_1(k(g_3(x)))) & \rightarrow & f_1(h(x)) & g_2(k(x)) & \rightarrow & k(g_3(x)) \\ h(f_2(x)) & \rightarrow & f(f_2(x)) & h(g_3(x)) & \rightarrow & g_2(h(x)) \end{array}$$

Show that this term rewriting system is convergent (see chapter16). Then prove successively that E is simple, almost-free and Noetherian but not finite.
Answer: This is fully described in [citeBurckertHS-LU90] page 27, lemma 3.3.8

2.5 Sorted equational logic

2.5.1 Syntax

In sorted equational logic, equalities are built from many-sorted terms and universally quantified. Sorted presentations specify the sorted signature and the axioms of a given theory.

Definition 2.28 A *presentation*, also called *specification*, and denoted $SP = (\Sigma, E)$, is given by a many-sorted signature Σ, and a set E of universally quantified equalities $(\forall X, t = t')$ where $\mathcal{V}ar(t) \cup \mathcal{V}ar(t') \subseteq X$. (The quantification may be omitted when $X = \mathcal{V}ar(t) \cup \mathcal{V}ar(t')$).

Example 2.14 Consider a presentation of lists with two sorts *List* for lists and *Elt* for elements. Lists are built with two constructors *nil* for the empty list and *push* that concatenate an element to a list. Moreover let us define an operation *alter* on lists that shuffles two lists and produces a third one. The list structure and the *alter* operation are described by the following presentation, where the two-sorted signature Σ gives sorts and ranges of operations:

$$\begin{array}{rcl} sorts: & & Elt, List \\ nil: & \mapsto & List \\ push: Elt\ List & \mapsto & List \\ alter: List\ List & \mapsto & List \end{array}$$

and equalities define the *alter* operation:

$$\begin{array}{rlcl} \forall z : List & alter(nil, z) & = & z \\ \forall x : Elt, y : List, z : List & alter(push(x, y), z) & = & push(x, alter(z, y)) \end{array}$$

Substitutions are defined as mappings σ from sorted variables to sorted terms such that if $x : s$ then $\sigma(x) \in \mathcal{T}(\Sigma, \mathcal{X})_s$.

2.5.2 Deduction system

The deduction rules for equational deduction, given in Figure 2.1 of Section 2.4, generalize to the many-sorted framework provided there is no *empty* sort. A precise analysis of the possible problems that may arise when this hypothesis is not satisfied can be found in [MG85].

2.5.3 Models

Many-sorted algebras have carriers corresponding to each sort and operations with sorted arguments.

Definition 2.29 Given a many-sorted signature Σ, a Σ-algebra $\mathcal{A}$ consists of a family $\{A_s | s \in S\}$ of subsets of $\mathcal{A}$, called the *carriers* of $\mathcal{A}$, and a family of operations $f_{\mathcal{A}} : A_{s_1} \times \ldots \times A_{s_n} \mapsto A_s$, associated to each function $f \in \Sigma$ such that $f : s_1, \ldots, s_n \mapsto s$.

Substitutions, defined as mappings σ from sorted variables to sorted terms such that if $x : s$ then $\sigma(x) \in \mathcal{T}(\Sigma, \mathcal{X})_s$, induce Σ-homomorphisms on the Σ-algebra of many-sorted terms.

In the following, only models with non-empty sorts are considered. This means that for any model $\mathcal{A}$ and any $s \in S$, $\mathcal{A}_s$ is a non-empty set.

A presentation $SP = (\Sigma, E)$ actually describes a class of algebras, namely the class of Σ-algebras satisfying the equalities E, denoted $Mod(E)$ or $ALG(SP)$. $ALG(SP)$ with SP-homomorphisms is a category also denoted by $ALG(SP)$.

Let $\overset{*}{\longleftrightarrow}_E$ denote the replacement of equals by equals on $\mathcal{T}(\Sigma, \mathcal{X})$ which is correct and complete for deduction in $ALG(SP)$:

$$t \overset{*}{\longleftrightarrow}_E t' \text{ iff } \forall \mathcal{A} \in ALG(SP), \mathcal{A} \models (\forall X, t = t').$$

The class $ALG(SP)$ has an initial algebra denoted by $\mathcal{T}(\Sigma)/E$ or by $\mathcal{T}_{SP}$. $\mathcal{T}(\Sigma)/E$ is built as the quotient algebra of the ground term algebra $\mathcal{T}(\Sigma)$ by the congruence $\overset{*}{\longleftrightarrow}_E$ generated by E.

2.6 Conditional logic

2.6.1 Syntax

The formulas being considered are *conditional equalities*, written "$l = r$ if Γ", where Γ is a conjunction of equalities. The meaning of such a formula is that l and r are equal *if* the condition Γ is satisfied. A conditional equality is nothing but an equational Horn clause.

Definition 2.30 An *equational clause* is a clause built with the only equality predicate, and denoted $\Gamma \Rightarrow \Delta$ where Γ is the antecedent and Δ the succedent.

An *equational Horn clause* is an equational clause whose succedent is reduced to one equality. A *conditional equality* is an equational Horn clause $s = t$ if Γ, also denoted

$$l = r \text{ if } (s_1 = t_1 \wedge \cdots \wedge s_n = t_n).$$

when conditions need to be explicite. $\Gamma = (s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ and $l = r$ are respectively called the *condition* and the *conclusion* of the conditional equality.

A *conditional system* is a set of conditional equalities.

In the following, we most often omit the word "equational" but restrict our approach to this kind of clauses. This is actually not a real restriction, since up to an encoding of predicates with boolean functions, any general clause can be translated to an equational clause.

2.6.2 Deduction system

For any set of conditional axioms E, conditional equalities $s = t$ if Γ, can be deduced via inference rules.

A deduction system for conditional equational deduction is given in Figure 2.4. In this system, when $\Gamma = \bigwedge_{i=1,\ldots,n} u_i = v_i$ with $n \geq 0$, $\sigma(\Gamma) = \bigwedge_{i=1,\ldots,n} \sigma(u_i) = \sigma(v_i)$.

Definition 2.31 Given a set of conditional axioms E and a set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$, the *conditional theory of* E, denoted $\mathcal{TH}(E)$, is the set of conditional equalities that can be obtained, starting from E, by applying the inference rules given in Figure 2.4.

2.6.3 Algebraic semantics and models

From the algebraic point of view, conditional systems and equational systems have very similar kinds of results.

Definition 2.32 Let E be a set of conditional axioms. An algebra $\mathcal{A}$ is a *model* of E if for any axiom $l = r$ if $(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ in E, for any assignment ν of variables in $\mathcal{A}$,

$$\text{if } \forall i \in [1, \ldots, n], \nu(s_i) = \nu(t_i) \text{ then } \nu(l) = \nu(r).$$

We also say that the axiom $l = r$ if $(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ is *valid* in $\mathcal{A}$, or that $\mathcal{A}$ *satisfies* $l = r$ if $(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$, and this is denoted by $\mathcal{A} \models s = t$.

Given a set of conditional axioms E, $\mathcal{A}$ is a model of E if $\mathcal{A}$ satisfies all the conditional axioms in E.

Definition 2.33 A set of conditional axioms E is called *consistent* if it has a model, and *inconsistent* or *unsatisfiable* otherwise.

E implies C (written as $E \models C$) if every model of E satisfies C.

1. Reflexivity	
	$\vdash$
	$(t = t)$
2. Symmetry	$(t = t')$
	$\vdash$
	$(t' = t)$
3. Transitivity	$(t = t'), (t' = t'')$
	$\vdash$
	$(t = t'')$
4. Congruence	$\{(t_i = t'_i) \mid i = 1, \ldots, n\}$
	$\vdash$
	$(f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n))$
	if $f \in \mathcal{F}_n$
5. Substitutivity	$t_1 = t_2$ if Γ
	$\vdash$
	$\sigma(t_1) = \sigma(t_2)$ if $\sigma(\Gamma)$
	if $\sigma \in \mathit{Subst}$
6. XXX	$t_1 = t_2$ if $(\Gamma \wedge t'_1 = t'_2),\ t'_1 = t'_2$ if Γ'
	$\vdash$
	$t_1 = t_2$ if $(\Gamma \wedge \Gamma')$

Figure 2.4: The rules of equational conditional deduction

Let $\mathcal{M}od(E)$ denote the set of models of E. It can be proved that there exists a smallest congruence on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ generated by E. This congruence is obtained as the least fixpoint of a continuous function defined on the complete lattice of congruences built on $\mathcal{T}(\mathcal{F}, \mathcal{X})$. For more details, see [GTW78, Kap83, Rém82].

The quotient algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})/E$ of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ by the congruence generated by E is a model of E. Moreover the initial model is the quotient algebra $\mathcal{T}(\mathcal{F})/E$ of ground terms by the congruence generated by E.

Following earlier work, a Birkhoff-like theorem establishes the completeness of conditional replacement of equals by equals [Sel72](see also [BDJ78, BK86]).

The deduction system for conditional equational deduction given in Figure 2.4 is sound and complete with respect to this notion of models.

Theorem 2.7 *[Kap83] For any set of conditional axioms E, for any conditional equality $s = t$ if Γ,*

$$\mathcal{M}od(E) \models s = t \text{ if } \Gamma \text{ iff } E \vdash s = t \text{ if } \Gamma.$$

2.6.4 Herbrand interpretations

Considering a conditional axiom as an equational Horn clause leads to the consideration of another kind of semantics, through Herbrand interpretations.

Definition 2.34 An *equality Herbrand interpretation* $\equiv$ is a congruence on ground terms.

Definition 2.35 An interpretation $\equiv$ is said to *satisfy* a ground equational axiom $C = (l = r \text{ if } \Gamma)$ if either $\Gamma \not\subseteq \equiv$ or $(l = r) \subseteq \equiv$. Then C is said true in $\equiv$, otherwise C is false

An interpretation $\equiv$ is said to *satisfy* a non-ground equational axiom $C = (l = r \text{ if } \Gamma)$ if it satisfies all its ground instances.

Of course, the congruence generated by a set of conditional axioms E on $\mathcal{T}(\mathcal{F})$ is an equality Herbrand interpretation. The relation between the two proposed semantics is summarized in the following proposition.

Proposition 2.6 A ground equational axiom $C = (l = r \text{ if } \Gamma)$ holds in the initial model $\mathcal{T}(\mathcal{F})/E$ iff the equality Herbrand interpretation generated by E satisfies C.

Proof: Let C be $l = r$ if $\bigwedge_{i=1,\ldots,n} s_i = t_i$. The result is due to the equivalence of the two propositions:

- $\forall\sigma$ ground substitution on TF/E, if $\forall i = 1, \ldots, n, \sigma(s_i) =_E \sigma(t_i)$ then $\sigma(l) =_E \sigma(r)$.

- $\forall\sigma$ ground substitution on TF/E, either $\sigma(\Gamma) \not\subseteq =_E$, or $(\sigma(l) = \sigma(r)) \subseteq =_E$.

□

Definition 2.36 A conditional axiom satisfied by any interpretation is called a *tautology.*

A conditional axiom satisfied by no interpretation is said *unsatisfiable* and is called a *contradiction.* A contradiction is denoted by the empty clause $\Rightarrow$.

Example 2.15 A conditional axiom of the form $\Gamma \wedge s = t \Rightarrow s = t$ or $\Gamma \Rightarrow t = t$ are tautologies.

2.6.5 An example

The interest on conditional equalities can by justified by an example due to Bergstra and Meyer [BM84] of a conditional specification whose initial algebra cannot be specified (in the same signature) by means of equations.

Example 2.16 [BM84] The following specification describes finite sets of natural numbers, with a counting function *card* yielding the number of elements in a set. There are two sorts, natural numbers and sets of natural numbers. $\bullet$ denotes the insertion operation of a natural number in a set.

$$\begin{array}{rcl}
sort & & Nat, SetOfNat \\
0 : & \mapsto & Nat \\
succ : Nat & \mapsto & Nat \\
\emptyset : & \mapsto & SetOfNat \\
\bullet : Nat, SetOfNat & \mapsto & SetOfNat \\
card : SetOfNat & \mapsto & Nat
\end{array}$$

$$\begin{array}{rcl}
\forall x : Nat, s : SetOfNat, x \bullet (x \bullet s) & = & x \bullet s \\
\forall x, y : Nat, s : SetOfNat, x \bullet (y \bullet s) & = & y \bullet (x \bullet s) \\
card(\emptyset) & = & 0 \\
\forall x : Nat, card(x \bullet \emptyset) & = & succ(0) \\
\forall x : Nat, card(0 \bullet (succ(x) \bullet \emptyset)) & = & succ(succ(0)) \\
\forall x, y : Nat, card(succ(x) \bullet (succ(y) \bullet \emptyset)) & = & card(x \bullet (y \bullet \emptyset)) \\
\forall x, y : Nat, \forall s : SetOfNat, & & \\
(card(x \bullet (y \bullet \emptyset)) = succ(0) \wedge & & \\
card(x \bullet s) = succ(card(s)) \wedge & & \\
card(y \bullet s) = succ(card(s))) & \text{if} & \\
card(x \bullet (y \bullet s)) & = & succ(succ(card(s)))
\end{array}$$

In [BM84], it is proved that the corresponding initial algebra cannot be specified with finitely many equations in the given signature. Of course a finite equational specification can be obtained if auxiliary functions are allowed.

Chapter 3

Computations in the term algebra

This chapter deals with specific computations in the term algebra: matching, generalisation and unification. It enlights the lattice structure of the free algebra.

3.1 The lattice of terms

The set of terms can be seen as a lattice for the subsumption ordering. We now describe this structure.

3.1.1 Renaming

The relation $\leq$ is a quasi-ordering on terms whose associated equivalence $\equiv$ and strict ordering $<$ are respectively called *subsumption equivalence* and *strict subsumption.*

In case s and t are subsumption equivalent, the following result shows that σ is a one to one mapping from $\mathcal{V}ar(s)$ to $\mathcal{V}ar(t)$, called a *conversion* or more often a *renaming.*

Lemma 3.1 For all terms t and t',

$$t \equiv t' \Leftrightarrow \exists \xi \in Perm,\ t = \xi(t')$$

Proof: It relies on the classical lemma stating that for two mappings $f : E \to F$ and $g : F \to G$, if $g.f$ is injective then f is injective too and, if $g.f$ is surjective then g is surjective too.

If $t = \xi(t')$, since ξ is a permutation, it has an inverse, which prove that $t \equiv t'$.
Conversely, there exist σ and σ' such that

$$\sigma(t) = t' \text{ and } \sigma'(t') = t.$$

and it is not restrictive to assume their domains such that:

$$\mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(t) \text{ and } \mathcal{D}om(\sigma') \subseteq \mathcal{V}ar(t').$$

We then have $\sigma(\sigma'(t')) = t'$ and $\sigma'(\sigma(t)) = t$. By the proposition 2.1 we can write

$$\mathcal{D}om(\sigma.\sigma') \cap \mathcal{V}ar(t') = \mathcal{D}om(\sigma'.\sigma) \cap \mathcal{V}ar(t) = \emptyset$$

This implies that $\forall x \in \mathcal{V}ar(t)$, $\sigma'\sigma(x) = x$ and thus $\sigma'\sigma$ is injective. The same symmetrically hold for $\sigma\sigma'$. Thus σ and σ' are both injective because of the result recalled above.

Now let us prove that $\sigma'\sigma$ is bijective. Assume that there exists x such that $|\sigma'\sigma(x)| \geq 1$. This will contradict the fact that $\sigma'\sigma$ is the identity on t and thus $\sigma'\sigma$ is a finite substitution that maps variables to variables and thus which is bijective by application of lemma 2.3. The same holds symmetrically for $\sigma\sigma'$ and thus both σ and σ' are permutations.

□

It should be emphasized that this result is not true anymore in an arbitrary equational theory:

Example 3.1 If we assume the operator $+$ idempotent and thus satisfying $x+x=x$, there exist substitutions σ and σ' such that:

$$\sigma((x+y)+z) =_E u+v$$
$$\sigma'(u+v) = (x+y)+z$$

(take for example $\sigma = \{(x \mapsto u)(y \mapsto u)(z \mapsto v)\}$ and $\sigma' = \{(u \mapsto x+y), (v \mapsto z)\}$) where σ should identify x and y and thus can not be a permutation.

The previous lemma extends easily to substitutions:

Lemma 3.2 For all substitutions σ and σ',

$$\sigma \equiv \sigma' \Leftrightarrow \exists \xi \in Perm\ \xi.\sigma = \sigma'.$$

This last result is no more true if substitutions with infinite domains are considered as shown by the following example.

Example 3.2 [Hue76] Let $\sigma_1 = \{x_i \mapsto x_{2i}\}_{i\in\mathbf{N}}$ and $\sigma_2 = \{x_{2i} \mapsto x_i\}_{i\in\mathbf{N}}$. Then $\sigma_2.\sigma_1 = Id$ and $\sigma_1.Id = \sigma_1$ thus $Id \equiv \sigma_1$ but obviously there does not exist any permutation ξ such that $\xi.Id = \sigma_1$.

We are denoting $(\widehat{T}, \leq)$ the ordered set obtained as the quotient of the pre-ordered set $\mathcal{T}(\mathcal{F},\mathcal{X})$ by the subsumption equivalence $\equiv$. Since the subsumption equivalence is not a congruence $\widehat{T}$ is not an algebra.

3.1.2 Matching

The matching substitution from t to t', when it exists, is unique and can be computed by a simple recursive algorithm given for example by G. Huet [Hue76] and that we are describing now.

Definition 3.1 A *match-equation* is any formula of the form $t \ll^? t'$, where t and t' are terms. A substitution σ is solution of the match-equation $t \ll^? t'$ if $\sigma t = t'$. A *matching system* is a conjunction of match-equations. A substitution is solution of a matching system P if it is solution of all the match-equations in P. We denote by $\mathbf{F}$ a matching system without solution.

We are now ready to describe the computation of matches by the following set of transformation rules **Match** where the symbol $\wedge$ is assumed to be associative, commutative and idempotent.

Delete		$t \ll^? t \wedge P$	
	$\mapsto\!\!\!\!\rightarrow$	P	
Decomposition		$f(t_1,\ldots,t_n) \ll^? f(t'_1,\ldots,t'_n) \wedge P$	
	$\mapsto\!\!\!\!\rightarrow$	$\bigwedge_{i=1,\ldots,n} t_i \ll^? t'_i \wedge P$	
SymbolClash		$f(t_1,\ldots,t_n) \ll^? g(t'_1,\ldots,t'_m) \wedge P$	
	$\mapsto\!\!\!\!\rightarrow$	$\mathbf{F}$	if $f \neq g$
MergingClash		$x \ll^? t \wedge x \ll^? t' \wedge P$	
	$\mapsto\!\!\!\!\rightarrow$	$\mathbf{F}$	if $t \neq t'$
SymbolVariableClash		$f(t_1,\ldots,t_n) \ll^? x \wedge P$	
	$\mapsto\!\!\!\!\rightarrow$	$\mathbf{F}$	if $x \in \mathcal{X}$

Match: Rules for syntactic matching

Theorem 3.1 *The normal form by the rules in* **Match**, *of any matching problem $t \ll^? t'$ such that $\mathcal{V}ar(t) \cap \mathcal{V}ar(t') = \emptyset$, exists and is unique.*

1. *If it is $\mathbf{F}$, then there is no match from t to t'.*
2. *If it is of the form $\bigwedge_{i\in I} x_i \ll^? t_i$ with $I \neq \emptyset$, the substitution $\sigma = \{x_i \mapsto t_i\}_{i\in I}$ is the unique match from t to t'.*
3. *If it is empty then t and t' are identical: $t = t'$.*

Proof: Termination of the set of rules is clear since each application of a rule strictly decreases the size of a term in the system.
Let us now prove that each of the above rule preserves the set of solutions.
Delete: Since the property that $\mathcal{V}ar(t) \cap \mathcal{V}ar(t') = \emptyset$ is preserved by rule application, $t \ll^? t \Rightarrow \mathcal{V}ar(t) = \emptyset$. It is thus clear that **Delete** preserves the solutions.
Decomposition: If $\sigma(f(t_1, \ldots, t_n)) = f(t'_1, \ldots, t'_n)$, then $f(\sigma(t_1), \ldots, \sigma(t_n)) = f(t'_1, \ldots, t'_n)$ and since f is a free symbol, σ is a solution of all match-equations $t_i \ll^? t'_i$.
For the remaining rules, the proof is clear. □

Exercice 12 — Write a program, in the language of your choice, implementing a matching algorithm derived from the **Match** set of rules.
Answer:
Exercice 13 — Compute the match from the term $f(g(z), f(y, z))$ to the term $f(g(f(a, x)), f(g(c), f(a, x)))$.
Answer: $\{z \mapsto f(a, x), y \mapsto g(c)\}$

It should be noted that the rule **Delete** in the previous set of rule is not correct when the two members of the match equation $t \ll^? t'$ share variables. This can be seen on the following example. The matching problem $f(x, x) \ll^? f(x, a)$ has no solution but we have the following set of transformations: $f(x, x) \ll^? f(x, a) \vdash_{\textbf{Decomposition}} \{x \ll^? x, x \ll^? a\} \vdash_{\textbf{Delete}} \{x \ll^? a\}$ which has an obvious solution.

3.1.3 Lower semi-lattice

Definition 3.2 Let E be a set, $\leq$ be a binary relation on E and $\wedge$ be a binary operation on E. The structure $(E, \leq, \wedge)$ is a *well-founded lower semi-lattice* if:

1. the relation $\leq$ is an ordering
2. all elements (t_1, t_2) of E admit a greatest lower bound denoted $t_1 \wedge t_2$ and defined by:

$$\begin{array}{l} t_1 \wedge t_2 \leq t_i \text{ for } i = 1, 2 \\ \forall t \in E, t \leq t_1 \text{ and } t \leq t_2 \Rightarrow t \leq t_1 \wedge t_2 \end{array}$$

3. the ordering $\leq$ is well-founded.

The next property is a well-known result of lattice theory [Bir67, Hue76]:

Proposition 3.1 If $(E, \leq, \wedge)$ is a well-founded lower semi-lattice then:

1. every non-empty subset F of E admits a greatest lower bound denoted $\wedge F$ or $glb(F)$,
2. every subset F of E, bounded from above, admits a least upper bound denoted $\vee F$ ou $lub(F)$.

3.1.4 Least generalization

The next result, despite of its elementary appearance, requires a technical proof [Hue76] based on the size of the terms and on the number of their variables.

Lemma 3.3 For any finite term t, the set $\{t' \in \widehat{T} | t' \leq t\}$ is finite.

If the ordering considered on the set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is $\leq$, then a set of terms is bounded from above if these terms have a common instance (may be using different substitutions), and bounded from below if there exists a term that can be instanciated in every term of the set.

In order to show that two terms t and t' always have a greatest lower bound for $\leq$, we adopt the presentation of [DJ90a].

We consider the following transformation rules, taking a pair of the form (w, E) where w is a greatest lower bound of the initial set of terms and E is the set of pairs to be generalized.

GeneDecompose	$(w; f(t_1, \ldots, t_n) =_x f(s_1, \ldots, s_n) \wedge E)$
	$\mapsto\!\!\!\twoheadrightarrow$
	$((x \mapsto f(x_1, \ldots, x_n)).w; \{t_1 =_{x_1} s_1 \wedge \ldots \wedge t_n =_{x_n} s_n\} \wedge E)$
	if $x_1, \ldots, x_n$ are new variables
GeneMerging	$(w; s =_x t \wedge s =_y t \wedge E)$
	$\mapsto\!\!\!\twoheadrightarrow$
	$((x \mapsto y).w; s =_y t \wedge E)$

Generalization: Rules computing the least generalization

Lemma 3.4 For all terms t and t' and any variable x not appearing in t and t', the normal form of $(x; \{s =_x t\})$ for the set of rules **Generalization** exists and is called the least generalization of t and t'.

Exercice 14 — Compute the least generalization of $f(g(a), f(a, x))$ and $f(y, f(h(z), v))$. Same question for $f(a, g(a, z))$ and $f(x, g(x, c))$.
Answer: The least generalization are respectively: $f(x_1, f(x_2, x_3))$ and $f(x_1, x_2)$.

Lemma 3.5 The least generalization of any two terms t and t' is the greatest lower bound of these two terms for $\leq$.

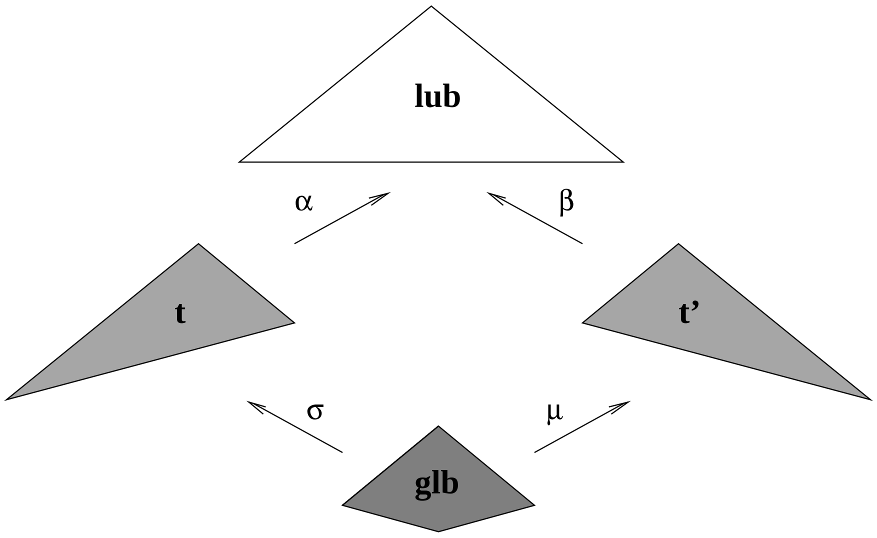
Theorem 3.2 $(\mathcal{T}(\mathcal{F}, \mathcal{X})/\equiv, \leq, \wedge)$ *is a well-founded lower semi-lattice.*

Proof: This is an immediate consequence of the previous lemmas. □

A consequence of the last result and of the Proposition 3.1, is the following:

Corollary 3.1 Any subset of terms bounded from above have a least upper bound.

This last result can be depicted as follow:



Exercice 15 — Let $\mathcal{F}\{f, g, a\}$ where f, g, a are respectively of arity 2, 1, 0. Build the graph of the relation $\leq$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ for the terms of size less than 4.
Answer:
Exercice 16 — Compute the least upper bound of $f(a, b)$ and $f(c, d)$. Same question for $f(g(z), a)$ and $f(z, y)$.
Answer: $f(a, b)$ and $f(c, d)$ are not bounded from above (i.e. there is no term which is a common instance of this two terms) thus they do not have a least upper bound. The least upper bound of $f(g(z), a)$ and $f(z, y)$ is $f(g(x), a)$.

3.1.5 Syntactic unification and least upper bound

Obviously there is a relationship between unification and the notion of least upper bound, but both notions do not coincide in general. In this section, inspired from [Hue76], we study the relationship between these two concepts.

Lemma 3.6 If two term t_1 and t_2 are unifiable then t_1 and t_2 are bounded from above.

The converse is true only for terms that do not share variables. This can be examplified with the two terms $f(a, x)$ and $f(x, b)$, whose least upper bound is $f(a, b)$ but that are clearly not unifiable.

Indeed the computation of the unified term is equivalent to the computation of the least upper bound. This is because every common instance $\sigma_1(t_1) = \sigma_2(t_2)$ determines a unifier $\sigma_{1|Var(t_1)} + (\sigma_2\xi^{-1})_{|Var(t_2)}$ of t_1 and $\xi(t_2)$. One can thus compute the common instances of t_1 and t_2 using unification of t_1 and $\xi(t_2)$.

Conversely, let us show how to compute unifiers from majorant. Let t, t' be two terms to be unified and $V = \mathcal{V}ar(t) \cup \mathcal{V}ar(t')$. Let us call $x_1, \ldots, x_n$ the variables of V and consider the terms

$$t_1 = f(x_1, f(x_2, \ldots, f(x_n, t))) \text{ and } t'_1 = f(x_1, f(x_2, \ldots, f(x_n, t')))$$

If u is a common instance of t_1 and t'_1, then necessarily we have $u = \sigma_1(t_1) = \sigma_2(t_2)$ with $\sigma_{1|V} = \sigma_{2|V} = \{x_i \mapsto u_i | 1 \leq i \leq n\}$. σ_1 and σ_2 coincide on the common variables of t and t', and we can define the substitution $\eta = \sigma_{1|Var(t)} + \sigma_{2|Var(t')}$ for which $\eta(t) = \eta(t')$. Thus t and t' are unifiable. This allows us to state the following result:

Lemma 3.7 When there exists at least one symbol f of arity at least two in $\mathcal{F}$, then two terms t and t' sharing the variables $\{x_1, \ldots, x_n\}$ are bounded from above iff $f(x_1, f(x_2, \ldots, f(x_n, t)))$ and $f(x_1, f(x_2, \ldots, f(x_n, t')))$ are unifiable.

Exercice 17 — Use the previous study to prove that the terms $f(a, x)$ and $f(x, b)$ are not unifiable.
Answer: One has only to build the terms $f(x, f(a, x))$ and $f(x, f(x, b))$ and to check that they have no common instance.

3.2 Syntactic unification

Syntactic unification is the process of solving equations in the free algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})$. In this section, unification problems are assumed to be unquantified conjunctions of equations. This is so because there is no need for new variables to express the (unique) most general unifier.

3.2.1 Definitions

Let us first precise the definition of unification in the term algebra. The following definitions are in fact instances of the general definitions given before.

Definition 3.3 An equation is an unquantified formula of the form $s =^? t$ where s and t are terms. We also consider *unification problems* that are conjunction of equations and that are denoted either $P = (s_1 =^? t_1 \wedge \ldots \wedge s_n =^? t_n)$ or $P = \{s_1 =^? t_1, \ldots, s_n =^? t_n\}$. *Syntactic unification* is the problem of solving unification problems in the free algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A *solution* or *unifier* of an equation $s =^? t$ is a substitution σ such that $\sigma(s) = \sigma(t)$. A substitution is solution of a unification problem P if it is solution of every equation in P. Thus if P is empty, any substitution unifies it. The set of all unifiers of a unification problem P is denoted $\mathcal{U}(P)$. A *most general unifier* of a unification problem P, denoted $mgu(s, t)$, is a minimal element σ of $\mathcal{U}(P)$ with respect to the subsumption strict ordering i.e. for any solution α of the system P, $\sigma \leq^{\mathcal{V}ar(P)} \alpha$. A variable x in $\mathcal{V}ar(P)$ is a *solved variable* for the system P when $P = (Q \wedge x =^? t)$ and $x \notin \mathcal{V}ar(Q)$ and $x \notin Vart$.

Example 3.3 The equation $f(x, a) =^? f(b, y)$ has the unifier $\sigma = \{x \mapsto b, y \mapsto a\}$ in the term algebra $\mathcal{T}(\{f, a, b\}, \{x, y\})$. It is a most general unifier, since it is the unique solution.

Example 3.4 Let us now consider the equation $e = (x + y =^? u + h(x))$. It has an infinite number of unifiers, for example:

$$\begin{array}{rcl}
\mu_1 & = & \{x \mapsto a, u \mapsto a, y \mapsto h(a)\} \\
\mu_2 & = & \{x \mapsto z, u \mapsto z, y \mapsto h(z), z \mapsto u\} \\
\mu_3 & = & \{x \mapsto u, y \mapsto h(u)\} \\
\mu_4 & = & \{x \mapsto z, u \mapsto z, y \mapsto h(z)\}.
\end{array}$$

We can notice that $\mu_1 = \{z \mapsto a\}.\mu_2$, and thus μ_2 is more general than μ_1. But we also have $\mu_2 = \{u \mapsto z, z \mapsto u\}.\mu_3$ and $\mu_3 = \{u \mapsto z, z \mapsto u\}.\mu_2$ and thus $\mu_2 \leq \mu_3$ and $\mu_3 \leq \mu_2$. What looks curious is that $\mu_4 = \{u \mapsto z\}.\mu_3$ but μ_4 is not smaller than μ_3. Similarly μ_2 is smaller than μ_4 but the converse is false! But if one considers the ordering restricted to the variable of interest (in this case $W = V(e) = \{x, y, u\}$) we get $\mu_4 \leq^W \mu_2$ and $\mu_2 \leq^W \mu_4$. Note finally that μ_3 is idempotent.

The previous example illustrates two points:

- First, equivalent unifiers (for example μ_2 and μ_3) can have in general an arbitrary large size, since they differ by a permutation which can be chosen arbitrary large. Thus in most applications, one would like to find "the best most general unifier", for example the idempotent one.
- Second, unifiers can involve variables that are in some sense unnecessary to consider, since they are not in the set of variables of the terms to be unified.

Thus we will see in this section that all most general unifiers are equivalent up to permutation and that the subsumption pre-order $\leq$ can be usefully restricted to consider only the variables of the terms to be unified.

3.2.2 Tree solved forms

For our purpose, we use variations of two different kinds of solved forms:

Definition 3.4 A *tree solved form* for a unification problem P is any conjunction of equations:

$$x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$$

equivalent to P such that $\forall i, x_i \in \mathcal{X}$ and:

$$\begin{array}{ll} (i) & \forall 1 \leq i \leq n, x_i \in \mathcal{V}ar(P), \\ (ii) & \forall 1 \leq i, j \leq n, i \neq j \Rightarrow x_i \neq x_j, \\ (iii) & \forall 1 \leq i, j \leq n, x_i \notin \mathcal{V}ar(t_j). \end{array}$$

Notice that the variables x_i $(i = 1..n)$ are solved. The other variables in P are called *parameters.*

We may speak of a tree solved form without any reference to P, in case P is the tree solved form itself. We may also require that all variables in the solved form are variables of P.

Example 3.5 The unification problem $\{x =^? f(y, a), z =^? g(w)\}$ is in tree solved form but $\{x =^? f(y, z), z =^? a\}$ or $\{x =^? f(y, a), x =^? a\}$ are not in tree solved forms.

Tree solved forms have the following straightforward but useful property:

Lemma 3.8 A unification problem P with tree solved form:

$$P = (x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n)$$

has, up to subsumption equivalence, a unique most general idempotent unifier $\{x_1 \mapsto t_1, \cdots, x_n \mapsto t_n\}$ denoted μ_P.

This is an immediate consequence of the following result:

Lemma 3.9 If x is a variable that does not occur in the term t, then the equation $x =^? t$ has $\sigma = \{x \mapsto t\}$ as most general idempotent unifier.

Proof: σ is obviously a solution.
Conversely, let θ be a solution, then $\theta\sigma x = \theta t = \theta x$. If $y \in \mathcal{V}ar(t)$ then $\theta\sigma y = \theta y$.
Thus $\sigma \leq^{\mathcal{V}ar(t)\cup\{x\}} \theta$. The idempotency of σ is an immediate consequence of the fact that $x \notin \mathcal{V}ar(t)$.
□

Note that these two results remain valid in any equational theory E.
It is clear that in general a unification problem has not a unique tree solved form: for example the problem $P = \{x =^? y\}$ is itself a tree solved form but $\{y =^? x\}$ is also such a solved form for P. This is of course related to the (stricto-sensus non-) unicity of most general unifiers. Let us look at that problem from the point of view of solved forms. As quoted first in [LMM88], for a given solvable unification problem P, the number of equations of tree solved forms of P does not depend on the computing strategy of the unification algorithm but is a natural invariant of the unification problem.

Theorem 3.3 *Let S be a solvable unification problem and P, Q be two of its tree solved forms. Then $|P| = |Q|$ and the number of solved variables and parameters of P and Q are the same.*

By analogy with linear algebra, the number of solved variables of a tree solved form is called in [LMM88] its *rank* and the number of parameters, which is also an invariant, is called the *dimension* of the equational problem.
This result is a consequence of the following lemmas where we first prove that two equivalent tree solved forms have the same set of variables.

Lemma 3.10 In any regular theory, if P and Q are two equivalent tree solved forms then $\mathcal{V}ar(P) = \mathcal{V}ar(Q)$.

Proof: By contradiction. Assume that x is a solved variable in P but is not a variable of Q. If x is solved in P then $\mu_P = \{x \mapsto t\} + \mu$ for some term t and substitution μ. Since x is not a variable of Q and assuming the theory regular, μ is solution of Q but not of P, contradicting the fact that P and Q are equivalent.

If x is a parameter in P but is not a variable of Q. Then $\mu_P = \{y \mapsto t[x]\} + \mu$ for some term t, some variable y and a substitution μ. Since μ_P is an idempotent unifier, we have following Lemma 2.4:

$$\mu_Q\mu_P =^{\mathcal{V}ar(P)\cup\mathcal{V}ar(Q)} \mu_Q.$$

But since $x \notin \mathcal{V}ar(Q)$, we have:

$$\mu_Q\mu_P(y) = \mu_Q(t[x]) = r[x]$$

for some term r. On the other side $\mu_Q(y) = s$ where s does not contain any x because x is assumed not to be a variable of Q. Assuming the theory regular, which is the case of the empty theory, it is not possible to have $r[x] = s$ and thus x should also be a variable of Q. $\Box$

Lemma 3.11 If P and Q are two equivalent tree solved forms then $|P| = |Q|$.

Proof: The principle of the proof is to establish an injection π between the solved variables of P and of Q.
Let x be a solved variable in P, i.e. $P = \{x =^? s\} \cup P'\}$.
If x is a solved variable of Q then we define $\pi(x) = x$.
If x is not a solved variable of Q then there exists a variable y such that $y =^? t[x] \in Q$. And since x is not solved in Q:

$$x = \mu_Q(x) = \mu_Q(s).$$

But this is possible only if s is a variable z and $\mu_Q(s)$ is also a variable. And because of the last equality, this variable should be x. In this situation we necessary have:

$$P = \{x =^? z\} \cup P'\} \text{ and } Q = \{z =^? x\} \cup Q'\}$$

In this case, let us define π by $\pi(x) = z$.

We have now to prove that π is an injection.
Let us assume that this is not true. In which case:

$$P = \{x =^? s, z =^? t\} \cup P'\} \text{ and } Q = \{y =^? u\} \cup Q'\}$$

such that (1) : $\pi(x) = y$ and (2) : $\pi(z) = y$. By definition of π, there are two possible cases for each of the equalities (1) and (2). We summarize all the possible cases in the following table:

	$x = y$	$x = u, y = s$
$z = y$	$x = y = z$	$z = y = s$
$z = u, y = t$	$x = y = t$	$x = z = u$

But all these cases are clearly impossible since P and Q are tree solved forms (and thus x and z, x and t, z and s must be different variables).
This prove that π is injective and thus that $|P| \leq |Q|$. But we can in the same way define an injection from Q to P and thus prove that $|Q| \leq |P|$ which terminates the proof. $\Box$

The reader may had noticed that the previous proof can be extended to the case of regular collapse free theories.

3.2.3 Dag solved form

We now define another kind of solved form, which is important for complexity (and efficiency) reasons, by relaxing the third condition defining tree solved forms:

Definition 3.5 *A* dag solved form *for a unification problem P is any system of equations:*

$$x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$$

equivalent to P such that $\forall i, x_i \in \mathcal{X}$ and:

$$\begin{array}{ll} (i) & \forall 1 \leq i \leq n, x_i \in \mathcal{V}ar(P), \\ (ii) & \forall 1 \leq i, j \leq n, i \neq j \Rightarrow x_i \neq x_j, \\ (iii) & \forall 1 \leq i \leq j \leq n, x_i \notin \mathcal{V}ar(t_j). \end{array}$$

Notice that a tree solved form is a dag solved form and than the two definitions differ only in item (*iii*). Of course, dag solved forms save space, since the value of the variable x_j may not be duplicated in the $t_i's$ for $j \geq i$ (see section 3.2.5).

Example 3.6 The equational problem $\{x =^? f(y,z) \wedge y =^? a\}$ is a dag solved form but $\{y =^? a \wedge x =^? f(y,z)\}$ is not and $\{x =^? f(x,y)\}$ is of course not, because of the cycle on the variable x.

Lemma 3.12 A unification problem:

$$P = (x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n)$$

in dag solved form has, up to subsumption equivalence, a unique most general idempotent unifier $\sigma = \sigma_n \cdots \sigma_2\sigma_1$, where $\sigma_i = \{x_i \mapsto t_i\}$.

Proof: First, σ is a solution of P, since

$$\forall j \quad \sigma x_j = \sigma_n \ldots \sigma_2\sigma_1 x_j = \sigma_n \ldots \sigma_{j+1} t_j = \sigma t_j$$

by the above conditions on the variables.
Second, σ is idempotent by definition of a dag solved form.
Last, in order to prove that σ is most general it is enough to prove $\theta\sigma =^{\mathcal{V}ar(P)} \theta$ (see Lemma 2.4). Let us prove this by induction on n. The base case ($j = 0$): the system of equation is empty and the substitution is the identity which is most general.
Let n be at least 1, and θ be an arbitrary solution.

- for y different from all the x_i:
$$\theta\sigma_n \cdots \sigma_2\sigma_1(y) = \theta(y)$$
- for x_n:
$$\theta\sigma_n \cdots \sigma_2\sigma_1(x_n) = \theta(t_n) = \theta(x_n)$$
- for x_i *s.t.* $i < n$:

$\theta\sigma_n \cdots \sigma_2\sigma_1(x_i)$	by definition of the σ_j we get:
$= \theta\sigma_n \cdots \sigma_{i-1}\sigma_i(x_i)$	by definition of the σ_i we get:
$= \theta\sigma_n \cdots \sigma_{i-1}(t_i)$	by application of the induction hypothesis on the system P without the first i equations we get:
$= \theta(t_i) = \theta(x_i)$	

which concludes the proof. □

Dag solved forms relate to the so-called occur-check ordering on $\mathcal{X}$:

Definition 3.6 Given a unification problem P, let $\sim_P$ be the equivalence on $\mathcal{X}$ generated by the pairs (x, y) such that $x =^? y \in P$. The *occur-check* relation $\prec^{oc}$ on $\mathcal{X}$ defined by P is the quasi-ordering generated by the pairs (x', y') such that $x' \sim_P x, x =^? f(s_1, ..., s_n) \in P, y \in \mathcal{V}ar(f(s_1, ..., s_n)), y \sim_P y'$.

Example 3.7 For the following system $P = x =^? f(u,a) \wedge u =^? g(f(a,x)) \wedge x =^? y \wedge x =^? z$ we have $x \sim_P y \sim_P z$ and $y \prec^{oc} u \prec^{oc} x$.

In a dag solved form, any two variables are not in the equivalence of the occur-check ordering. Conversely, a system of equations of the form $x =^? t$ with $x \in \mathcal{X}$ and such that $\prec^{oc}$ is acyclic, can be ordered (using topological sort) so as to meet the above condition. Accordingly, such a set of equations will be considered in dag solved form.

3.2.4 Complete sets of rules for syntactic unification

We are now giving transformation rules for computing solved forms of unification problems. They use a constant **F** which denote a unification problem without solution.
Notation: Given a set of equations P, $\{x \mapsto s\}P$ denotes the conjunction of equations obtained from P by replacing all the occurrences of the variable x by the term s.

Let **SyntacticUnification** be the following set of transformation rules:

Delete	$P \wedge s =^? s$	
$\mapsto\!\!\!\!\rightarrow$	P	
Decompose	$P \wedge f(s_1, \dots, s_n) =^? f(t_1, \dots, t_n)$	
$\mapsto\!\!\!\!\rightarrow$	$P \wedge s_1 =^? t_1 \wedge \dots \wedge s_n =^? t_n$	
Conflict	$P \wedge f(s_1, \dots, s_n) =^? g(t_1, \dots, t_p)$	
$\mapsto\!\!\!\!\rightarrow$	$\mathbf{F}$	if $f \neq g$
Coalesce	$P \wedge x =^? y$	
$\mapsto\!\!\!\!\rightarrow$	$\{x \mapsto y\}P \wedge x =^? y$	if $x, y \in \mathcal{V}ar(P)$ *and* $x \neq y$
Check*	$P \wedge \ x_1 =^? s_1[x_2] \wedge \dots$ $\dots \wedge x_n =^? s_n[x_1]$	
$\mapsto\!\!\!\!\rightarrow$	$\mathbf{F}$	if $s_i \notin \mathcal{X}$ *for some* $i \in [1..n]$
Merge	$P \wedge x =^? s \wedge x =^? t$	
$\mapsto\!\!\!\!\rightarrow$	$P \wedge x =^? s \wedge s =^? t$	if $0 < \lvert s \rvert \leq \lvert t \rvert$
Check	$P \wedge x =^? s$	
$\mapsto\!\!\!\!\rightarrow$	$\mathbf{F}$	if $x \in \mathcal{V}ar(s)$ *and* $s \notin \mathcal{X}$
Eliminate	$P \wedge x =^? s$	
$\mapsto\!\!\!\!\rightarrow$	$\{x \mapsto s\}P \wedge x =^? s$	if $x \notin \mathcal{V}ar(s), s \notin \mathcal{X}, x \in \mathcal{V}ar(P)$

SyntacticUnification: Rules for syntactic unification

The transformation rules **Conflict** and **Decompose** must be understood as schemas, f and g being quantified over the signature. We avoid merging **Coalesce** and **Eliminate** into a single rule on purpose, because they do not play the same role. **Coalesce** takes care of variable renaming: this is the price to pay for alpha-conversion. **Eliminate** is quite different from **Coalesce** because it makes terms growing, thus we will see how to avoid applying it.

First of all let us prove that all these rules are sound i.e. preserve the set of unifiers.

Lemma 3.13 All the rules in **SyntacticUnification** are sound.

Proof: Let us prove that **Delete** is sound: If σ is a solution of P then it is also solution of $P \wedge s =^? s$ and the converse is also obvious. Notice that this will also hold in any equational theory.

A more interesting case is **Decompose**: If σ is solution of $P \wedge s_1 =^? t_1 \wedge \dots \wedge s_n =^? t_n$ then clearly by congruence σ is solution of $P \wedge f(s_1, \dots, s_n) =^? f(t_1, \dots, t_n)$. Conversely, when σ is solution of $P \wedge f(s_1, \dots, s_n) =^? f(t_1, \dots, t_n)$ then it should be solution of P and $f(\sigma(s_1), \dots, \sigma(s_n)) = f(\sigma(_1), \dots, \sigma(t_n))$. Since this equality is syntactic, this implies that σ is solution of $P \wedge s_1 =^? t_1 \wedge \dots \wedge s_n =^? t_n$. Notice that this cannot be extented as such to an equational theory (imagine for example f commutative). □

Definition 3.7 A *syntactic unification procedure* is any sequence of application of the transformation rules in **SyntacticUnification** on a finite set of equations P.

In fact a strategy of application of the rules in **SyntacticUnification** determines a unification procedure. Some are complete, some are not. Let us first show that a brute force fair strategy is complete.

Theorem 3.4 *Starting with a unification problem P and using the above rules repeatedly until none is applicable results in* **F** *iff P has no solution, or else it results in a tree solved form of P:*

$$x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n.$$

Moreover

$$\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$$

is a most general unifier of P.

Proof: By Lemma 3.13, all the considered rules preserve the set of solutions. We shall now prove that the process stops and that the normal forms are indeed tree solved forms.

This last point follows immediately from a step by step inspection of the different cases, left as exercise to the reader.

The most difficult point is to prove that the process terminates. This is not completely obvious, since the rule **Eliminate** makes terms bigger but **Decompose** decrease their size. The type of an equation is a positive integer defined by:

$$\begin{array}{l} type(x,y)\!=\!2 \text{ if } x,y,\in \mathcal{X} \\ type(x,t)\,=\!1 \text{ if } x\in\mathcal{X}, t\notin\mathcal{X} \\ type(s,t)\,=\!0 \text{ if } s,t\notin\mathcal{X} \end{array}$$

the complexity of one equation is:

$$I(s =^? t) = (\max(|s|,|t|), type(s,t))$$

and the complexity of a system is defined as:

$$\begin{array}{rcl} I(\mathbf{F}) & = & (0,\emptyset) \\ I(s_1 =^? t_1 \ \wedge\ \ldots\ \wedge\ s_n =^? t_n) & = & (N, \{I(s_1 =^? t_1),\ldots,I(s_n =^? t_n)\}) \end{array}$$

where N is the number of non-solved variables. We compare the complexities lexicographically, using the standard ordering on naturals for the first component and the multiset ordering for the second component (for a definition of the multiset ordering see section 4.2.2).

We will now check that each application of the rule in **SyntacticUnification** decreases the complexity of the system on which it is applied.

Delete decreases the second component.

Decompose may increase the number of solved variables but in any case it decreases the second component since it replaces an equation by stricly smaller ones.

Conflict obviously decreases I.

Coalesce stricly increases the number of solved variables since $y \in \mathcal{V}ar(P)$.

Check obviously decreases I.

Eliminate increases by one the number of solved variable since $x \in \mathcal{V}ar(P)$ and $x \notin s$ and $s \notin \mathcal{X}$.

Check* obviously decreases I.

Merge may increase the number of solved variables, but in any case it decreases the type of the equation considered. Let us detail this in the last case. Let $S = (P \ \wedge\ x =^? s \ \wedge\ x =^? t)$. Then

$$I(S) = (\xi, \{I(P), I(x =^? s), I(x =^? t)\})$$

and

$$I(P \ \wedge\ x =^? s \ \wedge\ s =^? t) = (\xi, \{I(P), I(x =^? s), I(s =^? t)\}).$$

So because we impose $0 < |s| \leq |t|$, $\max(|s|,|t|) = \max(|x|,|t|)$ and $type(x =^? t) > type(s =^? t)$, thus I as decreased.

Finally, σ is a mgu of P since it is a mgu for a tree solved form of P by Lemma3.8. □

As shown by the previous proof, the condition $0 < |s| \leq |t|$ is fundamental in the **Merge** rule.

Exercice 18 — Give a unification problem P such that without that condition **Merge** does not terminate

Answer: Take $P = \{x =^? f(x) \ \wedge\ x =^? f(f(x))\}$

Now that we have proved that the whole set of rule terminates, we can envisage complete restrictions of it. Let us first define useful subsets of the rules in **SyntacticUnification**. We introduce the set of rules:

TreeUnify=\{**Delete, Decompose, Conflict, Coalesce, Check, Eliminate**\}
and =
DagUnify =\{**Delete, Decompose, Conflict, Coalesce, Check*, Merge**\}.

Corollary 3.2 Starting with a unification problem P and using the rules **TreeUnify** repeatedly until none is applicable, results in **F** iff P has no solution, or else in a tree solved form:

$$\{x_1 =^? t_1 \ \wedge\ \ldots\ \wedge\ x_n =^? t_n\}$$

such that $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is a most general unifier of P.

Proof: This is a clear consequence of Theorem 3.4. In fact **Merge** and **Check*** are useless for getting the tree solved forms. Termination is of course not affected when the set of rules is restricted. □

Exercice 19 — Apply the set of rules **TreeUnify** to the following unification problems:

$$\begin{array}{rcl} f(g(h(x),a),z,g(h(a),y)) & =^? & f(g(h(g(y,y)),a),x,z) \\ f(g(h(x),a),z,g(h(a),y)) & =^? & f(a,x,z) \end{array}$$

Answer:

We can also forbid the application of the **Eliminate** rule, in which case we get dag solved forms:

Corollary 3.3 Starting with a unification problem P and using the rules **DagUnify** repeatedly until none is applicable, results in **F** iff P has no solution, or else in a dag solved form

$$x_1 =^? t_1 \wedge \ldots \wedge x_n =^? t_n$$

such that $\sigma = \{x_n \mapsto t_n\} \ldots \{x_1 \mapsto t_1\}$ is a most general unifier of P.

Proof: This is also a clear consequence of Theorem 3.4. One can check easily that the normal forms are indeed dag solved forms and termination is not affected. □

Exercice 20 — Apply the set of rules **DagUnify** to the following unification problem:

$$f(g(h(x),a),z,g(h(a),y)) =^? f(g(h(g(y,y)),a),x,z)$$

Compare with what you get using the set of rules **TreeUnify**.
Answer:

3.2.5 Complexity of Syntactic Unification

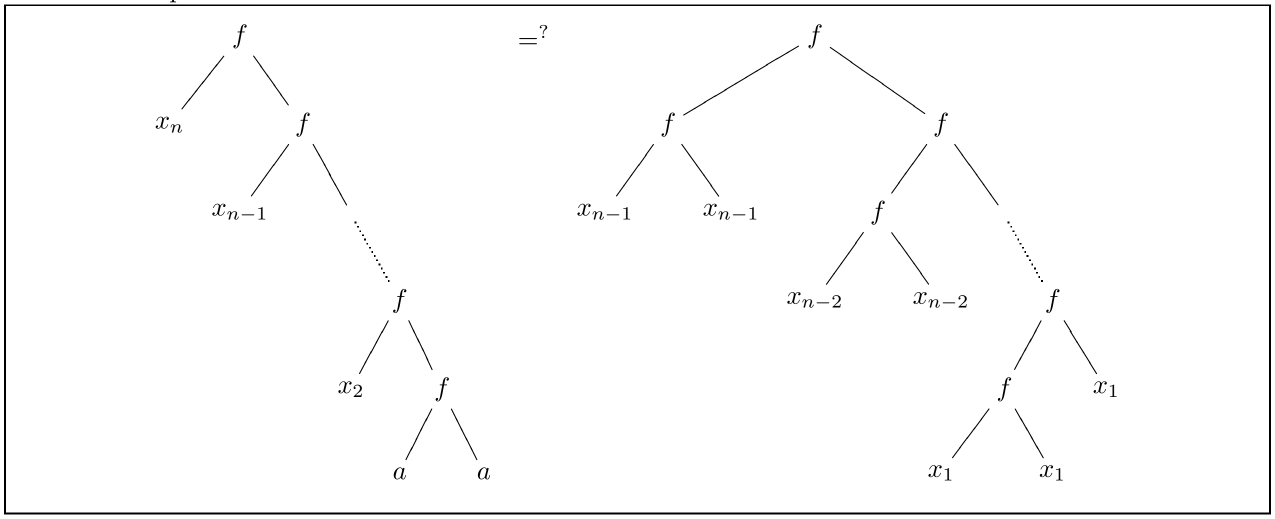
Designing an algorithm for syntactic unification involves implementing the above rules, with appropriate data structures for representing terms and storing equations, as well as a specific search plan. Terms can be represented as labeled trees without sharing, or as dags. Equations can be stored in a queue or a stack, or in a more elaborated data structure in order to speed up **Merge**, **Eliminate** or **Check***. The search can be driven by the data structure storing the equations, or it can again contribute to the speed-up by performing **Eliminate** and **Check** efficiently. We review some of these issues in turn.

Syntactic unification can be polynomial or exponential in space and time according to the data structure of terms representation. If terms are represented by labeled trees, then unification is exponential in the worst case. If terms are represented by labeled dags, then unification can be linear. One reason is the following: given two unifiable terms s and t with most general unifier σ, then the size of $\sigma(s)$ depends on the data structure of terms.

Example 3.8 Consider for instance the terms:

$$\begin{array}{rcl} s & = & f(x_n, f(x_{n-1}, \cdots, f(x_2, f(a,a)) \cdots)) \\ t & = & f(f(x_{n-1}, x_{n-1}), \cdots, f(f(x_2,x_2), f(f(x_1,x_1),x_1)) \cdots). \end{array}$$

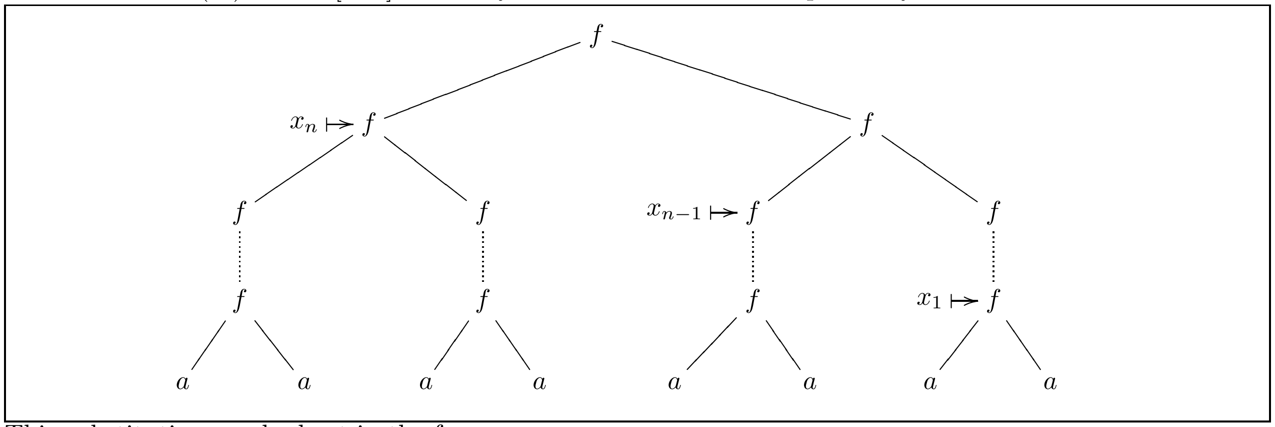
that can be represented as follows:



then the most general unifier is given by the substitution

$$\sigma = \{x_1 \mapsto f(a,a), x_2 \mapsto f(f(a,a), f(a,a)), \cdots, x_n \mapsto f(f(\cdots,\cdots), f(\cdots,\cdots))\}$$

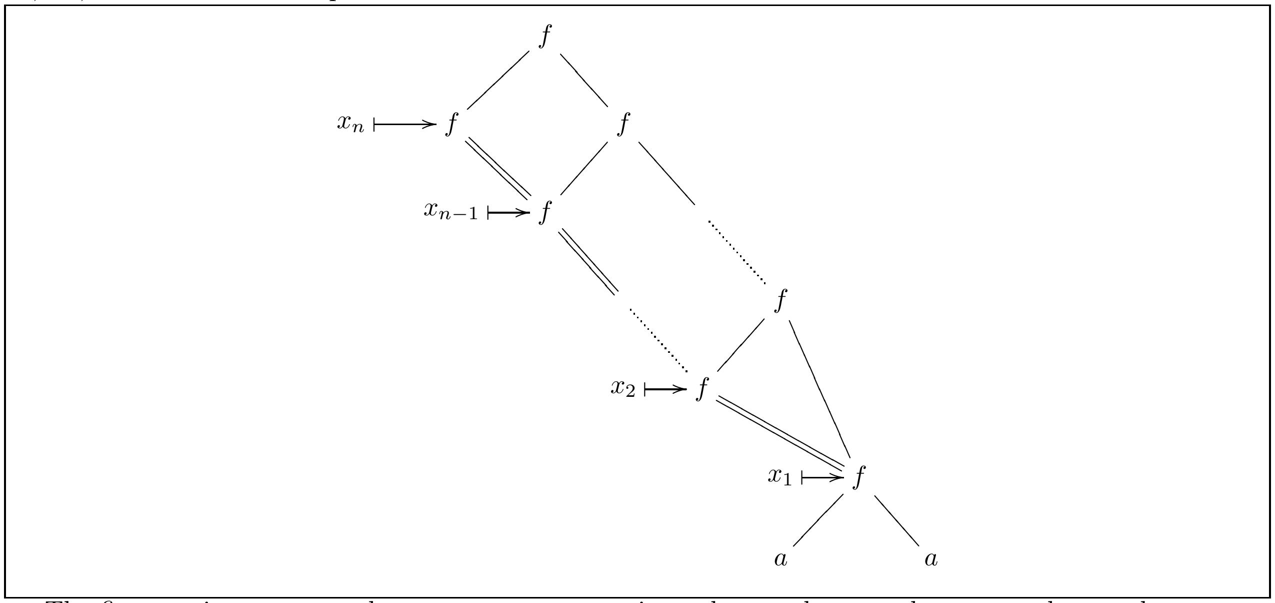
and the size of $\sigma(x_i)$ for $i \in [1..n]$ is exactly $2^{i+1} - 1$. This can be depicted by:



This substitution can be kept in the form:

$$\sigma = \{x_1 \mapsto f(a,a), x_2 \mapsto f(x_1,x_1), \cdots, x_n \mapsto f(x_{n-1}, x_{n-1})\}.$$

In which case the size needed for the storage of x_i is now equal to $3 \times i$, since it implies the storage of $x_1, \cdots, x_{i-1}$. This can be represented as follows:



The first version corresponds to a tree representation, whereas the second corresponds to a dag representation. This shows that a tree representation may require exponential storage, while the corresponding dag representation requires only linear space. Note that a dag representation is actually described by a dag solved form.

Proposition 3.2 The dag solved form of a unification problem P contains distinct subterms of P only and hence has a size of at most $|P|^2$.

A quadratic size is obtained for example when solving the equation

$$x_1 + (x_2 + \cdots (x_n + a)\ldots) =^? ((\ldots(x + x_n) + \cdots) + x_2) + x_1.$$

It is actually possible to obtain a linear size dag solved form at the price of a slight complication in the expression of **Merge**. This can be done by taking the *common part* of s and t as the value of x, and adding the *frontier* to the transformed problem, more sharing is allowed. What is important, however, is that all terms in the dag solved form are subterms of the terms in the starting unification problem. Hence, the dag solved form can be represented in linear space (actually within the space of the starting problem) by sharing common subterms. It does not follow, however, that the dag solved form can be obtained in linear time.

This is indeed not true in general, since an arbitrary use of the above rules may do useless work, resulting in a non-linear complexity. Linear complexity algorithms are due to Paterson and Wegman [PW78] (see also [dC84]) and H. Zhang [Zha92] using the following control principles:

1. Apply **Decompose** whenever possible, hereby decreasing the size of the dag representation of the terms in the problem. Each application of **Decompose** can be done in constant time.

2. Choose a maximal variable x with respect to the occur-check ordering. This can be simply built in the data structure of equations, by using variable counters as in [MM82], so as to use constant time. Another technique is described in [PW78]. Note that **Check*** can be applied if no such variable exists.

3. Apply **Coalesce** to all variables y equivalent to x. Note that each application of **Coalesce** eliminates a variable, hereby decreasing the size of the problem. **Coalesce** can be efficiently implemented by using pointers from each variable to the father nodes of its occurrences.

4. Apply **Merge** to all equations $x =^? s$. This can be simply done by replacing all pointers to x by pointers to s and generating the appropriate equations between the different values of x. Note that this actually corresponds to a specific instance of **Eliminate**.

This principles ensures that the resulting set of equations is a solved form without ever explicitly applying **Check***. Moreover, it yields a linear algorithm, since each step either decreases the size of the problem or moves (once and for all) an arc in the dag data structure of terms.

The crux of this linear complexity algorithm is that all **Merge**s on the variables x belonging to the same equivalence class are performed "at the same time": pointers to all occurrences of all these variables are moved once and for all to their common value. Repeated updatings, as in [Hue76], in an equation of the form

$$f(x_1, x_3, x_5, x_7, x_1, x_5, x_1) =^? f(x_2, x_4, x_6, x_8, x_3, x_7, x_5)$$

results in a complexity at least of the order of $O(n * G(n))$, where $G(n)$ is an extremely slowly growing function related to the inverse of the Ackerman function. So are the algorithms by [Hue76, MM82]. The linear time algorithm was discovered independently by [PW78, MM76]. The almost linear algorithm was discovered independently by several people, including A. Robinson and [RP89]. The first written version appeared in [Hue76].

One may ask whether syntactic unification can be speeded up by using massive parallelism. It is somewhat surprising that this is not the case, due to the non-linearity of terms. Dwork et al. [DKM84] show that unification of terms is logspace complete for P: unless $P \subset NC$, no parallel algorithm for unifying s and t will run in a time bounded by a polynomial in the logarithm of $|s| + |t|$ with a number of processors bounded by a polynomial in P.

3.3 Unification in infinite rational terms

Eliminating **Check*** from **DagUnify** yields a unification algorithm for infinite rational terms. Infinite rational terms are infinite terms with finitely many different subterms only, i.e., they are the unifiers of equations of the form $\{x_1 =^? t_1, \cdots, x_n =^? t_n\}$ such that $x_i \neq x_j$ when $i \neq j$ and $\forall\ j \neq i,\ x_i \notin \mathcal{V}ar(t_j)$ if $t_i \in \mathcal{X}$. Of course, finite terms are particular infinite rational terms. Infinite rational terms are represented by finite directed graphs, with possibly cycles. Unification for infinite rational terms was first solved in [Hue76] (see also [Col84]). The above results and proofs extend to infinite rational terms, except the complexity analysis, which relies on the acyclicity of the occur-check ordering. Actually no linear time unification algorithm is known for infinite rational terms, the problem is open.

3.4 Further Readings

Most unification algorithms described in the literature are reviewed in [Kni89] which also includes interesting historical remarks. The notion of dimension is explored in [LMM88]. A thorough study of unification as a Gentzen's style deduction system is done in [LC89]. Unification in rational trees is studied in [Hue76, Fag83, Col84, Jaf84]. The relationship between congruence closure and unification is studied in [KR89a] and is applied to give some parallelizable case of unification. A general survey on unification, including equational and higher-order unification is [JK91].

Part II

Rewriting

Chapter 4

Abstract reduction systems

4.1 Introduction

This chapter is mainly devoted to the study of the reflexive and transitive closure of a binary relation defined on a set that becomes then a quasi-ordered set. Quasi orderings of special interest are well-founded orderings and well-quasi orderings whose properties are studied in the first part of this chapter. Such relations are then used to define abstract reduction systems and to prove their termination.

Several examples of abstract reduction systems are presented in the following chapters: term rewriting systems are one of the most prominent examples, but also class rewriting on a quotient term algebra, ordered rewriting on the set of ground terms, different notions of conditional rewriting and rewriting with constraints, and polynomial reduction illustrate this concept. Abstract reduction systems usefully abstract the usual "concrete" notion of term rewriting systems and provide a better understanding of the common properties of such relations.

4.2 Quasi orderings

Let us first set the formal background concerning orderings that are needed in this work.

4.2.1 Basic definitions

Definition 4.1 A binary relation $\mathcal{R}$ on a set $\mathcal{T}$ is:

- *reflexive* if $\forall x \in \mathcal{T},\ x \mathrel{\mathcal{R}} x$,
- *antisymmetric* if $\forall x, y \in \mathcal{T},\ x \mathrel{\mathcal{R}} y$ and $y \mathrel{\mathcal{R}} x \Rightarrow x = y$,
- *transitive* if $\forall x, y, z \in \mathcal{T},\ x \mathrel{\mathcal{R}} y$ and $y \mathrel{\mathcal{R}} z \Rightarrow x \mathrel{\mathcal{R}} z$,
- a *quasi ordering* if it is reflexive and transitive and in this case, $(\mathcal{T},\ \mathcal{R}\ )$ is called a quasi-ordered set,
- an *ordering* if it is reflexive, antisymmetric and transitive, such an ordering is also called a *partial ordering* and $(\mathcal{T},\ \mathcal{R}\ )$ is called a *poset*,
- a *total (quasi) ordering* if it is a (quasi) ordering and $\forall x, y \in \mathcal{T}\ x \mathrel{\mathcal{R}} y$ or $y \mathrel{\mathcal{R}} x$.

A quasi ordering is sometimes called a *pre ordering.* An ordering on a set $\mathcal{T}$ is often denoted $\geq$, its symmetric relation is denoted by $\leq$. When $\geq$ is a quasi ordering on $\mathcal{T}$, we say that $(\mathcal{T}, \geq)$ is a *quasi-ordered set.*

The equivalence relation associated to a quasi ordering $\geq$ on a set $\mathcal{T}$ is denoted $\asymp_{\geq}$ and defined as usual as the intersection of $\geq$ and its symmetric relation $\leq$, i.e.

$$\forall x, y \in \mathcal{T}, x \asymp_{\geq} y \Leftrightarrow x \geq y \text{ and } y \geq x.$$

The associated *strict ordering* $>$ is defined by:

$$t > t' \text{ if } t \geq t' \text{ and } t \not\asymp_{\geq} t'.$$

Definition 4.2 Given a poset $(\mathcal{T}, \mathcal{R})$, and a subset T of $\mathcal{T}$,

- an element $lb \in \mathcal{T}$ is a *lower bound* of T if $\forall y \in T, lb \leq y$.
- an element $ub \in \mathcal{T}$ is a *upper bound* of T if $\forall y \in T, y \leq ub$.
- an element $le \in T$ is the *least element* of T if $\forall y \in T, le \leq y$.
- an element $ge \in T$ is the *greatest element* of T if $\forall y \in T, y \leq ge$.
- an element $lub \in \mathcal{T}$ is the *least upper bound* of T if the set of upper bounds of T is nonempty and lub is the least element of this set.
- an element $glb \in \mathcal{T}$ is the *greatest lower bound* of T if the set of lower bounds of T is nonempty and glb is the greatest element of this set.
- an element $x \in T$ is *minimal* in T if $\forall y \in T, y \leq x \Rightarrow x = y$.
- an element $x \in T$ is *maximal* in T if $\forall y \in T, x \leq y \Rightarrow x = y$.
- T is a *Chain* when all the elements in T are comparable, i.e., $\forall x, y \in T, x \leq y$ *or* $y \leq x$.

It can be easily shown that least and greatest elements are unique when they exist and thus so are greatest lower bound and least upper bound. On the contrary, lower and upper bounds as well as minimal and maximal elements are not necessary unique.

Exercice 21 — For a given poset, give examples of all elements presented in the last definition.
Answer: Just do it!

We will need the following well known result:

Theorem 4.1 *(Zorn's lemma) Given a poset $(\mathcal{T}, \mathcal{R})$, if every nonempty chain in $\mathcal{T}$ has an upper bound, then $\mathcal{T}$ has a maximal element.*

4.2.2 Well-founded orderings

Definition 4.3 A quasi-ordered set $(\mathcal{T}, \geq)$ is *well-founded* or *Noetherian* if there exists no infinite decreasing sequence $t_1 > t_2 > \dots$ of elements of $\mathcal{T}$. In this case $\geq$ is called a *well-founded ordering*.

This can be equivalently stated by: a quasi ordering $\geq$ is well-founded if every infinite sequence $t_1 \geq t_2 \geq \dots$ is stationary, i.e. there exists n such that $\forall i \geq n, t_i \asymp_{\geq} t_n$.

Well-founded sets are especially interesting because they enjoy a very powerful induction principle, called *Noetherian induction principle*.

Given a quasi-ordered set $(\mathcal{T}, \geq)$, a predicate $Pred$ is said *complete* on $(\mathcal{T}, \geq)$ if:

$$\forall t \in \mathcal{T},\ [\forall t' \in \{t' | t > t'\}, Pred(t')] \Rightarrow Pred(t).$$

Noetherian induction principle: Let $(\mathcal{T}, \geq)$ be a well-founded set. If $Pred$ is a predicate complete on $(\mathcal{T}, \geq)$ such that $Pred(t)$ is true for every minimal element t, then $\forall t \in \mathcal{T}, Pred(t)$.

Lexicographic extension

A new ordering can be built from elementary ones by combining them lexicographically. Tuples of same length can be ordered by a so-called lexicographic extension as follows.

Definition 4.4 Consider n quasi-ordered sets $(\mathcal{T}_i, >_i)_{i=1,\dots,n}$. The *lexicographical extension* of $(>_i)_{i=1,\dots,n}$, denoted $(>^{lex})$, is defined on the product $\prod_{i=1,..,n} \mathcal{T}_i$ by:

$$(s_1, \dots, s_n) >^{lex} (t_1, \dots, t_n)$$

if there exists i, $1 \leq i \leq n$ such that:

- $s_i >_i t_i$,
- and $\forall j, 1 \leq j < i,\ s_j = t_j$.

Proposition 4.1 Consider n quasi-ordered sets $(\mathcal{T}_i, >_i)_{i=1,\dots,n}$. If for every $i = 1, \dots, n$, $>_i$ is well-founded on $\mathcal{T}_i$, then $>^{lex}$ is well-founded on $\prod_{i=1,..,n} \mathcal{T}_i$.

Note that this result would not extend to an infinite product of ordered sets: Take for example $\mathcal{T} = \{a, b\}$ with $a < b$, then we have:

$$b >^{lex} ab >^{lex} aab >^{lex} aaab >^{lex} \ldots$$

An important particular case is when all the ordered sets are the same set $\mathcal{T}$. The previous definition then allows defining the *lexicographic extension* of any well-founded ordering $>$ on $\mathcal{T}$, which is well-founded on $\mathcal{T}^n$. Note also that, given n totally ordered sets $(\mathcal{T}_i, >_i)_{i=1,\ldots,n}$, $>^{lex}$ is then total on $\prod_{i=1,..,n} \mathcal{T}_i$.

Tuples of possibly different but bounded length can also be ordered by a lexicographic extension extending the previous one.

Definition 4.5 Consider n quasi-ordered sets $(\mathcal{T}_i, >_i)_{i=1,\ldots,n}$. The *lexicographical extension* of $(>_i)_{i=1,\ldots,n}$, denoted $(>^{lexe})$, is defined on the sum of products $\bigoplus_{j=1,\ldots,n} \prod_{i=1,..,j} \mathcal{T}_i$ by:

$$(s_1, \ldots, s_m) >^{lexe} (t_1, \ldots, t_p) \text{ with } m, p \leq n$$

if $(s_1, \ldots, s_k) >^{lex} (t_1, \ldots, t_k)$ where k is the minimum of m and p.

Proposition 4.2 Consider n quasi-ordered sets $(\mathcal{T}_i, >_i)_{i=1,\ldots,n}$. If for every $i = 1, \ldots, n$, $>_i$ is well-founded on $\mathcal{T}_i$, then $>^{lexe}$ is well-founded on $\bigoplus_{j=1,\ldots,n} \prod_{i=1,\ldots,j} \mathcal{T}_i$.

Proof: Assume that there exists an infinite decreasing sequence for $>^{lexe}$,

$$(s_1, \ldots, s_m) >^{lexe} \ldots >^{lexe} (t_1, \ldots, t_p) >^{lexe} \ldots$$

Since the length of tuples is bounded by n, there exists some element, say $(s'_1, \ldots, s'_k)$ of maximal length in the sequence. Complete other tuples of length less than k with elements of $(s'_1, \ldots, s'_k)$. This allows building an infinite sequence:

$$(s_1, \ldots, s_m, s'_{m+1}, \ldots, s'_k) >^{lex} \ldots >^{lex} (t_1, \ldots, t_p, s'_{p+1}, \ldots, s'_k) >^{lex} \ldots,$$

which contradicts the fact that $>^{lex}$ is well-founded. □

Multiset extension

Lexicographic extensions compare tuples built on components having a natural ordering. But for collections of arbitrary size, in which no such component ordering exists, the notion of multiset is useful. A multiset is a finite collection of elements in which the number of occurrences of identical elements is significant, unlike sets.

Formally, a multiset can be defined through a mapping on the set of natural numbers that counts how many times the element occurs in the multiset:

Definition 4.6 A *multiset* $\mathcal{M}$ on a set $\mathcal{T}$ is given by a map from $\mathcal{T}$ to the set of natural numbers. $\mathcal{M}(\mathcal{T})$ denotes the set of multisets of elements of $\mathcal{T}$.

Notation: A multiset on a set $\mathcal{T}$ is denoted in extension using a set-like notation $\{\ ,\ldots,\ \}$ when its element need to be precised, as in the following example.

Example 4.1 Let $\mathcal{T} = \{a, b, c\}$. The finite multiset $\mathcal{M} = \{a, a, a, b, b\}$ is defined by the mapping $\mathcal{M}(a) = 3$, $\mathcal{M}(b) = 2$, $\mathcal{M}(c) = 0$.

An element t of $\mathcal{T}$ belongs to the multiset $\mathcal{M}$ if $\mathcal{M}(t) > 0$. $\mathcal{M}$ is a sub-multiset of $\mathcal{M}'$ written $\mathcal{M} \subseteq \mathcal{M}'$, if $\mathcal{M}(t) \leq \mathcal{M}'(t)$ for all $t \in \mathcal{T}$. The union and intersection of multisets are defined by the identities:

$$\forall t \in \mathcal{T}, \mathcal{M}_1 \cup \mathcal{M}_2(t) = \mathcal{M}_1(t) + \mathcal{M}_2(t),$$
$$\forall t \in \mathcal{T}, \mathcal{M}_1 \cap \mathcal{M}_2(t) = min(\mathcal{M}_1(t), \mathcal{M}_2(t)).$$

In what follows, we only consider finite multisets and call them simply multisets for short.

Definition 4.7 If $>$ is a quasi ordering on $\mathcal{T}$, its (strict) *multiset extension* $>^{mult}$ is defined by: $\mathcal{M} >^{mult} \mathcal{N}$ if:

- $\mathcal{M} \neq \mathcal{N}$, and
- $\mathcal{N}(t) > \mathcal{M}(t)$ implies $\exists t' \in \mathcal{T}$ such that $t' > t$ and $\mathcal{M}(t') > \mathcal{N}(t')$.

This means that a multiset becomes smaller when we replace one of its elements by a multiset (possibly empty) of smaller elements. An equivalent statement of this definition is the following (notice the limit case $k = 0$):

Definition 4.8 If $>$ is a quasi ordering on $\mathcal{T}$, its (strict) *multiset extension* denoted $>^{mult}$ is defined by:

$$\mathcal{M} = \{s_1, \ldots, s_m\} >^{mult} \mathcal{N} = \{t_1, \ldots, t_n\}$$

if there exist $i \in \{1, \ldots, m\}$ and $1 \leq j_1 < \ldots < j_k \leq n$ with $k \geq 0$, such that:

- $s_i > t_{j_1}, \ldots, s_i > t_{j_k}$ and,
- either $\mathcal{M} - \{s_i\} >^{mult} \mathcal{N} - \{t_{j_1}, \ldots, t_{j_k}\}$ or the multisets $\mathcal{M} - \{s_i\}$ and $\mathcal{N} - \{t_{j_1}, \ldots, t_{j_k}\}$ are equals.

A more operational definition of the multiset extension $>^{mult}$ can be proved equivalent to the previous one:

Proposition 4.3 If $>$ is a quasi ordering on $\mathcal{T}$, its strict multiset extension $>^{mult}$ is the transitive closure of the following relation:

$$\{s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_n\} >^{mult} \{s_1, \ldots, s_{i-1}, t_1, \ldots, t_k, s_{i+1}, \ldots, s_n\}$$

if $k \geq 0$ and $s_i > t_j$ for $j = 1, \ldots, k$.

Even more operational, the following set of deduction rules describe the multiset extension:

Bigger	$\mathcal{M} \cup \{t\} >^{mult} \mathcal{N} \cup \{s\}$	$\mapsto\!\!\!\!\rightarrow$	$\mathcal{M} \cup \{t\} >^{mult} \mathcal{N}$
			if $t > s$
Erase	$\mathcal{M} \cup \{t\} >^{mult} \mathcal{N} \cup \{t\}$	$\mapsto\!\!\!\!\rightarrow$	$\mathcal{M} >^{mult} \mathcal{N}$
Success	$\mathcal{M} \cup \{t\} >^{mult} \emptyset$	$\mapsto\!\!\!\!\rightarrow$	$\mathbf{T}$

in the sense that $\mathcal{M} >^{mult} \mathcal{N}$ if and only if this formula can be reduced to $\mathbf{T}$ by the rules above using the strategy consisting to first normalize with **Erase** and then apply the other rules.

Proposition 4.4 [DM79] If $>$ is well-founded on $\mathcal{T}$, its multiset extension $>^{mult}$ is well-founded on finite multisets of $\mathcal{T}$.

Example 4.2 The multisets of natural numbers $\{1, 1, 2, 3, 3, 3\}$ and $\{3, 1, 2, 3, 3, 1\}$ are equal, but are distinct from $\{1, 2, 3\}$.
$\{3, 3, 1, 2\} >^{mult} \{3, 1\}$, $\{3, 3, 1, 2\} >^{mult} \{3, 2, 2, 2, 2\}$, $\{3, 3, 1, 2\} >^{mult} \{3, 0\} >^{mult} \{3\} >^{mult} \{\}$.
Using the rules above we get for example:
$\{3, 2, 2, 1\} >^{mult} \{3, 2, 1\} \mapsto\!\!\!\!\rightarrow^*_{\mathbf{Erase}} \{2\} >^{mult} \{\} \mapsto\!\!\!\!\rightarrow_{\mathbf{Success}} \mathbf{T}$.

4.2.3 Well-quasi orderings

Definition 4.9 [Kru60] A quasi ordering $\geq$ on a set $\mathcal{T}$ is a *well-quasi ordering* if every infinite sequence $t_1, t_2, \ldots$ of elements of $\mathcal{T}$ contains two elements t_j and t_k such that $j < k$ and $t_j \leq t_k$.

Every total order is a well-quasi ordering, as for example $\mathbf{N}$ with its natural ordering $\geq$. An example of a Noetherian ordering that is not a well-quasi ordering is the following. Consider the divisibility relation on $\mathbf{N}$ (i.e. $x \geq y \Leftrightarrow y$ divises x) then the set of prime numbers is an infinite subset of $\mathbf{N}$ contradicting the previous definition.

Embedding

In fact a very natural example of well-quasi ordering is given by the embedding relation on the set of terms $\mathcal{T}(\mathcal{F})$. It is defined as follows:

Definition 4.10 The *homeomorphic embedding* is a well-quasi ordering on $\mathcal{T}$ denoted $\trianglerighteq_{emb}$ and defined as the smallest reflexive transitive monotonic relation containing the following relation:

$$\forall 1 \leq i \leq n, f(s_1, \ldots, s_n) \trianglerighteq_{emb} s_i.$$

An intuitive idea of the method to build a term t embedded in a term s is to get t by "removing some nodes" to s. The following equivalent definition gives a more procedural point of view:

Definition 4.11 The homeomorphic embedding on $\mathcal{T}$ is defined by:

$$s = f(s_1, \ldots, s_m) \trianglerighteq_{emb} t = g(t_1, \ldots, t_n)$$

if:

- either $f = g$ and $s_i \trianglerighteq_{emb} t_{j_i}$ for all i, $1 \leq i \leq m$ and $1 \leq j_1 < \ldots < j_m \leq n$,
- or $s \trianglerighteq_{emb} t_j$ for some j, $1 \leq j \leq n$.

Example 4.3 $f(h(g(a, h(b))), h(c)) \trianglerighteq_{emb} f(g(a, b), c)$
$---(0 + --1) \trianglerighteq_{emb} --(0 + 1)$.

When $\mathcal{F}$ is finite, Kruskal [Kru54, Kru60] has shown that $\trianglerighteq_{emb}$ is a well-quasi ordering. We will see in section 6.4.1 that this can be extended to a similar result on a more general notion of embedding.

Properties of well-quasi ordering

G. Higman has studied equivalent properties of the above definition of well-quasi orderings. The following result gives the most useful ones for our purpose.

Proposition 4.5 [Hig52] Let $\geq$ a quasi ordering on a set $\mathcal{T}$. The following properties are equivalent:

1. $\geq$ is a well-quasi ordering on $\mathcal{T}$,
2. $\geq$ is well-founded and all $\mathcal{T}$ subsets of pairwise incomparable elements is finite,
3. every infinite sequence $t_1, t_2, \ldots$ of elements of $\mathcal{T}$ contains an infinite increasing subsequence $u_1 \leq u_2 \leq \ldots$

Proof: $3 \Rightarrow 1$ is clear.
$1 \Rightarrow 3$: Consider an infinite sequence $Seq = \{t_1, t_2, \ldots\}$ of elements of $\mathcal{T}$ and call maximal elements of Seq the t_i such that $\forall j > i$, we do not have $t_i \leq t_j$. The set M of maximal elements cannot be infinite, otherwise this would contradict the definition of a well-quasi ordering. Since M is finite, let n be the greatest index such that $t_n \in M$. Then $\exists j > n$, such that t_j is not a maximal element. Let $t_{n_1} = t_j$. Since $\forall n_i > n$, t_{n_i} is not a maximal element, there exists $n_{i+1} > n_i$ such that $t_{n_i} \leq t_{n_{i+1}}$. Thus it is possible to construct an infinite increasing sequence $t_{n_1} \leq \ldots \leq t_{n_i} \leq \ldots$ for $n_1 < \ldots < n_i < \ldots$.
$1 \Rightarrow 2$: Consider an infinite sequence $t_1 > t_2 > \ldots$, it contains an infinite increasing subsequence $u_1 \leq u_2 \leq \ldots$. Let $t_i = u_1$. Then $\forall j > i$, there exists k such that $u_k = t_{j+l}$, otherwise this would contradicts the fact that the sequence of u_k is infinite. Since $u_1 \leq u_k$ and $u_1 = t_i \geq t_j \geq t_{j+l} = u_k$, then $u_1 \simeq u_k$ and $t_j \simeq t_i$. $\square$

Well-quasi ordering are especially important in the context of incrementally building well-founded orderings. As pointed out in [Les89], from a practical point of view, the problem that often arises is the following question: can a well-founded ordering be extended by adding pairs (possibly an infinite number) that are currently incomparable, and still remain well-founded? The point is to forbid the possibility of adding an infinite decreasing chain, therefore an incremental ordering is exactly a well-quasi ordering:

Proposition 4.6 If $\geq$ is a well-quasi ordering on $\mathcal{T}$ then any extension of $\geq$ which is a quasi ordering is also a well-quasi ordering on $\mathcal{T}$.

Further Readings: The reader will find in the paper of J. Kruskal [Kru72] an history of the notion of well-quasi orderings.

4.3 Abstract reduction systems

Many of the basic definitions and properties of rewrite systems can be stated abstractly, namely via binary relations on sets.

Definition 4.12 An *abstract reduction system* is a structure $\langle \mathcal{T}, (\rightarrow_i)_{i\in I}\rangle$ consisting of a set $\mathcal{T}$ and a sequence of binary relations $\rightarrow_i$ on $\mathcal{T}$ indexed by some set I. In the case of only one relation, the index is omitted.

For each binary relation $\rightarrow$, on a set $\mathcal{T}$ whose elements are denoted by $t, t', \ldots$, we also define:

$$\begin{array}{lll} t \leftarrow t' & \text{iff} & t' \rightarrow t \\ t \longleftrightarrow t' & \text{iff} & t \rightarrow t' \text{ or } t \leftarrow t' \\ t \xleftarrow{*} t' & \text{iff} & t' \xrightarrow{*} t \\ t \xleftarrow{+} t' & \text{iff} & t' \xrightarrow{+} t \end{array}$$

For any natural number n, the composition of n steps of $\rightarrow$ or $\longleftrightarrow$ is denoted by $\xrightarrow{n}$ or $\xleftrightarrow{n}$ respectively. Note that $\xrightarrow{0}$ and $\xleftrightarrow{0}$ are nothing else but syntactic equality. As usual, $\xleftrightarrow{+}$ and $\xleftrightarrow{*}$ denote respectively the transitive and the reflexive transitive closure of $\longleftrightarrow$.

Composition of binary relations $\rightarrow_1$ followed by $\rightarrow_2$ will be denoted by $\rightarrow_1 \circ \rightarrow_2$.

4.4 Normalizing abstract reduction systems

Definition 4.13 Let $\langle \mathcal{T}, \rightarrow\rangle$ be an abstract reduction system.

- An element $t \in \mathcal{T}$ is a $\rightarrow$*-normal form* if there exists no $t' \in \mathcal{T}$ such that $t \rightarrow t'$. Furthermore $t \in \mathcal{T}$ has a normal form if $t \xrightarrow{*} t'$ for some normal form t' which is denoted $t\!\downarrow$. A derivation issued from a term t and leading to one of its normal form is denoted $t \xrightarrow{!} t\!\downarrow$.
- The relation $\rightarrow$ is *terminating* (or *strongly normalizing*, or *Noetherian*) if every reduction sequence is finite.
- The relation $\rightarrow$ is *weakly normalizing* (or weakly terminating) if every element $t \in \mathcal{T}$ has a normal form.
- The relation $\rightarrow$ has the *unique normal form property* if for any $t, t' \in \mathcal{T}$, $t \xleftrightarrow{*} t'$ and t, t' are normal forms imply $t = t'$.

Example 4.4 1. The relation $a \rightarrow b$ is terminating, but $a \rightarrow a$ is not.

2. The relation defined by $a \rightarrow b, b \rightarrow a, a \rightarrow c, b \rightarrow d$ is weakly terminating but not terminating. It has not the unique normal form property.

Exercice 22 — [dV91] Let R_0 and R_1 be two abstract reduction systems on the same set, $\rightarrow_0$ and $\rightarrow_1$ the respective reduction relations. Let NF_i be the set of normal forms of R_i for $i = 0, 1$. Prove that R_0 has the unique normal form property if each of the following conditions holds:

1. $\xleftrightarrow{*}_1$ contains $\xleftrightarrow{*}_0$,
2. $\xleftrightarrow{*}_1 \subseteq \xrightarrow{*}_1 \circ \xleftarrow{*}_1$,
3. NF_1 contains NF_0.

4.5 Well-founded ordering and termination

The termination property of an abstract relation associated to an abstract reduction system $\langle \mathcal{T}, \rightarrow\rangle$ is studied in the general framework of well-founded orderings on the set $\mathcal{T}$.

The first straightforward idea to prove termination of $\rightarrow$ relies on the idea of using a well-founded ordering:

Proposition 4.7 [MN70] Let $\langle \mathcal{T}, \rightarrow\rangle$ be an abstract reduction system. The relation $\rightarrow$ is terminating on $\mathcal{T}$ iff there exists a well-founded ordering $>$ over $\mathcal{T}$ such that $s \rightarrow t$ implies $s > t$.

The problem is now the construction of well-founded orderings. We shall focus later on, in Chapter 6, on building explicit well-founded orderings on the set of terms.

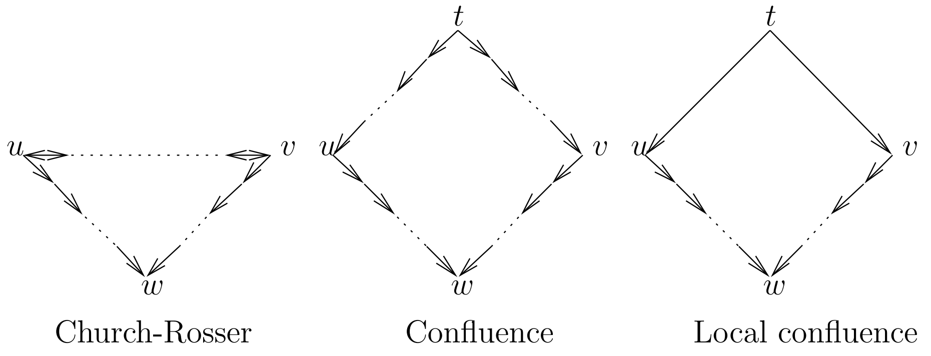


Figure 4.1: Definitions of the relations

4.6 Abstract Church-Rosser property and confluence

In the context of a rewriting relation $\rightarrow_R$, the Church-Rosser property states the relation between replacement of equals by equals and rewriting. More generally, the property is expressed as follows:

Definition 4.14 *A binary relation* $\rightarrow$ *on a set* $\mathcal{T}$ *is* Church-Rosser *if*

$$\xleftrightarrow{*} \subseteq \xrightarrow{*} \circ \xleftarrow{*} .$$

A relation is confluent if it satisfies the so called diamond property schematized in Figure 4.1 and formally defined as follows:

Definition 4.15 The relation $\rightarrow$ is *confluent* on $\mathcal{T}$ if

$$\xleftarrow{*} \circ \xrightarrow{*} \quad \subseteq \quad \xrightarrow{*} \circ \xleftarrow{*} .$$

Theorem 4.2 *The relation* $\rightarrow$ *is confluent iff it is Church-Rosser .*

Proof: The Church-Rosser property obviously implies the confluence. The converse is shown by induction on the number of $\longleftrightarrow$-steps that appear in $\xleftrightarrow{*}$. Assume that it is n, which is denoted by $t \xleftrightarrow{n} t'$.

If $n = 0$, t and t' are equal. But $\xrightarrow{*}$ and $\xleftarrow{*}$ contain equality and the composition of equality with itself is again equality.

Otherwise, let $t \longleftrightarrow t'' \xleftrightarrow{n} t'$. By induction hypothesis, there exists t'_1 such that $t'' \xrightarrow{*} t'_1 \xleftarrow{*} t'$. Now, either $t \rightarrow t''$ and the Church-Rosser property is satisfied, or $t \leftarrow t''$ and by confluence, there exists t'_2 such that $t \xrightarrow{*} t'_2 \xleftarrow{*} t'_1$. Again the Church-Rosser property is satisfied. $\Box$

In the following, $t \downarrow t'$ means that both t and t' have a common descendant.

Proposition 4.8 If $\rightarrow$ is confluent, the normal form of any element is unique, provided it exists.

Proof: Assume an element t has two normal forms t_1 and t_2. Then by confluence $t_1 \downarrow t_2$ and thus $t_1 = t_2$ since both are irreducible. $\Box$

Exercice 23 — Consider a relation $\rightarrow$ which has the unique normal form property and is weakly normalizing. Prove that $\rightarrow$ is confluent.
Answer:

4.6.1 Local confluence

The confluence property has a more local version that only considers two different applications of the relation.

Definition 4.16 The relation $\rightarrow$ is *locally confluent* on $\mathcal{T}$ if:

$$\longleftarrow \circ \longrightarrow \quad \subseteq \xrightarrow{*} \circ \xleftarrow{*} .$$

Local confluence is pictured in Figure 4.1.

Confluence clearly implies local confluence but the converse is not true, as shown by the next example:

Example 4.5 Consider four distinct elements a, b, c, d of $\mathcal{T}$ and the relation defined by $a \to b, b \to a, a \to c, b \to d$ pictured:

$$\begin{tikzcd} a \arrow[r, bend left] \arrow[d] & b \arrow[l, bend left] \arrow[d] \\ c & d \end{tikzcd}$$

The relation, although locally confluent (consider any possible case of ambiguity) and weakly terminating, is not confluent since $c \xleftarrow{*} a \xrightarrow{*} d$ but neither c nor d can be rewritten.

In this example of course, the relation is not terminating since there exists a cycle $a \to b \to a$. Provided the relation terminates, confluence and local confluence are equivalent.

Lemma 4.1 [New42] If $\to$ is terminating, then $\to$ is confluent iff $\to$ is locally confluent.

Proof: Obviously confluence implies local confluence. We prove the converse by Noetherian induction, following Huet Huet's elegant and concise proof [Hue80].

Let $Pred(t)$ be the property:

$$\forall t_1, t_2, t_1 \xleftarrow{*} t \xrightarrow{*} t_2, \exists t'', t_1 \xrightarrow{*} t'' \xleftarrow{*} t_2.$$

Let us prove that $Pred$ is a predicate complete on the well-founded set $(\mathcal{T}, \to)$. For that let us assume that $Pred$ holds for any t' such that $t \xrightarrow{+} t'$ and prove that it holds for t.

Assume that

$$t_1 \xleftarrow{m} t \xrightarrow{n} t_2.$$

If $m = 0$, then $t'' = t_2$, if $n = 0$, then $t'' = t_1$.
Otherwise, as pictured in the diagram in Figure 4.2, consider t'_1 and t'_2 in $\mathcal{T}$ such that:

$$t_1 \xleftarrow{*} t'_1 \leftarrow t \to t'_2 \xrightarrow{*} t_2.$$

Using local confluence, there exists an element t' such that $t'_1 \xrightarrow{*} t' \xleftarrow{*} t'_2$.
Then since $t \to t'_1$, $Pred(t'_1)$ holds. So:

$$\exists t''_1, t_1 \xrightarrow{*} t''_1 \xleftarrow{*} t'.$$

Also since $t \to t'_2$, $Pred(t'_2)$ holds. Thus:

$$\exists t'', t''_1 \xrightarrow{*} t'' \xleftarrow{*} t_2.$$

Which proves that $Pred(t)$ holds.
Applying the principle of Noetherian induction, the property of confluence holds on $\mathcal{T}$, which concludes the proof. □

This allows show that the Church-Rosser property is equivalent to the local confluence property, under the termination assumption. These results are summarized by the following theorem.

Theorem 4.3 *If the relation $\to$ is terminating, the following properties are equivalent:*

1. *$\to$ is Church-Rosser,*
2. *$\to$ is confluent,*
3. *$\to$ is locally confluent,*
4. *for any $t, t' \in \mathcal{T}, t \xleftrightarrow{*} t'$ iff $t\downarrow = t'\downarrow$.*

Proof: (1) and (2) are equivalent by Theorem 4.2, (2) and (3) are equivalent by Lemma 4.1. (4) clearly implies (1). Finally (1) implies (4) by applying twice the Church-Rosser property. □

Definition 4.17 A relation $\to$ is *convergent* if it is confluent and terminating.

Exercice 24 — Prove that the relation $\to$ is convergent iff it is terminating and has the unique normal form property.
Answer: Just apply the result of Exercice 4.6

$$\begin{tikzcd}
 & & t \arrow[dl] \arrow[dr] & & \\
 & t'_1 \arrow[dl, "m-1"'] \arrow[dr, "*"] & & t'_2 \arrow[dl, "*"'] \arrow[dr, "n-1"] & \\
t_1 \arrow[dr, "*"] & & t' \arrow[dl, "*"'] & & t_2 \arrow[ddll, "*"'] \\
 & t''_1 \arrow[dr, "*"] & & & \\
 & & u & &
\end{tikzcd}$$

Figure 4.2: Proof diagram

4.6.2 Confluence without termination

Other notions of confluence appear in the literature. Let $\xrightarrow{0,1}$ denote at most one application of a rewriting step, and $\xleftarrow{0,1}$ the converse relation.

Definition 4.18 Let $\langle \mathcal{T}, \to \rangle$ an abstract reduction system. The relation $\to$ is *strongly confluent* on $\mathcal{T}$ if:

$$\leftarrow \circ \to \subseteq \; \xrightarrow{0,1} \circ \xleftarrow{0,1} .$$

Strong confluence implies confluence [New42]. Confluence of $\to$ is exactly strong confluence of $\xrightarrow{*}$. This property is used in classical proofs of the Church-Rosser property for the lambda-calculus [Bar84].

Another result, based on Noetherian induction, makes use of a well-founded ordering that contains the abstract relation.

Lemma 4.2 [WB83] Let $\langle \mathcal{T}, \to \rangle$ an abstract reduction system such that there exists a well-founded ordering $>$ such that for any $t, t' \in \mathcal{T}$, $t \to t'$ implies $t > t'$. $\to$ is confluent iff for any $t, t', t'' \in \mathcal{T}$, such that $t \to t'$ and $t \to t''$, there exist $t_1, \ldots, t_n$ such that $n \geq 1$, $t > t_i$ for $i = 1, \ldots, n$ and $t' = t_1 \longleftrightarrow \ldots \longleftrightarrow t_n = t''$.

4.6.3 Confluence for weakly normalizing systems

We have seen how the Church-Rosser property can be proved under the (quite strong) hypothesis of local confluence and termination. This leads to the study of completion algorithms which will be presented in the last part of this book.

It is of course of main interest to give a method to establish the confluence of systems that are only weakly normalizing. This has been done in [CG91]. The main idea of the following result is to establish a relation between two abstract reduction systems in such a way that one will inherit of the confluence property of the other.

Theorem 4.4 *[CG91]*

Let $\langle \mathcal{T}, \to \rangle$ be a weakly terminating abstract reduction system and $\langle \mathcal{T}', \to' \rangle$ be a confluent abstract reduction systems such that there exists a mapping π from $\mathcal{T}$ to $\mathcal{T}'$ satisfying:

1. *for all elements r and s in $\mathcal{T}$, if $r \to s$ then $\pi(r) \overset{*}{\longleftrightarrow}{}' \pi(s)$,*
2. *The image with π of a normal form in $\langle \mathcal{T}, \to \rangle$ is a normal form in $\langle \mathcal{T}', \to' \rangle$.*
3. *π is injective on the normal forms in $\langle \mathcal{T}, \to \rangle$.*

Under these conditions, $\langle \mathcal{T}, \to \rangle$ is confluent.

Proof: It is remarquably clear and simple. Let a, b, c elements in $\mathcal{T}$ such that

$$c \xleftarrow{*} a \xrightarrow{*} b$$

Since $\rightarrow$ is weakly terminating, b and c have normal forms $b{\downarrow}$ and $c{\downarrow}$. Because of the first condition on π and since $c{\downarrow} \xleftarrow{*} a \xrightarrow{*} b{\downarrow}$, we get that $\pi(b{\downarrow}) \xleftrightarrow{*} \pi(c{\downarrow})$. Now because of the second condition on π, $\pi(b{\downarrow})$ and $\pi(c{\downarrow})$ are normal forms. But $\langle \mathcal{T}', \rightarrow' \rangle$ being confluent, it follows that $\pi(b{\downarrow}) = \pi(c{\downarrow})$ and since π is assumed to be injective on normal forms, $b{\downarrow} = c{\downarrow}$, which prove that $\langle \mathcal{T}, \rightarrow \rangle$ is confluent. □

This result can be specialized to several cases, in particular when $\mathcal{T} = \mathcal{T}'$ and $\pi = Id$. One can also notice that if $\langle \mathcal{T}', \rightarrow' \rangle$ has the unique normal form property, then under the previous hypothesis on $\langle \mathcal{T}, \rightarrow \rangle$ and π, $\langle \mathcal{T}, \rightarrow \rangle$ has also the unique normal form property. This can be applied to prove the confluence of the $\lambda\beta$-calculus following an idea of Pottinger [Pot81] as developed in [CG91].

Chapter 5

Definition and properties of rewrite systems

5.1 Introduction

In order to mechanize as much as possible equational reasoning, the problem is to find a decision procedure for equational theories. To illustrate the problem, let us consider the example of group theory, defined by the following axioms, where $*$ follows the associative law, e the identity element and $i(x)$ the inverse of x:

$$\begin{array}{rcl} x * e &=& x \\ x * i(x) &=& e \\ (x * y) * z &=& x * (y * z) \end{array}$$

The proof that e is also a left-identity can be done by equational replacement as follows:

$$\begin{array}{rcl} e * x &=& e * (x * e) = e * (x * (i(x) * i(i(x)))) \\ &=& e * ((x * i(x)) * i(i(x))) = e * (e * i(i(x))) \\ &=& (e * e) * i(i(x)) = e * i(i(x)) = (x * i(x)) * i(i(x)) \\ &=& x * (i(x) * i(i(x))) = x * e = x. \end{array}$$

Automating this proof needs to guess which is the right axiom to be applied at each step, in which direction, and possibly to backtrack. The idea of rewriting is to suppress the need for backtracking, first by using (oriented) axioms from left to right only, second by giving enough (oriented) axioms to have the same deduction power than originally. For instance, deciding equality in group theory needs ten oriented axioms (equivalent to the three previous ones), first discovered in the pioneer work of Knuth and Bendix [KB70]

$$\begin{array}{rcl} x * e &\rightarrow& x \\ e * x &\rightarrow& x \\ x * i(x) &\rightarrow& e \\ i(x) * x &\rightarrow& e \\ i(e) &\rightarrow& e \\ i(i(x)) &\rightarrow& x \\ i(x * y) &\rightarrow& i(y) * i(x) \\ (x * y) * z &\rightarrow& x * (y * z) \\ x * (i(x) * y) &\rightarrow& y \\ i(x) * (x * y) &\rightarrow& y \end{array}$$

Of course with this system, proving the previous theorem becomes obvious, because of the existence of the oriented axiom $e * x \rightarrow x$.

In this chapter, rewrite systems are defined, together with properties that rewriting must satisfy in order to provide a decision procedure for equational theories. Concepts and notations are mostly coherent with those in [Bac87, DJ90a, DJ91, HO80].

5.2 Rewrite systems

The central idea of rewriting is to impose directionality in the use of equalities.

Definition 5.1 A *rewrite rule* is an ordered pair of terms denoted $l \to r$. The terms l and r are respectively called the *left-hand side* and the *right-hand side* of the rule.

A *rewrite system* or *term rewriting system* is a (finite or infinite) set of rewrite rules.

Example 5.1 Combinatory Logic, originally devised by Schönfinkel in 1924 and rediscovered by Curry, is an example of a theory with a binary function symbol $\cdot$ (application), constant symbols S, K, I (combinators) and variables. The theory is defined by three axioms that can be applied as rewrite rules, using the following rewrite system CL:

$$\begin{array}{rcl} ((S \cdot x) \cdot y) \cdot z & \to & (x \cdot z) \cdot (y \cdot z) \\ (K \cdot x) \cdot y & \to & x \\ (I \cdot x) & \to & x. \end{array}$$

The symbol $\cdot$ is often omitted for a better readability.

A rule is applied by replacing an instance of the left-hand side by the same instance of its right-hand side, but never the converse, contrary to equalities. Note that two rules are considered to be the same if they only differ by a renaming of their variables. A rewrite system R induces a binary relation on terms called the *rewriting relation* or the *derivability relation.*

Definition 5.2 Given a rewrite system R, a term t *rewrites* to a term t', which is denoted by $t \to_R t'$ if there exists

- a rule $l \to r$ of R,
- a position ω in t,
- a substitution σ, satisfying $t_{|\omega} = \sigma(l)$ and called a *match* from l to $t_{|\omega}$,

such that $t' = t[\omega \hookleftarrow \sigma(r)]$.

When t rewrites to t' with a rule $l \to r$ and a substitution σ, it will be always assumed that variables of l and t are disjoint. So $Dom(\sigma) \cap Ran(\sigma) = \emptyset$. This is not a restriction, since variables of rules can be renamed without lost of generality.
Notation: When either the rule, the substitution and/or the position need to be precised, a rewriting step is denoted by

$$t \to_R^{\omega,\sigma,l\to r} t'.$$

The subterm $t_{|\omega}$ where the rewriting step is applied is called the *redex.* A term that has no redex is said to be *irreducible* for R or in *R-normal form.* The irreducible form of t is denoted by $t{\downarrow}_R$. A rewrite system is *(weakly) normalizing* if any term has a normal form.

When R is a set of rewrite rules $\{rr\}$, the abstract reduction system corresponding to the rewriting relation $\to_R$ is the structure $\langle \mathcal{T}(\mathcal{F}, \mathcal{X}), (\to_{rr})_{rr \in R}\rangle$.

Definition 5.3 A *rewriting derivation* is any sequence of rewriting steps

$$t_1 \to_R t_2 \to_R \dots$$

The *derivability relation* $\xrightarrow{*}_R$ is defined on terms by $t \xrightarrow{*}_R t'$ if there exists a rewriting derivation from t to t'. If the derivation contains at least one step, it is denoted by $\xrightarrow{+}_R$.

Example 5.2 In Combinatory Logic (c.f. Example 5.1), we can derive

$$S(KS)Kxyz \to_{CL} KSx(Kx)yz \to_{CL} S(Kx)yz \to_{CL} Kxz(yz) \to_{CL} x(yz),$$

where the operator $\cdot$ has been omitted. Abbreviating $S(KS)K$ by B, this establishes that $Bxyz \xrightarrow{*}_{CL} x(yz)$ and defines Bxy as the composition $x \circ y$ where x and y are variables representing functions.

Another interesting combinator is SII often abbreviated as Ω and called self-applicator. This name is justified by the following derivation

$$\Omega x = SIIx \rightarrow_{CL} Ix(Ix) \rightarrow_{CL} Ixx \rightarrow_{CL} xx.$$

But the term $\Omega\Omega$ admits a cyclic derivation:

$$\Omega\Omega = SII(SII) \rightarrow_{CL} I(SII)(I(SII)) \rightarrow_{CL} I(SII)(SII) \rightarrow_{CL} SII(SII).$$

It can also be proved that any term F in Combinatory Logic has a fixed point P defined as $\Omega(BF\Omega)$. Indeed

$$P = \Omega(BF\Omega) \xrightarrow{*}_{CL} (BF\Omega)(BF\Omega) \xrightarrow{*}_{CL} F(\Omega(BF\Omega)) = FP.$$

Exercice 25 — In Combinatory Logic (c.f. Example 5.1), define C, W, Y as combinators $C = S(BBS)(KK), W = SS(KI), Y = WS(BWB)$. Develop the following derivations:

$$\begin{array}{rcl} Cxyz & \xrightarrow{*}_{CL} & xzy, \\ Wxy & \xrightarrow{*}_{CL} & xyy, \\ Yx & \xrightarrow{*}_{CL} & x(Yx). \end{array}$$

Answer:

The various relations defined from R have in common to be closed under context and substitution:

Definition 5.4 A binary relation $\rightarrow$ over a set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is *term-closed* (also called sometimes *monotonic*) if it is closed both under context application:

$$\forall s, t, u \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \forall \omega \in \mathcal{D}om(u), s \rightarrow t \text{ implies } u[s]_\omega \rightarrow u[t]_\omega,$$

and under substitutions:

$$\forall s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X}), s \rightarrow t \text{ implies } \sigma(s) \rightarrow \sigma(t) \text{ for any substitution } \sigma.$$

A binary relation $\rightarrow$ over a set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is a *rewrite ordering* if it is transitive, irreflexive and term-closed.

Lemma 5.1 The relations $\rightarrow_R, \leftarrow_R, \longleftrightarrow_R, \xrightarrow{*}_R, \xleftarrow{*}_R, \xleftrightarrow{*}_R, \xrightarrow{+}_R, \xleftarrow{+}_R, \xleftrightarrow{+}_R$ are closed under context and substitutions on $\mathcal{T}(\mathcal{F}, \mathcal{X})$. The relation $\xrightarrow{+}_R$ is a rewrite ordering.

Definition 5.5 For a rewrite system $R = \{l_i \rightarrow r_i\}_{i \in I}$, we denote $=_R$ the equational theory generated by the set of equational axioms $E = \{l_i = r_i\}_{i \in I}$.

An easy characterization of two equivalent terms modulo $=_R$ can be given using $\longrightarrow^R$:

Lemma 5.2 Let $R = \{l_i \rightarrow r_i\}_{i \in I}$ a term rewriting system, then for any two terms t, t', we have $t =_R t'$ iff there exists a finite sequence of terms $t = t_0, t_1, \ldots, t_{n-1}, t_n = t'$ such that for all $i \in [1..n-1]$ either $t_i \longrightarrow^R t_{i+1}$ or $t_{i+1} \longrightarrow^R t_i$.

5.3 A rewriting logic

We now introduce a logic which allows to build all the elements of a rewriting relation generated by a term rewriting system R. It has been presented for example in [Mes92]Meseguer and [Hus88]Hussmann.

For a given set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and a set of rules R, we define the *sequents* as the sentences of the form $t \rightarrow t'$ where t and t are terms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$.

A rewrite theory is a set R of rewrite rules. Each rule $(l \rightarrow r)$ has a finite set of variables $\mathcal{V}ar(l) \cup \mathcal{V}ar(r) = \{x_1, \ldots, x_n\}$ which are recorded in the notation $(l(x_1, \ldots, x_n) \rightarrow r(x_1, \ldots, x_n))$. A theory R entails the sequent $(t \rightarrow t')$, if it is obtained by the finite application of the deduction rules **REW** presented in Figure 5.1.

Lemma 5.3 The rule **Substitution** can be derived from the four other rules of the rewriting logic.

Reflexivity
$$\mapsto\!\!\!\!\twoheadrightarrow$$
$$t \to t$$
if $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$

Congruence $t_1 \to t'_1, \ldots, t_n \to t'_n$
$$\mapsto\!\!\!\!\twoheadrightarrow$$
$$f(t_1, \ldots, t_n) \to f(t'_1, \ldots, t'_n)$$
if $f \in \mathcal{F}_n$

Replacement $t_1 \to t'_1, \ldots, t_n \to t'_n$
$$\mapsto\!\!\!\!\twoheadrightarrow$$
$$l(t_1, \ldots, t_n) \to r(t'_1, \ldots, t'_n)$$
if $l(x_1, \ldots, x_n) \to r(x_1, \ldots, x_n) \in R$

Transitivity $t_1 \to t_2,\ t_2 \to t_3$
$$\mapsto\!\!\!\!\twoheadrightarrow$$
$$t_1 \to t_3$$

Substitution $u(x_1, \ldots, x_n) \to v(x_1, \ldots, x_n), \quad t_1 \to t'_1, \ldots, t_n \to t'_n$
$$\mapsto\!\!\!\!\twoheadrightarrow$$
$$u(t_1, \ldots, t_n) \to v(t'_1, \ldots, t'_n)$$

Figure 5.1: **REW**: The rules of rewrite deduction

If the rule:

Symmetry $t_1 \to t_2$
$$\mapsto\!\!\!\!\twoheadrightarrow$$
$$t_2 \to t_1$$

is added to the four rules **Reflexivity**, **Congruence**, **Replacement** and **Transitivity**, we clearly get equational logic, previously defined in section 2.4. The gap between equational logic and rewriting logic is actually quite deep because of the very strong requirement for symmetry.

Depending of the use of the previous rules, we can build several rewriting relations.

Definition 5.6 For a given rewrite system R, a sequent $t \to t'$ is:

- a *zero-step R-rewrite*, if it can be derived from R by application of the rules **Reflexivity** and **Congruence**. In this case t and t' coincide.
- a *one-step concurrent R-rewrite*, if it can be derived from R by application of the rules **Reflexivity**, **Congruence** and at least one application of **Replacement**. When **Replacement** is applied exactly once, then the sequent is called a one-step *sequential R-rewriting.*
- a *concurrent R-rewrite*, if it can be derived from R by a finite application of the rules in **REW**.

The relation between a one step sequential R-rewriting and the rewrite relation previously defined on terms is made precise in the following lemma.

Lemma 5.4 A sequent $t \to t'$ is a one step sequential R-rewriting iff there exist a rule $l \to r$ in R, a substitution σ and an occurrence ω of t such that $t \to_R^{\omega,\sigma,l\to r} t'$.

Proof: Immediate induction on the form of the proof of the sequent $t \to t'$. $\Box$

This result extends to any concurrent rewriting derivation:

Lemma 5.5 For each sequent $t \to t'$ computed using the rewriting logic relative to a set of rules R:

- either $t = t'$,
- or there exists a chain of one step concurrent R-rewrite:

$$t = t_0 \to t_1 \to \cdots \to t_{n-1} \to t_n = t'.$$

Moreover, in this chain, each step $t_i \to t_{i+1}$ can be chosen sequential.

Proof: This is also a consequence of the form of the proof of the sequent $t \to t'$. □

These last results show the equivalence between the operational definition of rewriting and the sequential rewriting relation obtained from the rewriting logic. The main advantage of the last one is precisely to get rid of an operational definition of rewriting that needs to handle the notion of occurrence, matching and replacement. This allows in particular to get simpler proof.

5.4 Church-Rosser property

Results valid for any abstract reduction system are now specialized to the set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and the rewriting relation $\to_R$. In this context, the Church-Rosser property states the relation between replacement of equals by equals and rewriting.

The relation $\to_R$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is Church-Rosser if

$$\forall t, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X}), t \overset{*}{\longleftrightarrow}_R t' \text{ iff } \exists t''\ t \overset{*}{\longrightarrow}_R t'' \overset{*}{\longleftarrow}_R t'.$$

The confluence property is the key concept to ensure the determinism of the rewriting relation seen as a computation process. It says that two computations starting from a same term will eventually give the same result:

$$\forall t, t_1, t_2 \in \mathcal{T}(\mathcal{F}, \mathcal{X}), t_1 \overset{*}{\longleftarrow}_R t \overset{*}{\longrightarrow}_R t_2 \text{ implies } \exists t''\ t_1 \overset{*}{\longrightarrow}_R t'' \overset{*}{\longleftarrow}_R t_2.$$

A rewrite system R is Church-Rosser, confluent, locally confluent, terminating, convergent on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ if $\to_R$ is Church-Rosser, confluent, locally confluent, terminating, convergent on $\mathcal{T}(\mathcal{F}, \mathcal{X})$.

A rewrite system R is *ground Church-Rosser*, *ground confluent*, *ground locally confluent*, *ground terminating*, *ground convergent* if $\to_R$ is Church-Rosser, confluent, locally confluent, terminating, convergent on the set of ground terms $\mathcal{T}(\mathcal{F})$.

Example 5.3 Right distributivity $(\{(x+y)*z \to (x*z)+(y*z)\ (\vec{Dl})\})$ or left distributivity $(\{x*(y+z) \to (x*y)+(x*z)\ (\vec{Dr})\})$ are confluent rewrite systems. Put together, the set of these two rules defining distributivity (i.e., $\{(x+y)*z \to (x*z)+(y*z), x*(y+z) \to (x*y)+(x*z)\}$) is terminating but not confluent since

$$(x*x)+(x*y)+(y*x)+(y*y) \xleftarrow[\vec{Dr}]{} \xleftarrow[\vec{Dl}]{} (x+y)*(x+y) \xrightarrow{\vec{Dr}} \xrightarrow{\vec{Dl}} (x*x)+(y*x)+(x*y)+(y*y)$$

A rewrite system R provides a decision procedure for an equational theory E if R is finite, convergent and $=_R$ coincides with $=_E$. As we have seen in Section 2.4.6, the word problem is the problem to decide if whether or not an equality $s = t$ between two ground terms follows from E. This is a particular case of provability in E for arbitrary terms. If R is ground convergent, the word problem is decidable by reducing both terms s and t to their normal forms and by testing the syntactic equality of the results.

Of course not every equational theory can be decided by rewriting. Some decidable theories are not finitely presented. Moreover some finitely presented theories are undecidable (the combinatorial logic for instance). Even some finitely presented and decidable theories are not decidable by rewriting.

Example 5.4 Kapur and Narendran [KN85] found the following finite Thue system, made of the only axiom:

$$a(b(a(x))) \quad = \quad b(a(b(x))),$$

with a decidable word problem and without equivalent finite convergent system.

5.5 Reduced systems

Definition 5.7 A rewrite system R is *(inter-)reduced* if for each rule $l \to r \in R$, the right-hand side r is irreducible by R and no term s strictly less than l in the encompassment ordering $\sqsubseteq$ is reducible.

Exercice 26 — If R is convergent, prove that R is reduced iff for any left-hand side l of a rule in R, l is irreducible by other rules with a different left-hand side.
Answer:

The interest of reduced systems relies on the fact that for a given equational theory, there is only one (up to variable renaming) convergent and reduced system contained in a given reduction ordering [BL80, Met83].

Definition 5.8 Let R be a set of rewrite rules and $>$ any reduction ordering. $>$ *contains* R if for any rewrite rule $l \to r$ in R, $l > r$.

5.6 Orthogonal systems

For non-terminating systems, confluence can be however ensured by adding strong syntactic conditions on the left-hand sides.

Definition 5.9 A rewrite system that is both left-linear (the left-hand side of each rule is a linear term) and non-overlapping is called *orthogonal*.

Example 5.5 The system from Combinatory Logic given in Example 5.1 is orthogonal.

Theorem 5.1 *If a rewrite system R is orthogonal, then it is confluent.*

Proof: This is proved in [Hue80], using the fact that the parallel reduction (i.e., parallel application of rules of R at disjoint redexes) is strongly confluent. □

We should notice that the orthogonality hypothesis cannot be weakened in order to get this result. Non-overlapping is clearly needed. Linearity is needed as shown by the following (of course non-terminating) example of rewriting system.

Example 5.6 [Klo80]

$$\begin{array}{lcl} d(x,x) & \rightarrow & e \\ c(x) & \rightarrow & d(x,c(x)) \\ a & \rightarrow & c(a) \end{array}$$

The confluence of orthogonal systems establishes the correctness of the operational semantics of recursive programming language, as [O'D85] for instance.

Several interesting theorems can be proved for orthogonal rewrite systems.

Remind first that a rewrite system is *Noetherian* or *strongly normalizing* if every term has no infinite reductions. A rewrite system is *weakly normalizing* if every term has a normal form (though infinite reductions may exists). Moreover,

Definition 5.10 A rewrite system is *weakly innermost normalizing* if every term has a normal form which can be reached by innermost reduction (i.e., a reduction in which only redexes are selected that do not properly contain other redexes).

These properties are equivalent if the rewrite system is orthogonal:

Theorem 5.2 *[O'D77] For an orthogonal rewrite system R, R is strongly normalizing iff R is weakly innermost normalizing.*

For orthogonal systems, efficient evaluation strategies of normal forms have been studied. First normal forms can be computed by a parallel-outermost strategy. With additional sequentiality requirements, normal forms can be computed without look-ahead.

Theorem 5.3 *[Ken89] Every orthogonal rewrite system has a computable, sequential, normalizing reduction strategy.*

But the problem of finding an optimal strategy that avoids all unnecessary rewrites is undecidable in general.

5.7 Decidability results

Many (most!) properties of rewrite systems are undecidable. For a rewrite system built on a finite signature with finitely many rewrite rules, it is undecidable whether confluence holds and also whether termination holds. However for ground rewrite systems (i.e., with no variable), confluence is decidable [DHLT87] and termination is decidable [HL78b]. But ground confluence is undecidable as shown in [KNO90].

Chapter 6

Termination of rewrite systems

6.1 Introduction

This chapter focusses on proving termination of term rewriting systems specifically. An explicit use of term structure, substitutions and term properties is made here. After stating in general the termination problem, two methods for proving termination will be considered: the first one based on reduction orderings, the second one based on simplification orderings. The second approach is quite syntactic and more restrictive but already powerful enough to deal with a reasonable number of significant examples. Extensions of these methods to termination of rewriting in equivalence classes is then considered.

6.2 Termination

The (weakly) normalizing property of a rewrite system is not enough if infinite computations of normal forms must be avoided. Indeed, a term may have a normal form and nevertheless an infinite rewriting derivation. So a stronger, unfortunately undecidable, property is often needed, namely the *termination property.*

This termination requirement forbids rewrite rules like $f(x, a) \to g(y)$ with a variable in the right-hand side that does not occur on the left (otherwise $f(x, a) \to g(f(x, a)) \to g(g(f(x, a))) \to \ldots$). Also a rule like $x \to g(x)$ whose left-hand side is a variable, cannot belong to a terminating rewrite system, otherwise $x \to g(x) \to g(g(x)) \to \ldots$
These restrictions will be implicitely assumed in the following when speaking about terminating rewrite systems.

A less trivial cases of non-termination is given in the next example.

Example 6.1 The following rewrite system T, due to Toyama [Toy86], is not terminating:

$$\begin{array}{rcl} f(a, b, x) & \to & f(x, x, x) \\ g(x, y) & \to & x \\ g(x, y) & \to & y, \end{array}$$

as shown by the cycle:

$$\begin{tikzcd} f(g(a,b), g(a,b), g(a,b)) \arrow[r] & f(a, g(a,b), g(a,b)) \arrow[r] & f(a, b, g(a,b)) \arrow[ll, bend right] \end{tikzcd}$$

This is also a characteristic example of the non modularity of termination of term rewriting systems (see Chapter 8).

Example 6.2 The rewrite system restricted to the rule:

$$f(f(x)) \quad \to \quad f(g(f(x)))$$

is terminating. This can be proved by using the argument that rewriting makes the number of adjacent f's in any term decrease.

In general, it is undecidable whether a rewrite system is terminating. The idea of the proof is the following: Given any Turing machine $\mathcal{M}$, there exists a rewrite system $R_{\mathcal{M}}$ such that $R_{\mathcal{M}}$ terminates for all terms iff $\mathcal{M}$ halts for all input tapes. Since it is undecidable if a Turing machine halts uniformly, it is also undecidable if rewrite systems terminate [Der85b].

This undecidability result holds even for very particular cases: Huet and Lankford examine the case where both sides of the rules are monadic [HL78b]. Dauchet proved undecidability of one left-linear rule termination [Dau89], which is quite remarquable. However on the positive side, for ground rewrite systems, composed of rules without variables, termination is decidable [Tis89, HL78b].

Exercice 27 — Prove that $- - x \to x$ is terminating. Prove that $-x \to - - -x$ is not terminating.
Answer: Trivial in both cases using a size argument.

Example 6.3 A non-trivial example of non-termination is given by the following rule:

$$-(x+y) \to (- - x + y) + y.$$

There exists an infinite derivation:

$$\begin{aligned} - - (x+y) \to -((- - x+y)+y) &\to (- - (- - x+y)+y)+y \\ &\to (-((- - - - x+y)+y)+y)+y \to \dots \end{aligned}$$

Some abstract properties of the rewriting relation $\to_R$ can be studied in the more general framework of well-founded orderings on an arbitrary set $\mathcal{T}$ as it is done in Chapter 4 on Abstract Reduction Systems.

6.3 Reduction orderings

Proving termination of a rewrite system in full generality requires to look at an infinite set of possible contexts for each instance of a rule. This motivates the concept of *reduction ordering* that embodies in its definition the closure properties by instantiation and by context.

6.3.1 Definition

Definition 6.1 A *reduction ordering* $>$ is a well-founded ordering closed under context and substitution, that is such that for any context $C[\_]$ and any substitution σ, if $t > s$ then $C[t] > C[s]$ and $\sigma(t) > \sigma(s)$.

Lemma 6.1 In a reduction ordering $>$ a term is never smaller than one of its subterm, i.e. we never have $t_{|p} > t$ for $p \in \mathcal{D}om(t), p \neq \Lambda$.

Proof: Assume $t_{|p} > t$, then $t = t[t_{|p}]_p > t[t]_p$ and therefore we can built a sequence of infinitely decreasing terms:

$$t_{|p} > t > t[t]_p > t[t[t]_p]_p > t[t[t[t]_p]_p]_p > \dots$$

contradicting the fact that by definition a reduction ordering is well-founded. □

Lemma 6.2 When a reduction ordering is total then it contains the subterm ordering.

Proof: As a term and any of its subterm are comparable, because of the previous lemma it could only be as $t > t_{|p}$, and therefore the ordering contains the subterm ordering. □

Using a reduction ordering, termination of rewriting can be proved by comparing left and right-hand sides of rules.

Theorem 6.1 *[Lan77] A rewrite system R over the set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is terminating iff there exists a reduction ordering $>$ such that each rule $l \to r \in R$ satisfies $l > r$.*

Proof: If R is terminating on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, define $>$ by $s > t$ if $s \overset{+}{\longrightarrow}{}^{R} t$. Then $>$ is clearly a reduction ordering. Conversely, if there exists a reduction ordering $>$ such that each rule $l \to r \in R$ satisfies $l > r$, any descending chain $t_1 \longrightarrow^R t_2 \longrightarrow^R \dots$ correspond to a descending chain $t_1 > t_2 > t_3 > \dots$ and thus R should be terminating otherwise $>$ would not be well-founded. □

6.3.2 Building reduction orderings using interpretations

It is often convenient to build a reduction ordering by using an homomorphism τ from ground terms to an $\mathcal{F}$-algebra $\mathcal{A}$ with a well-founded ordering $>$. Let f_τ denote the image of $f \in \mathcal{F}$ using τ; τ and $>$ are constrained to satisfy the monotonicity condition:

$$\forall a, b \in \mathcal{A}, \forall f \in \mathcal{F},\ a > b \text{ implies } f_\tau(\ldots, a, \ldots) > f_\tau(\ldots, b, \ldots).$$

Then the ordering $>_\tau$ defined by:

$$\forall s, t \in \mathcal{T}(\mathcal{F}), s >_\tau t \text{ if } \tau(s) > \tau(t),$$

is well-founded.

To compare terms with variables, variables are added to $\mathcal{A}$ producing $\mathcal{A}(\mathcal{X})$ and variables in $\mathcal{X}$ are mapped to distinct variables in $\mathcal{A}(\mathcal{X})$. Then the ordering $>_\tau$ is extended by defining

$$\forall s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X}), s >_\tau t \text{ if } \nu(\tau(s)) > \nu(\tau(t))$$

for all assignment ν of values in $\mathcal{A}$ to variables of $\tau(s)$ and $\tau(t)$. Because $>$ is assume to be well-founded, a rewrite system is terminating if one can find $\mathcal{A}, \tau$ and $>$ as defined above.

The following examples use as algebra $\mathcal{A}$ the natural numbers with the usual ordering $>$, together with exponential interpretations and polynomial interpretations [Lan75a, Lan79b].

Example 6.4 Consider the one-rule system $i(f(x, y)) \rightarrow f(f(i(x), y), y)$ and the following interpretation of i and f respectively by the square and the sum functions on positive natural numbers:

$$\begin{aligned} \tau(i(x)) &= \tau(x)^2 \\ \tau(f(x, y)) &= \tau(x) + \tau(y) \end{aligned}$$

In addition, let $\tau(x) = x$ and $\tau(y) = y$.

The monotonicity condition $a > b$ implies $f_\tau(a) > f_\tau(b)$ is clearly satisfied, since each function is increasing on natural numbers. Now,

$$\begin{aligned} \tau(i(f(x, y))) &= (x + y)^2 = x^2 + y^2 + 2xy \\ \tau(f(f(i(x), y), y)) &= x^2 + 2y. \end{aligned}$$

and for any assignment of positive natural numbers n and m to the variables x and y, $n^2+m^2+2nm > n^2+2m$. So this one-rule system is terminating.

Example 6.5 The following rewrite system

$$\begin{aligned} \ominus \ominus x &\rightarrow x \\ \ominus(x \oplus y) &\rightarrow (\ominus x) \oplus (\ominus y) \\ \ominus(x \otimes y) &\rightarrow (\ominus x) \otimes (\ominus y) \\ x \otimes (y \oplus z) &\rightarrow (x \otimes y) \oplus (x \otimes z) \\ (x \oplus y) \otimes z &\rightarrow (x \otimes z) \oplus (y \otimes z) \end{aligned}$$

has been shown terminating in 1978 by Filman [Fil78] with an exponential mapping τ to natural numbers bigger than 2:

$$\begin{aligned} \tau(\ominus x) &= 2^{\tau(x)} \\ \tau(x \oplus y) &= \tau(x) + \tau(y) + 1 \\ \tau(x \otimes y) &= \tau(x)\tau(y) \\ \tau(c) &= 3 \end{aligned}$$

for all constants $c \in \mathcal{F}$.

The first rule for instance always decreases the interpretation of a term, since $\tau(\ominus(\ominus(x))) = 2^{2^{\tau(x)}}$, and for any natural n assigned to the variable $\tau(x)$, $2^{2^n} > n$. The cases of the other rules are left as exercise to the reader.

Exercice 28 — Let R be the following set of rules that axiomatizes the differentiation operator D with respect to a "variable" X. X and C are considered in this system as constants, while x and y are variables.

$$\begin{array}{rcl}
D(X) & \rightarrow & 1 \\
D(C) & \rightarrow & 0 \\
D(x+y) & \rightarrow & D(x)+D(y) \\
D(x-y) & \rightarrow & D(x)-D(y) \\
D(-x) & \rightarrow & -D(x) \\
D(x*y) & \rightarrow & (x*D(y))+(y*D(x)) \\
D(x/y) & \rightarrow & D(x)/y-((x*D(y))/(y*y)) \\
D(ln(x)) & \rightarrow & D(x)/x
\end{array}$$

Check that the following polynomial interpretation allows proving termination of R:

$$\begin{array}{rcl}
\tau(x+y) & = & \tau(x)+\tau(y) \\
\tau(x*y) & = & \tau(x)+\tau(y) \\
\tau(x-y) & = & \tau(x)+\tau(y) \\
\tau(x/y) & = & \tau(x)+\tau(y) \\
\tau(D(x)) & = & \tau(x)^2 \\
\tau(-x) & = & \tau(x)+1 \\
\tau(ln(x)) & = & \tau(x)+1 \\
\tau(a) & = & 4
\end{array}$$

where a is any constant.

As shown by the previous examples this method is quite powerful but in practice the main difficulty clearly comes from the choice of the interpretation. Some heuristics and methods can be found in [BCL87].

Expressivity of polynomial interpretations

Polynomial interpretations are easy to use but have severe limitations from two point of views. The first one is that if the termination of a term rewriting system can be proved using a polynomial interpretation, then the length of the rewriting derivations is bounded by a double-exponential. Formaly:

Proposition 6.1 [HL89, Lau88]
If R can be proved terminating using a polynomial interpretation then there exists a constant κ such that, for any terms s and t,

$$s \overset{n}{\longrightarrow}^{R} t \Rightarrow n \leq 2^{2^{\kappa.|t|}}.$$

Moreover this bound can be reached as shown by the following example.

Example 6.6 Let R be the following system of rules:

$$\begin{array}{rcl}
c(0,0) & \rightarrow & 0 \\
c(s(x),0) & \rightarrow & s(c(x,x)) \\
c(x,s(y)) & \rightarrow & s(s(c(x,y)))
\end{array}$$

with the following associated interpretation:

$$\begin{array}{rcl}
[0] & = & 2 \\
[s](X) & = & X+1 \\
[c](X,Y) & = & 2X^2+3Y
\end{array}$$

Then we have

$$c(s^m(0),s^n(0)) \overset{K}{\longrightarrow}^{R} \text{ with } K = \frac{m(n+1)}{2}+n+1$$

and thus

$$c^{k+1}((s(s(0)),0),\dots) \overset{*}{\longrightarrow}^{R} c(s^{2^{2^k}}(0),0) \overset{M}{\longrightarrow}^{R} s^{2^{2^{k+1}}}(0) \text{ with } M = 2^{2^{k+1}}+2^{2^k-1}+1.$$

The second limitation of polynomial interpretations, already announced in [HO80], concerns the complexity of the function computed with term rewriting systems whose termination is provable using polynomial interpretation. Assuming the function f completely defined over the natural numbers build on constructors 0 and s by a term rewriting system R, one can define the computed function $\{f\}$ as follows:

$$f(s^{n_1}(0), \ldots, s^{n_p}(0)) \xrightarrow{!}{}^{R} s^{\{f\}(n_1,\ldots,n_p)}(0).$$

Then A. Cichon and P. Lescanne have proved that $\{f\}$ is bounded by a polynomial function [CL91].

6.4 Simplification orderings

6.4.1 Well-quasi-ordering and general embedding

Another approach to prove termination is based on the following remark: given a total rewrite ordering (see definition 5.4), if it is well-founded, then it contains the proper subterm relation $\trianglerighteq_{sub}$: otherwise, if $t_{|\omega} > t$ for some term t and some position ω, then there is an infinite decreasing sequence $t > t[t]_\omega > t[t[t]_\omega]_\omega > \ldots$. Conversely, it can be proved that for a finite $\mathcal{F}$, if the rewrite ordering contains the subterm relation $\trianglerighteq_{sub}$, then it is well-founded. This result however needs a stronger notion than well-foundedness, namely *well-quasi-ordering* formally introduced in the section 4.2.3.

It is thus important to construct well-quasi-orderings. First, any well-quasi-ordering $\geq$ on $\mathcal{F}$ induces a well-quasi-ordering on $\mathcal{T}$ denoted $\trianglerighteq_{\geq}$ and defined as a generalization of the homeomorphic embedding, as follows:

Definition 6.2 The *embedding relation induced by a well-quasi-ordering* $\geq$ on $\mathcal{F}$, denoted $\trianglerighteq_{\geq}$, is the transitive reflexive term-closed closure of the following relation:

$$\begin{array}{lll} f(s_1, \ldots, s_n) & \trianglerighteq_{\geq} & s_i & 1 \leq i \leq n \\ f(s_1, \ldots, s_n) & \trianglerighteq_{\geq} & g(s_{i_1}, \ldots, s_{i_k}) & f \geq g, 1 \leq i_1 < \ldots < i_k \leq n, k \leq n. \end{array}$$

The first property is called *context deletion* and the second is called *subterm deletion*.

The main theorem, generalizing Kruskal's Tree Theorem [Kru60] and due to Higman [Hig52] for a finite set of function symbols $\mathcal{F}$ and to Nash-Williams [NW63] in the general case, states that this embedding relation is a well-quasi-ordering:

Theorem 6.2 *If $\geq$ is a well-quasi-ordering on $\mathcal{F}$, then the induced embedding relation $\trianglerighteq_{\geq}$ is a well-quasi-ordering on ground terms.*

Note that when $\mathcal{F}$ is finite, any quasi-ordering on $\mathcal{F}$, including syntactic equality, is a well-quasi-ordering and that the homeomorphic embedding is a special case of this definition where the well-quasi-ordering on the signature is just the equality of symbols.

6.4.2 Basic definitions and properties

Since any rewrite ordering containing the subterm relation $\trianglerighteq_{sub}$ also contains the homeomorphic embedding, the subterm condition suffices for well-foundedness of term orderings, provided that $\mathcal{F}$ is finite. This is the idea that underlies the notion of *simplification ordering*. In addition to be reduction orderings, simplification orderings enjoy the so-called *subterm property*: for any term t and any subterm u of t, then $t \geq u$. In other words:

Definition 6.3 A *simplification ordering* is a reflexive transitive and term-closed relation $\geq$ on terms, that contains the subterm ordering $\trianglerighteq_{sub}$.

Proposition 6.2 A simplification ordering contains the homeomorphic embedding.

Proof: The homeomorphic embedding is the smallest reflexive transitive term-closed relation satisfying the condition of Definition 4.10. Thus it clearly contains the subterm relation. So it is smaller than any simplification ordering. □

The class of rewrite systems whose termination can be proved using a simplification ordering is important enough to justify a specific terminology and study their properties.

Definition 6.4 A rewrite system R is *simply terminating* if there exists a simplification ordering $\geq$ such that for any rule $l \rightarrow r$ in R, $l > r$.

Theorem 6.3 *[Der82] Assume that the set $\mathcal{F}$ of operator symbols is finite. A rewrite system R is terminating if R is simply terminating.*

Proof: If there exists an infinite sequence $t_1 \rightarrow_R t_2 \ldots \rightarrow_R t_i \ldots$, then $t_1 > t_2 \ldots > t_i \ldots$. Since the homeomorphic embedding is a well-quasi-ordering on terms, there exist a pair of terms t_i and t_j such that $i < j$ and $t_i \trianglerighteq_{emb} t_j$, which implies $t_i \leq t_j$ by Proposition 6.2. But this contradicts the fact that $t_i > t_j$. $\square$

A useful characterization of simple termination is provided by the formalization of embedding with a rewrite system.

Lemma 6.3 [Zan92] Given a set of operator symbols $\mathcal{F}$, let $Emb(\mathcal{F})$ be the following set of rewrite rules

$$f(x_1, \ldots, x_n) \rightarrow x_i$$

for any $f \in \mathcal{F}$ of arity n and $i \in \{1, \ldots, n\}$ The following statements are equivalent:

- R is simply terminating.
- $R \cup Emb(\mathcal{F})$ is simply terminating.
- $R \cup Emb(\mathcal{F})$ is terminating.

Exercice 29 — Prove that if a rewrite system R contains a rule $g \rightarrow d$ such that g is embedded in d, there is no simplification ordering allowing to prove the termination of R. Therefore there is no simplification ordering that allows proving the termination of the rewrite system $f(f(x)) \rightarrow f(g(f(x)))$.

Exercice 30 — Prove that there is no total simplification ordering that allows proving the termination of the following system on the set of ground terms built on the set of operators $\{a, b, f, g\}$:

$$\begin{array}{rcl} f(a) & \rightarrow & f(b) \\ g(b) & \rightarrow & g(a) \end{array}$$

6.4.3 Path orderings

Simplification orderings can be built from a well-founded ordering on the function symbols $\mathcal{F}$ called a *precedence*. A powerful example is the following *multiset path ordering* also called *recursive path ordering*:

Definition 6.5 Let $>_{\mathcal{F}}$ be a precedence on $\mathcal{F}$. The *multiset path ordering* $>_{mpo}$ is defined on ground terms by $s = f(s_1, .., s_n) >_{mpo} t = g(t_1, \ldots, t_m)$ if one at least of the following conditions holds:

1. $f = g$ and $\{s_1, \ldots, s_n\} >_{mpo}^{mult} \{t_1, \ldots, t_m\}$
2. $f >_{\mathcal{F}} g$ and $\forall j \in \{1, \ldots, m\}, s >_{mpo} t_j$
3. $\exists i \in \{1, \ldots, n\}$ such that either $s_i >_{mpo} t$ or $s_i \sim t$
   where $\sim$ means equivalent up to permutation of subterms.

The congruence $\sim$ is precisely defined as:

$$s = f(s_1, .., s_n) \sim t = g(t_1, \ldots, t_m)$$

if $f = g$ and $s_j \sim t_{\pi(j)}$ for some permutation π of $[1, \ldots, n]$.

Notice that the precedence on $\mathcal{F}$ needs not be total and that the condition $f = g$ could be replaced by $f \sim_{\mathcal{F}} g$ where $\sim_{\mathcal{F}}$ is the equivalence induced on $\mathcal{F}$ by $>_{\mathcal{F}}$.

Exercice 31 — Assuming $- >_{\mathcal{F}} +$, prove that:

$$-(1 * (1 + 0)) >_{mpo} (-1) + -(0 + 1).$$

What is the minimal precedence allowing to prove:

$$h(f(a, b)) >_{mpo} f(b, a).$$

Proposition 6.3 [Der82] The multiset path ordering is a simplification ordering.

The difficulty in establishing this last result mainly relies is the transitivity proof.

The definition of the multiset path ordering can be extended to terms with variables by adding the following conditions:

1. two different variables are incomparable,
2. a function symbol and a variable are incomparable.

Example 6.7 Clearly by the subterm relation: $h(p(x,y)) >_{mpo} p(x,y) >_{mpo} y$.
Assuming $a >_{\mathcal{F}} p$, $a(p(x,y),z) >_{mpo} p(x,a(z,y)) >_{mpo} y$ since $a >_{\mathcal{F}} p$ and $a(p(x,y),z) >_{mpo} x$, $a(p(x,y),z) >_{mpo} a(z,y)$, the latter being true since $p(x,y) >_{mpo} y$.

Instead of comparing the multisets of subterms, the next ordering compares them lexicographically.

Definition 6.6 Let $>_{\mathcal{F}}$ be a precedence on F. The *lexicographic path ordering* $>_{lpo}$ is defined on terms by $s = f(s_1,..,s_n) >_{lpo} t = g(t_1,\ldots,t_m)$ if one at least of the following condition holds:

1. $f = g$ and $(s_1,\ldots,s_n) >^{lex}_{lpo} (t_1,\ldots,t_m)$ and $\forall j \in \{1,\ldots,m\}, s >_{lpo} t_j$
2. $f >_{\mathcal{F}} g$ and $\forall j \in \{1,\ldots,m\}, s >_{lpo} t_j$
3. $\exists i \in \{1,\ldots,n\}$ such that either $s_i >_{lpo} t$ or $s_i = t$.

Again, notice that the condition $f = g$ could be replaced by $f \sim_{\mathcal{F}} g$ where $\sim_{\mathcal{F}}$ is the equivalence induced by $>_{\mathcal{F}}$.

Notice also that in contrast with the mpo, the first condition of lpo requires that $\forall j \in \{1,\ldots,m\}, s >_{lpo} t_j$. This can be weakened as presented for exemple in [“T02, Proposition 6.4.8].

Proposition 6.4 [KL80] The lexicographic path ordering is a simplification ordering.

The lexicographic path ordering is extended to terms with variables as the multiset path ordering does.

Example 6.8 Assuming $ack >_{\mathcal{F}} succ$, it is easy to show that:

$$\begin{array}{rcl} ack(0,y) & >_{lpo} & succ(y) \\ ack(succ(x),0) & >_{lpo} & ack(x,succ(0)) \\ ack(succ(x),succ(y)) & >_{lpo} & ack(x,ack(succ(x),y)). \end{array}$$

This shows the termination of the rewriting system defining the Ackermann's function.

Exercice 32 — Prove that the associativity rule $x*(y*z) \to (x*y)*z$ terminates, using a lexicographic path ordering.
Answer: Choose the lexicographic ordering in the good direction and apply the definition of the lpo.

Both orderings may be combined to get the *recursive path ordering with status*. Let us assume that each symbol f in the signature has a status, $Stat(f)$ which can be either lexicographic (lex) or multiset ($mult$).

The equality up to multisets, $=^{mult}$, is defined on terms as follows:

$$s = f(s_1,\ldots,s_n) =_{mult} t = g(t_1,\ldots,t_m)$$

if $f = g$, $m = n$, and:

- either f is not AC and $\forall i = 1,\ldots,n, s_i =^{mult} t_i$,
- or f is AC and the two multisets $\{s_1,\ldots,s_n\}$ and $\{t_1,\ldots,t_m\}$ are equal.

Definition 6.7 Let $>_{\mathcal{F}}$ be a precedence on F. The *recursive path ordering with status* $>_{rpos}$ is defined on terms by $s = f(s_1,..,s_n) >_{rpos} t = g(t_1,\ldots,t_m)$ if one at least of the following condition holds:

1. $f >_{\mathcal{F}} g$ and $\forall j \in \{1,\ldots,m\}, s >_{rpos} t_j$
2. $g >_{\mathcal{F}} f$ and $\exists i \in \{1,\ldots,n\}$ such that either $s_i >_{rpos} t$ or $s_i =^{mult} t$.
3. $f = g$, $Stat(f) = lex$ and
   (a) $\exists i = 1,\ldots,n, s_i >_{rpos} t$ or $s_i =^{mult} t$ or else
   (b) $\exists i = 1,\ldots,n, \forall j < i, s_j =^{mult} t_j$ and $\forall k \in \{1,\ldots,n\}, s >_{rpos} t_k$.
4. $f = g$, $Stat(f) = mult$ and $(s_1,\ldots,s_n) >^{lex}_{rpos} (t_1,\ldots,t_m)$.

Another simplification ordering, well-suited for computations, is the *recursive decomposition ordering* [Les84], so-called because it pre-processes terms by decomposing them, in order to improve efficiency.

6.5 Conclusion

In complex proofs of termination, different orderings are combined. Usual combinations, such as multiset extension or lexicographic combination, have already been presented here. Another simple idea would be to simply combine two terminating rewrite systems by taking their union. But it is not true that the union of two terminating rewrite systems is still terminating, even if the function symbols are disjoint. A survey of the known results about the modularity of the termination property can be found in Chapter 8.
Further Readings: A comprehensive survey of termination is [Der87].

Chapter 7

Generalizations of rewriting

7.1 Introduction

This chapter is devoted to various extensions of the rewriting relation on terms. The two first notions, ordered rewriting and class rewriting, emerged from the problem of equational axioms like commutativity that cannot be oriented without loosing the termination property of reduction. The first proposed solutions in [LB77c, LB77a, LB77b, Hue80, PS81] amounted to define a rewrite relation on equivalence classes of terms and to simulate it by another rewrite relation on terms that transforms a representative element of the equivalence class. The problematic non-orientable axioms are then built in the matching process which becomes equational matching.

Another approach of the same problem is taken in the notion of ordered rewriting which appeared in [BDP89]. There only orientable instances of axioms may be used in the rewrite relation. The problematic non-orientable axioms are kept as equalities.

It appeared later on that these two concepts can be combined to take advantage of theories like associativity and commutativity where equational matching is available. This produces the notion of ordered class rewriting which underlies the simplification mechanism of equational theorem provers like SbREVE [AHM89].

Conditional rewriting started from a quite different motivation issued from abstract data types and algebras with partial functions and exceptions. This algebraic point of view was the base of earlier approaches [PEE81, Rém82, Dro83, Kap84, ZR85, BK86]. However to be able to associate with a conditional rewrite system a decidable and terminating reduction relation, it is necessary to provide a reduction ordering to compare terms involved in the consequence and in the condition of a conditional rewrite rule [JW86, Kap87, DOS87]. Then conditional rewriting has been shown to provide a computational paradigm combining logic and functional programming [Fri85a, DP88, GM86]. Only later the connection between conditional equalities and Horn clauses was exploited. Considering a conditional equality as an equational Horn clause leads to define ordered conditional rewriting [BG91b].

Rewriting with constraints emerged more recently as a unified way to cover the previous concepts by looking at ordering and equations as symbolic constraints on terms. But even further, it provides a framework to incorporate disequations, built-in data types and sort constraints.

7.2 Ordered rewriting

The termination property of a rewrite system is crucial to compute normal forms of terms and reduction orderings have been proposed to ensure termination as soon as, in the rewrite system, every left-hand side is greater than the corresponding right-hand side. This is the key point to choose an orientation for the equality and to use it as a rewrite rule. Two kinds of equalities may cause failure of orientation: the first one is due to equalities like $f(x) = g(y)$ that do not have the same variables in the left and right-hand sides. The second case is due to permutative axioms like commutativity that cannot be oriented without loosing the termination property of the reduction relation. However such non-orientable equalities may sometimes be used for reduction anyway, because some of their instances can be oriented. For instance, considering the commutativity axiom $(x + y = y + x)$, an instance like $(x + f(x) = f(x) + x)$ may be oriented using a lexicographic path ordering. Based on this idea, ordered rewriting does not require to use equalities always in the same direction, but the decreasing property of rewriting with respect to a given ordering has to be always satisfied. This approach needs to define an adequate rewrite relation and a corresponding Church-Rosser property.

7.2.1 Ordered rewrite systems

In ordered rewrite systems, the reduction ordering is made explicit.

Definition 7.1 [BDP89] An *ordered rewrite system*, denoted $(E, >)$, is a set of equalities E together with a reduction ordering $>$.

From the equalities and the ordering, a set of ordered instances can be built. The *ordered rewriting* relation is just rewriting with ordered instances of equalities.

Definition 7.2 Given an ordered rewrite system $(E, >)$, an *ordered instance* of an equality $(g = d) \in E$ (with respect to the ordering $>$), is an equality $(\sigma(g) = \sigma(d))$ such that $\sigma(g) > \sigma(d)$.

A simple example illustrates this notion.

Example 7.1 Consider a commutative monoid with two generators, that are constants a and b. Assuming that $(a * b) > (b * a)$, the ground term $(a * b)$ is reduced, by the commutativity equality $(x * y = y * x)$ to $(b * a)$ but not vice-versa.

Let $E^>$ denote the following set of rewrite rules that are ordered instances of equalities in E:

$$E^> = \{\sigma(g) \to \sigma(d) \mid (g = d) \in E, \ \mathit{and}\ \sigma\ \mathit{s.t.}\ \sigma(g) > \sigma(d)\}.$$

Definition 7.3 Given an ordered rewrite system $(E, >)$, the *ordered rewriting relation* is the rewriting relation $\to_{E^>}$ generated by the set $E^>$.

In a more operational way, this could also be defined as follows.

Definition 7.4 Given an ordered rewrite system $(E, >)$, a term t $(E, >)$*-rewrites* to a term t', which is denoted by $t \to_{E^>} t'$ if there exists

- an equality $l = r$ of E,
- a position ω in t,
- a substitution σ, satisfying $t_{|\omega} = \sigma(l)$ and $\sigma(l) > \sigma(r)$

such that $t' = t[\omega \hookleftarrow \sigma(r)]$.

Note that $s \to_{E^>} t$ implies $s \longleftrightarrow_E t$ and $s > t$.

7.2.2 Church-Rosser property for ordered rewriting

The corresponding Church-Rosser property crucially depends on the reduction ordering.

Definition 7.5 A set of equalities E is said to be *ground Church-Rosser* with respect to a reduction ordering $>$ if for all ground terms t and t' such that $t \overset{*}{\longleftrightarrow}_E t'$, there exists a ground term t'' such that $t \overset{*}{\longrightarrow}_{E^>} t'' \overset{*}{\longleftarrow}_{E^>} t'$.

The Church-Rosser property of E with respect to $>$ means that any proof $t \overset{*}{\longleftrightarrow}_E t'$ of a theorem $(t = t')$ with $t, t' \in \mathcal{T}(\mathcal{F})$, has a normal form, which is a rewrite proof using $\overset{*}{\longrightarrow}_{E^>}$.

Definition 7.6 A proof $t \overset{*}{\longrightarrow}_{E^>} t'' \overset{*}{\longleftarrow}_{E^>} t'$, with t and t' ground terms, is called a *ground rewrite proof.*

Such a proof contains no more subproof $t \longleftrightarrow_E t'$ nor peak $t' \leftarrow_{E^>} t \to_{E^>} t''$. In order to eliminate the $\longleftrightarrow_E$-steps, one needs a condition that ensures that two E-equivalent ground terms are always comparable in some ordering $\gg$ that extends $>$.

Definition 7.7 A reduction ordering $\gg$ that is total on $\mathcal{T}(\mathcal{F})$ is called *ground-total.*

Of course when $\gg$ is a ground-total reduction ordering, for all ground terms t, t' such that $t \longleftrightarrow_E t'$, then either $t \gg t'$ or $t' \gg t$.

We thus assume in what follows, that the reduction ordering $>$ extends to a ground-total reduction ordering $\gg$. This condition is fulfilled in most cases: orderings based on polynomial interpretations satisfy this condition for any set of equalities E [Lan79a, Lan75b]. Also any precedence which is total on the set $\mathcal{F}$ of operator symbols can be extended to such an ordering by the means of a recursive path ordering [Der87].

Note that it would be enough to assume that $\gg$ is a ground-total reduction ordering for E, which means that two E-equivalent ground terms are always comparable.

Example 7.2 [MN90] Consider the set of equalities E:

$$\begin{array}{rcl}(x*y)*z & = & x*(y*z)\\ x*y & = & y*x\\ x*(y*z) & = & y*(x*z)\end{array}$$

and a reduction ordering $>$ total on ground terms and satisfying for all ground terms x, y, z:

$$\begin{array}{rcll}(x*y)*z & > & x*(y*z) & \\ x*y & > & y*x & \text{if} \quad x>y\\ x*(y*z) & > & y*(x*z) & \text{if} \quad x>y\end{array}$$

Then $(E, >)$ is an ordered rewrite system such that E is ground Church-Rosser with respect to a reduction ordering $>$.

Assuming that $>$ is the lexicographic path ordering and that a, b, c are constants such that $c > b > a$,

$$b*(c*(b*a)) \rightarrow_{E^>} b*(c*(a*b)) \rightarrow_{E^>} b*(a*(c*b)) \rightarrow_{E^>} a*(b*(c*b)) \rightarrow_{E^>} a*(b*(b*c)).$$

This ordered rewrite system allows deciding the word problem for associativity and commutativity of $*$.

In this example, with a lexicographic path ordering, the first equality (associativity) is orientable, that is all its ground instances are ordered, so it could be replaced once for all by a rewrite rule. This leads to a more liberal definition of ordered rewrite system, where equalities are now split into two parts: an orientable part that gives a rewrite system contained in the reduction ordering, and a non-orientable part.

Example 7.3 [MN90] Groups of exponent two can be defined by the following ordered rewrite system composed of the set of equalities and rewrite rules E

$$\begin{array}{rcl}x*y & = & y*x\\ x*(y*z) & = & y*(x*z)\\ (x*y)*z & \rightarrow & x*(y*z)\\ x*x & \rightarrow & 1\\ x*(x*y) & \rightarrow & y\\ x*1 & \rightarrow & x\\ 1*x & \rightarrow & x\end{array}$$

and a reduction ordering $>$ total on ground terms and satisfying for all ground terms x, y, z:

$$\begin{array}{rcll}x*y & > & y*x & \text{if} \quad x>y\\ x*(y*z) & > & y*(x*z) & \text{if} \quad x>y\end{array}$$

and containing the rewrite rules.

7.3 Class rewriting

To deal with the problem of non-orientable equalities like commutativity, the solution proposed in this section is to quotient the set of terms by the congruence generated by these equalities, called axioms, and to rewrite in equivalence classes. This requires the elaboration of new abstract concepts, namely the notion of *class rewrite systems* (also called equational term rewriting systems) and *rewriting modulo a set of axioms* in which the matching takes into account non oriented equalities. Adequate notions of *confluence and coherence modulo* a set of equalities [Hue80, Jou83, JK86c] must be defined for expressing abstract properties of such rewriting relations.

7.3.1 Class rewrite systems

Given a set of axioms A, let $\overset{*}{\longleftrightarrow}_A$ be the generated congruence relation. The notion of rewriting formalized hereafter can be understood as making a set of rules R computing in equivalence classes modulo $\overset{*}{\longleftrightarrow}_A$.

Definition 7.8 A *class rewrite system* R/A is composed of a set of rewrite rules R and a set of equalities A, such that A and R are disjoint sets.

It may be sometimes useful to distinguish between A and $\vec{A} = \{(g \to d), (d \to g) | (g = d) \in A\}$. Note that $\longleftrightarrow_A = \to_{\vec{A}}$.

Example 7.4 A class rewrite system for abelian groups, i.e. associative and commutative groups, is given by the following sets of rewrite rules and equalities:

$$\begin{array}{rcl} x+0 & \to & x \\ x+(0+y) & \to & x+y \\ x+(-x) & \to & 0 \\ x+((-x)+y) & \to & y \\ --x & \to & x \\ -0 & \to & 0 \\ -(x+y) & \to & (-x)+(-y) \end{array}$$

$$\begin{array}{rcl} x+y & = & y+x \\ (x+y)+z & = & x+(y+z) \end{array}$$

The rewrite relation defined below applies to a term if there exists a term in the same equivalence class modulo A that is reducible with a rewrite rule of R.

Definition 7.9 Given a class rewrite system R/A, the term t (R/A)-*rewrites* to t', denoted $t \to_{R/A} t'$, if $t \overset{*}{\longleftrightarrow}_A u[\omega \hookleftarrow \sigma(l)]$ and $t' \overset{*}{\longleftrightarrow}_A u[\omega \hookleftarrow \sigma(r)]$, for some rule $l \to r \in R$, some term u, some occurrence ω in u and some substitution σ.

However this rewrite relation is not completely satisfactory from a more operational point of view, for the following reasons:

- Even if R is finite and $\overset{*}{\longleftrightarrow}_A$ decidable, $\to_{R/A}$ may not be computable since equivalence classes modulo A may be infinite or not computable. For instance, the axiom $-x = x$ generates infinite equivalence classes.
- The relation $\to_{R/A}$ does not terminate if R is non-empty and A contains an axiom like idempotency $x + x = x$ where a lone variable occurs on one side and several times on the other. Of course, this produces the infinite sequence from a left-hand side of a rule $(l \to r) \in R$:

  $$l \overset{*}{\longleftrightarrow}_A l+l \to_R r+l \overset{*}{\longleftrightarrow}_A r+(l+l) \to_R r+(r+l)....$$

  Other axioms that cause non-termination are equalities like $x * 0 = 0$ where a variable occurs on one side and not on the other. Then $0 \longleftrightarrow_A l * 0 \to_R r * 0 \longleftrightarrow_A 0$ is a cycle for $\to_{R/A}$. Indeed, if such axioms are present, they must be considered as rewrite rules and put in R.

To circumvent these problems with $\to_{R/A}$, especially to avoid scrutiny through equivalence classes, the idea is to use a weaker relation. Several approaches follow this idea and four of them can be mentioned:

- Huet's approach [Hue80] uses standard rewriting $\to_R$ but is restricted to left-linear rules.
- Peterson and Stickel's approach [PS81] uses *rewriting modulo A*, denoted $\to_{R,A}$, and needs matching modulo A.
- Pedersen's approach [Ped84] uses a restricted version of matching modulo A, confined to variables.
- Jouannaud and Kirchner's method [JK86c] uses standard rewriting with left-linear rules and rewriting modulo A with non-left-linear rules, mixing advantages of the two first methods.

Several abstract properties are common to these relations and are formalized in the next section for a general relation $\to_{R_A}$ that may by any of these relations. However at least to support intuition, and because this is the most commonly used rewrite relation for class rewrite systems, the Peterson and Stickel's relation is defined more precisely.

Definition 7.10 Given a class rewrite system R/A, a term t (R, A)-*rewrites* to a term t', which is denoted by $t \to_{R,A} t'$ if there exists

- a rule $l \to r$ of R,
- a position ω in t,
- a substitution σ, satisfying $t_{|\omega} \xleftrightarrow{*}_A \sigma(l)$ and called a *match modulo A* from l to $t_{|\omega}$,

such that $t' = t[\omega \hookleftarrow \sigma(r)]$.

Notation: When either the rule, the substitution and/or the position need to be precised, a rewriting step is denoted by

$$t \to^{\omega,\sigma,l\to r}_{R,A} t'.$$

Note that in this relation t and $t[\sigma(l)]_\omega$ differ only by A-equality steps below the position ω. So clearly $\to_{R,A}$ is included in $\to_{R/A}$.

7.3.2 Church-Rosser results

A class rewrite system R/A defines an abstract reduction system and all concepts and properties of abstract reduction systems are available. We rephrase some of them in the context of class rewrite systems.

Definition 7.11 The class rewrite system R/A is *terminating* if $\to_{R/A}$ is terminating.

A term irreducible for $\to_{R/A}$ is said in R/A-normal form. The R/A-normal form of a term t is denoted $t\downarrow_{R/A}$.

Definition 7.12 The class rewrite system R/A is *Church-Rosser* if

$$\xleftrightarrow{*}_{R\cup A} \quad \subseteq \quad \xrightarrow{*}_{R/A} \circ \xleftrightarrow{*}_A \circ \xleftarrow{*}_{R/A} .$$

Whenever the class rewrite system R/A is Church-Rosser and terminating, for any terms t, t', $t \xleftrightarrow{*}_{R\cup A} t'$ iff $t\downarrow_{R/A} \xleftrightarrow{*}_A t'\downarrow_{R/A}$. In other words, when R/A is Church-Rosser and terminating, $\xleftrightarrow{*}_{R\cup A}$ is decidable by checking A-equality of R/A-normal forms. This result can be understood as a method for proving equational theorems in a theory described by a class rewrite system. Then to be effective, this method assumes two conditions: A-equivalence must be decidable and R/A-normal forms must be computable. Since this is not always true with the relation $\to_{R/A}$, other computable rewrite relations must be considered. Let $\to_{R_A}$ (or R_A for short) be any (rewriting) relation satisfying $\to_R \;\subseteq\; \to_{R_A} \;\subseteq\; \to_{R/A}$.

The definition of a Church-Rosser property for R_A takes the following form:

Definition 7.13 The rewriting relation R_A is *Church-Rosser modulo A* if

$$\xleftrightarrow{*}_{R\cup A} \quad \subseteq \quad \xrightarrow{*}_{R_A} \circ \xleftrightarrow{*}_A \circ \xleftarrow{*}_{R_A} .$$

In other words, this property means that any theorem $t = t'$ with an equational proof using $R \cup A$ has also a so-called rewrite proof.

Definition 7.14 A proof of $(t = t')$ is a *rewrite proof* for R_A if $\exists t_1, t'_1$,

$$t \xrightarrow{*}_{R_A} t_1 \xleftrightarrow{*}_A t'_1 \xleftarrow{*}_{R_A} t'$$

Exercice 33 — Prove that If $R_1 \cup R_2$ is Church-Rosser then R_1/E_2 is Church-Rosser modulo E_2, where E_2 is obtained from R_2 by treating the rules as equations.
Answer: The proof is due to P. Narendran, M. Subrahmaniam and Q. Guo.

$$\begin{array}{rcl}
\xrightarrow{*}_{R_1\cup R_2} & = & (\xrightarrow{*}_{R_2} \circ \longrightarrow_{R_1} \circ \xrightarrow{*}_{R_2})^* \cup \xrightarrow{*}_{R_2} \\
 & \subseteq & (\longleftrightarrow *_{E_2} \circ \longrightarrow_{R_1} \circ \longleftrightarrow *_{E_2})^* \cup \longleftrightarrow *_{E_2} \\
 & = & \longleftrightarrow *_{R_1/E_2} \cup \longleftrightarrow *_{E_2}
\end{array}$$

Thus

$$\xrightarrow{*}_{R_1\cup R_2} \circ \xleftarrow{*}_{R_1\cup R_2} \quad \subseteq \quad (\xrightarrow{*}_{R_1/E_2} \cup \longleftrightarrow *_{E_2}) \circ (\longleftrightarrow *_{E_2} \cup \xrightarrow{*}_{R_1/E_2})$$

Exercice 34 — It is not the case that if R/A is Church-Rosser then R, A is Church-Rosser. Prove this assertion using the following counterexample where

$$A: \quad a + (b + c) = (a + b) + c$$

$$R: \quad (a + b) \to c$$

Answer: For R, A to be Church-Rosser it must be the case that for any two terms s and t, if $(s, t) \in \xleftrightarrow{*}_{R\cup A}$ then $(s, t) \in \xrightarrow{*}_{R,A} \circ \xleftrightarrow{*}_{A} \circ \xleftarrow{*}_{R,A}$. Let s be $a + (b + b)$ and t be $c + b$. $(a + (b + b), c + b) \in \xleftrightarrow{*}_{R\cup A}$, since $(a + (b + b)) \xleftrightarrow{*}_{A} ((a + b) + b) \to_R (c + b)$. But $(a + (b + b)) \not\in \xrightarrow{*}_{R,A} \circ \xleftrightarrow{*}_{A} \circ \xleftarrow{*}_{R,A}$, since $(a + (b + b))$ and $(c + b)$ are irreducible with respect to $\to_{R,A}$ and are not A equivalent.
The system

$$R': \quad a + (b + c) \to (a + b) + c$$

$$(a + b) \to c$$

obtained by orienting the equation in A is convergent. Hence R/A is Church-Rosser modulo A by the result of the previous exercise 7.3.2.

Confluence of R_A is not enough to be equivalent to the Church-Rosser property, as in the case where $A = \emptyset$. Coherence, another diamond property, is needed.

Definition 7.15

- The rewriting relation R_A is *confluent modulo A* if

$$\xleftarrow{*}_{R_A} \circ \xrightarrow{*}_{R_A} \quad \subseteq \quad \xrightarrow{*}_{R_A} \circ \xleftrightarrow{*}_{A} \circ \xleftarrow{*}_{R_A} .$$

- The rewriting relation R_A is *coherent modulo A* if

$$\xleftarrow{*}_{R_A} \circ \xleftrightarrow{*}_{A} \quad \subseteq \quad \xrightarrow{*}_{R_A} \circ \xleftrightarrow{*}_{A} \circ \xleftarrow{*}_{R_A} .$$

- The rewriting relation R_A is *locally coherent with R modulo A* if

$$\leftarrow_{R_A} \circ \to_{R} \quad \subseteq \quad \xrightarrow{*}_{R_A} \circ \xleftrightarrow{*}_{A} \circ \xleftarrow{*}_{R_A} .$$

- The rewriting relation R_A is *locally coherent with A modulo A* if

$$\leftarrow_{R_A} \circ \longleftrightarrow_{A} \quad \subseteq \quad \xrightarrow{*}_{R_A} \circ \xleftrightarrow{*}_{A} \circ \xleftarrow{*}_{R_A} .$$

The following lemma enlights the role of local coherence with A. Any term A-equivalent to a term in R_A-normal form, is itself in R_A-normal form, provided R/A is terminating.

Lemma 7.1 Assume that R/A terminates and that R_A is locally coherent with A modulo A. Then for any terms t, t', if $t \xleftrightarrow{*}_{A} t' \downarrow_{R_A}$ then t is R_A-irreducible.

Proof: By induction on the number n of steps $\longleftrightarrow_A$. If $n = 0$, the result is clear. Else $t \longleftrightarrow_A t_1 \xleftrightarrow{*}_{A} t' \downarrow_{R_A}$ and by induction hypothesis, t_1 is R_A-irreducible. Now assume that t is R_A-reducible to t''. Then by local coherence of R_A with A modulo A, $t'' \xrightarrow{*}_{R_A} \xleftrightarrow{*}_{A} t_1$. Thus $t \xrightarrow{+}_{R/A} t$, which contradicts the termination of R/A. □

This lemma is useful to prove the next theorem that relates the different properties.

Theorem 7.1 *If R/A is terminating, the following properties are equivalent:*

1. *R_A is Church-Rosser modulo A.*

2. *R_A is confluent modulo A and R_A is coherent modulo A.*

3. *R_A is locally coherent with R modulo A and locally coherent with A modulo A.*

4. *$\forall t, t'$, $t \xleftrightarrow{*}_{R\cup A} t'$ iff $t \downarrow_{R_A} \xleftrightarrow{*}_{A} t' \downarrow_{R_A}$.*

Proof: Note that property (4) asserts that the Church-Rosser property modulo A is true when computing R_A-normal forms of t and t', provided R/A is terminating. This is perfectly similar to the usual case of an empty set A of equalities: in that case the Church-Rosser property can be checked on normal forms, provided R is terminating.

(2) $\Rightarrow$ (3) is clear and does not use the termination hypothesis.

(4) $\Rightarrow$ (1) is straightforward, but (4) assumes termination.

(1) $\Rightarrow$ (2) : the proofs are straightforward, and the termination hypothesis implies that actually $\xleftarrow{*}_{R_A} \circ \xleftrightarrow{*}_A \subseteq \xrightarrow{*}_{R_A} \circ \xleftrightarrow{*}_A \circ \xleftarrow{+}_{R_A}$.

(3) $\Rightarrow$ (4) : the right part of the equivalence clearly implies the left. Let us prove the converse by multiset induction on $\rightarrow_{R/A}$: let $\mathcal{M} = \{t_0, t_1, ..., t_{n+1}\}$ be a multiset of at least two elements (the one element case is straightforward) such that $t = t_0 \longleftrightarrow_{R\cup A} t_1 ... t_n \longleftrightarrow_{R\cup A} t_{n+1} = t'$. The basic (one step) case being obvious, we consider the general one. Three cases are to be distinguished according to the last equality step $t_n \longleftrightarrow_{R\cup A} t_{n+1}$.

- $t_{n+1} \rightarrow_R t_n$: since $t_n \downarrow_{R_A}$ is a R_A-normal form of t_{n+1}, the result follows from the induction hypothesis applied to the multiset $\mathcal{M} - \{t_{n+1}\}$.
- $t_n \longleftrightarrow_A t_{n+1}$: the result follows from either the induction hypothesis applied to the multiset $\mathcal{M} - \{t_{n+1}\}$ if t_n, and thus t_{n+1}, is R_A-irreducible, or else it follows from the local coherence of R_A and from the induction hypothesis applied to the multiset $\{t_0, ..., t_n, t''_n, ..., t'_n, ..., t'_{n+1}\}$ which is strictly smaller than $\mathcal{M}$, since its terms are all proper descendents of t_{n+1} for $\rightarrow_{R/A}$.
- $t_n \rightarrow_R t_{n+1}$: we first apply the induction hypothesis to the multiset $\mathcal{M} - \{t_{n+1}\}$. As t_n is reducible by R (therefore by R_A), we can apply local coherence modulo A of R_A with R. We end the proof by applying the induction hypothesis to the multiset $\{t_n \downarrow_{R_A}, ..., t''_n, ..., t'_n, ..., t'_{n+1}, ..., t_{n+1} \downarrow_{R_A}\}$.

□

Theorem 7.1 is false if termination of R_A is assumed in place of termination of R/A, as proved by the following counterexamples where local properties are true but global ones are not. Note that these examples are similar to [Hue80], with added complexity to ensure that they are not coherent.

Example 7.5 Assume given a finite set of terms $\{t_0, t_1, t_2, t_3, t_4, t_5\}$ and let R_A be the relation given by: $\{t_1 \rightarrow_{R_A} t_0, t_2 \rightarrow_{R_A} t_1, t_3 \rightarrow_{R_A} t_4, t_4 \rightarrow_{R_A} t_5\}$ and let $\longleftrightarrow_A$ be given by: $\{t_1 \longleftrightarrow_A t_2, t_2 \longleftrightarrow_A t_3, t_3 \longleftrightarrow_A t_4\}$.

R_A is locally coherent modulo A with R and A, its confluence modulo A is trivially satisfied, but coherence modulo A is not. Moreover R/A is clearly non-terminating, but R_A is terminating.

Similarly, we get a counterexample where local coherence of R with R modulo A is true but confluence modulo A is not satisfied.

Example 7.6 Assume given a finite set of terms $\{t_0, t_1, t_2, t_3, t_4, t_5, t_6\}$ and let R be the relation given by: $\{t_1 \rightarrow_R t_0, t_2 \rightarrow_R t_1, t_3 \rightarrow_R t_2, t_3 \rightarrow_R t_4, t_4 \rightarrow_R t_5\}$ and let $\longleftrightarrow_A$ be given by: $\{t_1 \longleftrightarrow_A t_2, t_2 \longleftrightarrow_A t_3, t_3 \longleftrightarrow_A t_4, t_4 \longleftrightarrow_A t_5\}$.

R is locally coherent modulo A with R and A, but neither confluence modulo A nor coherence modulo A is satisfied. Moreover R/A is clearly non-terminating, but R is terminating.

The following exercise provides another counter-example due to P. Narendran.

Exercice 35 — Consider the signature with constants symbols a, b, c, d, e, i and a binary operator $+$. Let A be the set of ground axioms:

$$\begin{aligned}(a+b)+d &= (a+c)+d\\ (a+c)+d &= e\end{aligned}$$

and R be the set of ground rewrite rules:

$$\begin{aligned}c &\rightarrow b\\ a+b &\rightarrow i\end{aligned}$$

Prove the following properties:

1. R, A is locally coherent with R modulo A and locally coherent with A modulo A.

2. R, A is terminating.
3. R/A is not terminating.
4. R/A is not Church-Rosser modulo A

Answer:

1. R, A is locally coherent with R modulo A and locally coherent with A modulo A.

   Note that $\rightarrow_{R,A}$ is equivalent to $\rightarrow_R$. Hence for local confluence, it suffices to show that $\leftarrow_R \circ \rightarrow_R \subseteq \xrightarrow{*}_{R,A} \circ \xleftrightarrow{*}_A \circ \xleftarrow{*}_{R,A}$. This is a consequence of local confluence of R, since R is a reduced ground rewriting system with no critical pairs.

   For local coherence, let $(s, s') \in \leftarrow_R \circ \longleftrightarrow_A$. Therefore there exists a term m such that $m \rightarrow_R s$ and $m \longleftrightarrow_A s'$. Let ω and υ be any two positions in m such that $m[r]_\omega = s$ for some rule $l \rightarrow r \in R$ and $m[d]_\upsilon = s'$ for some $g = d \in A$.

   If the positions ω and υ are disjoint, then the $\rightarrow_R$ and $\longleftrightarrow_A$ commute and the property is satisfied. Otherwise one position is a prefix of the other. It is easy to see from A and R that $\omega \leq \upsilon$ is impossible. Hence $\upsilon \leq \omega$ and we can assume w.l.o.g. that $\upsilon = \Lambda$.

   (a) If $m = (a+b)+d$, $s' = (a+c)+d$. The only redex available for rewrite is $(a+b)$. Hence $s = i+d$. Since $s' \rightarrow_R (a+b)+d \rightarrow_R i+d$, local coherence is satisfied.

   (b) If $m = (a+c)+d$, $s' = e$. Again by a similar argument as in the previous case $s = (a+b)+d \xleftrightarrow{*}_A s'$, so local coherence is satisfied.

   The above two are the only two possible cases. Therefore, $\rightarrow_{R,A}$ is locally coherent.

2. R, A is terminating. Given any term t there exists no position ω in t such that $t_{|\omega} \xleftrightarrow{+}_A \sigma(l)$, for any substitution σ and rule $l \rightarrow r \in R$. Hence any rewrite sequence using R, A can be embedded in a rewrite sequence using $\rightarrow_R$. Since it can be trivially shown that $\rightarrow_R$ is terminating, R, A is terminating.

3. R/A is not terminating due to the loop:

$$(a+b)+d \longleftrightarrow_A (a+c)+d \rightarrow_R (a+b)+d$$

4. R/A is not Church-Rosser modulo A: Let $s = (i+d)$ and $t = e$.

$$(a+c)+d \longleftrightarrow_A (a+b)+d \rightarrow_R i+d \quad \textit{and} \quad (a+c)+d \xleftrightarrow{*}_A e$$

   But $i+d$ and e both are irreducible with respect to $\rightarrow_{R,A}$ and are not A-equal.

Exercice 36 — Assume that $\rightarrow_R \subseteq \rightarrow_{R_A} \subseteq \rightarrow_{R/A}$. Prove the following properties:

1. R Church-Rosser modulo A implies R_A Church-Rosser modulo A implies R/A Church-Rosser.
2. R/A terminating implies R_A terminating implies R terminating.
3. If R_A is Church-Rosser modulo A and R/A terminating, then any term t is R_A-irreducible iff t is R/A-irreducible. Deduce that $t\downarrow_{R_A}$ is in normal form for R/A, and $t\downarrow_{R/A}$ is in normal form for R_A.

Answer:

Exercice 37 — The following counter-example and its proof are due to P. Narendran, M. Subrahmaniam and Q. Guo.

It is not the case that if $R_1 \cup R_2$ is convergent then $E_2 \setminus R_1$ is Church-Rosser where E_2 is obtained from R_2 by treating the rules as equations. Consider the following example :

$$R_1 : g(f(x)) \rightarrow x$$

$$R_2 : g(h(x)) \rightarrow h(g(x))$$

Thus $E_2 = \{g(h(x)) = h(g(x))\}$.

Answer: R_1, E_2 is not Church-Rosser. Let s be $g(h(f(x)))$ and t be $h(x)$.
With the precedence on the function symbols being given as $g >_{\mathcal{F}} h$, termination of $R_1 \cup R_2$ can be easily seen using a recursive path ordering. $R_1 \cup R_2$ is locally confluent since it is a reduced rewrite system with no critical pair.

Exercice 38 — The following counter-example and its proof are due to P. Narendran, M. Subrahmaniam and Q. Guo.

It is not the case that if R/A is Church-Rosser then R is locally coherent with respect to A. Prove this assertion using the counter-example given below.

$$A : a = b$$

$$R : f(x,x) \rightarrow x$$

Answer: R is not locally coherent with respect to S: Consider the term $f(b,b)$. $f(b,b) \rightarrow_R b$ and $f(b,b) \longleftrightarrow_A f(a,b)$. Since b and $f(a,b)$ are both irreducible with respect to $\rightarrow_R$ and are not A equivalent, $(b, f(a,b))$ provides the counter-example for local coherence.
R/A is Church-Rosser: This can be easily proved using Exercise 7.3.2. This is also a consequence of the proof that R, A is Church-Rosser:

- R/A is terminating since $\rightarrow_R$ decreases size and $\longleftrightarrow_A$ preserves size.
- R, A is locally coherent and locally confluent: $CP_A(R, A)$ is empty since $f(x, x)$ cannot be A-unified with a or b. The only critical pair in $CP_A(R, R)$ is trivial since $f(x, x) \overset{*}{\longleftrightarrow}_A f(y, y)$ iff $x \overset{*}{\longleftrightarrow}_A y$.

7.3.3 Termination

The termination of R/A is thus a crucial condition to ensure. Unfortunately there is no general method for proving it for any set of axioms A. Researches have been pursued in two different directions: on one hand, finding ad-hoc mechanisms for proving termination for a given set of axioms in A, especially for associativity and commutativity; on the other hand, proposing properties for reducing the termination of R/A to the termination of R.

Ad-hoc mechanism for associative commutative theories

The first method, already proposed by Lankford in 1975, further developped in [BCL87] and currently implemented in REVE for instance, is based on *polynomial interpretations*. The idea is to interpret terms by polynomials using an homorphism τ from the set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ into an $\mathcal{F}$-algebra $\mathcal{A}$ with a well-founded ordering $>$. For that, a multivariate integer polynomial $f_\tau(x_1, ..., x_n)$ is associated to each n-ary function symbol f. The choice of coefficients must satisfy the monotonicity condition: $\forall a, b \in \mathcal{A}$, $a > b$ implies $f_\tau(...a...) > f_\tau(...b...)$. The interpretation of an AC-operator ought to be an AC polynomial, so must be linear (i.e. of the form $x + y + n$) or quadratic (i.e. of the form $mx + ny + p$). Moreover terms must be mapped onto nonnegative integers only. Then to each rule $l \rightarrow r \in R$, the polynomial $\tau(l) - \tau(r)$ must be positive for all values of variables greater than the minimal value of a ground term.

The method can be better explained on an example.

Example 7.7 Consider the class rewrite system BR/AC given by the following set BR of rewrite rules

$$\begin{array}{rcl} x \oplus 0 & \rightarrow & x \\ x \oplus x & \rightarrow & 0 \\ x \wedge 0 & \rightarrow & 0 \\ x \wedge 1 & \rightarrow & x \\ x \wedge x & \rightarrow & x \\ x \wedge (y \oplus z) & \rightarrow & (x \wedge y) \oplus (x \wedge z) \end{array}$$

and the asssociativity and commutativity axioms for $\oplus$ and $\wedge$.

The following polynomial interpretation can be used to prove termination of BR/AC.

$$\begin{array}{rcl} \oplus_\tau(x_1, x_2) & = & x_1 + x_2 + 1 \\ \wedge_\tau(x_1, x_2) & = & x_1 x_2 \\ constant_\tau & = & 2. \end{array}$$

The set of rules BR is then mapped onto the set of polymomials

$$\begin{array}{l} x + 2 + 1 - x \\ x + x + 1 - 2 \\ 2x - 2 \\ 2x - x \\ x^2 - x \\ x(y + z + 1) - (xy + xz + 1) \end{array}$$

Each of these polynomials takes only positive values, provided that variables are always greater than 2. This proves the termination of BR/AC.

Other techniques, for the associativity and commutativity case, are obtained by applying transformations on terms and then comparing their results. For instance, a usual transformation is *flattening* that transforms for instance the term $x + (y + z)$ into $+(x, y, z)$ for an associative commutative symbol $+$. More general transformations can be formalized by application of rewrite rules [BD86a, GL86] to both terms to compare; then the resulting terms are compared using a usual reduction or simplification ordering on terms.

Total AC-compatible orderings

A simplification ordering AC-compatible and total on non-AC-equivalent ground terms was defined by [NR91b]. Another ordering based on a total precedence on function symbols, instead of polynomial interpretations, was proposed in [RN93], which has moreover tha advantage to be extendable to terms with variables.

Let $>_1$ be a recursive path ordering with status based on a total precedence $>_{\mathcal{F}}$, in which AC-symbols have a lexicographical left-to-right status. The interpretation of a term t is defined thanks to a set R_F of flattening rules

$$f(x_1, \ldots, x_m, f(y_1, \ldots, y_r), z_1, \ldots, z_n) \rightarrow f(x_1, \ldots, x_m, y_1, \ldots, y_r, z_1, \ldots, z_n)$$

for all AC-function symbols f with $m + n \geq 1$ and $r \geq 2$, and a set R_I of interpretation rules of the form

$$f(x_1, \ldots, x_m, g(t_1, \ldots, t_r), y_1, \ldots, y_n) \rightarrow f(x_1, \ldots, x_m, t, y_1, \ldots, y_n)$$

for all symbols g and all AC-symbols $f >_{\mathcal{F}} g$ in the precedence, $m + n \geq 1$, where t is the maximal term w.r.t. $>_1$ of $\{t_1, \ldots, t_r\}$ if $r > 0$ and where t is the smallest constant symbol $\perp$ if $r = 0$ and $g \neq \perp$.

The interpretation $I(t)$ of a term t is defined as its normal form w.r.t. $R_F \cup R_I$ under the leftmost-innermost strategy using R_F first.

Using this interpretation, the ordering $>$ is defined as follows.

Definition 7.16 $s > t$ if

- $I(s) >_1 I(t)$ or
- $I(s) =^{mult} I(t)$, i.e. $I(s)$ and $I(t)$ are equal up to a permutation of arguments of AC-operators. Let f be the top symbol of both s and t and $f(s_1, \ldots, s_m), f(t_1, \ldots, t_m)$ be the normal forms of s and t obtained by rewriting with R_F only at topmost position. Then
  1. f is AC and $\{s_1, \ldots, s_m\} >^{mult} \{t_1, \ldots, t_m\}$ or
  2. f is not AC and $\exists i = 1, \ldots, m,\ \forall j < i, s_j =_{AC} t_j$ and $s_i > t_i$.

Example 7.8 Suppose $a >_{\mathcal{F}} b >_{\mathcal{F}} f >_{\mathcal{F}} g >_{\mathcal{F}} h$ where f is AC. We have $f(a, b) > g(a, b)$ because $I(f(a, b)) = f(a, b) >_1 g(a, b) = I(g(a, b))$. We also have $f(a, f(a, b)) > f(a, g(a, b))$ because $I(f(a, f(a, b))) = f(a, a, b) > f(a, a) = I(f(a, g(a, b)))$.

The following results are proved in [RN93]:

Theorem 7.2 *$>$ is AC-compatible and total on non-AC-equivalent ground terms.*

Moreover there exists an extension $>_V$ of $>$ to terms with variables such that $s >_V t$ implies $\sigma(s) > \sigma(t)$ for all ground substitutions σ.

General methods

Designing a general method for proving termination of R/A from termination of R requires good properties to be satisfied by both R and A. The first one is that rules must be compatible with equivalence classes.

Definition 7.17 A reduction ordering $>$ is *compatible* with A if for any terms s, s', t, t', $s' \overset{*}{\longleftrightarrow}_A s > t \overset{*}{\longleftrightarrow}_A t'$ implies $s' > t'$.

Any ordering that is compatible with A induces an ordering on A-congruence classes, also denoted $>$.

Proposition 7.1 A class rewrite system R/A is terminating iff R is contained in some reduction ordering $>$ compatible with A.

Proof: If R/A is terminating, let $>$ be defined by $t > t'$ iff $t \rightarrow_{R/A} t'$. Then this ordering is a reduction ordering compatible with A.

Conversely, assume that R is contained in some reduction ordering $>$ compatible with A. If there exists an infinite derivation sequence

$$t_1 \overset{*}{\longleftrightarrow}_A \rightarrow_R \overset{*}{\longleftrightarrow}_A t_2 \overset{*}{\longleftrightarrow}_A \rightarrow_R \overset{*}{\longleftrightarrow}_A t_3 \overset{*}{\longleftrightarrow}_A \rightarrow_R \overset{*}{\longleftrightarrow}_A \ldots,$$

then the ordering would have an infinite decreasing chain $t_1 > t_2 > t_3 \ldots$. □

Ensuring compatibility is a strong requirement and a weaker condition is proposed in [JM84].

Definition 7.18 A relation $>$ is A-commuting if for any terms s, t, s' such that $s' \overset{*}{\longleftrightarrow}_A s > t$, there exists t' such that $s' > t' \overset{*}{\longleftrightarrow}_A t$.

Proposition 7.2 The class rewrite system R/A is terminating if there exists a A-commuting reduction ordering $>_R$ that contains R.

Proof: If there exists an infinite derivation sequence

$$t_1 \overset{*}{\longleftrightarrow}_A \rightarrow_R \overset{*}{\longleftrightarrow}_A t_2 \overset{*}{\longleftrightarrow}_A \rightarrow_R \overset{*}{\longleftrightarrow}_A t_3 \overset{*}{\longleftrightarrow}_A \rightarrow_R \overset{*}{\longleftrightarrow}_A \ldots,$$

there exists an infinite sequence $t_1 \overset{*}{\longleftrightarrow}_A >_R \overset{*}{\longleftrightarrow}_A t_2 \overset{*}{\longleftrightarrow}_A >_R \overset{*}{\longleftrightarrow}_A t_3 \overset{*}{\longleftrightarrow}_A >_R \overset{*}{\longleftrightarrow}_A \ldots$, and by A-commuting property, $t_1 >_R t_2' >_R t_3' >_R \ldots \overset{*}{\longleftrightarrow}_A \ldots$, obtained by repeatedly pushing to the end the A-equivalence steps. This contradicts the well-foundness of $>_R$. □

An even more general result can be stated, along the same lines:

Proposition 7.3 The class rewrite system R/A is terminating if R is contained in a well-founded monotonic ordering $>$ that commutes over a monotonic equivalence that contains A.

Example 7.9 The example of boolean ring and algebra

A boolean algebra is an algebra $\mathcal{B} = (B, \mathcal{F}_\mathcal{B})$ where $\mathcal{F} = \{0, 1, \wedge, \vee, \neg\}$ that satisfies the following equalities.

$$\begin{aligned}
x \vee 0 &= x \\
x \wedge 1 &= x \\
x \wedge (y \vee z) &= (x \wedge y) \vee (x \wedge z) \\
x \vee (y \wedge z) &= (x \vee y) \wedge (x \vee z) \\
(x \vee y) \wedge y &= y \\
(x \wedge y) \vee y &= y \\
x \vee (\neg x) &= 1 \\
x \wedge (\neg x) &= 0
\end{aligned}$$

$$\begin{aligned}
x \wedge y &= y \wedge x \\
(x \wedge y) \wedge z &= x \wedge (y \wedge z) \\
x \vee y &= y \vee x \\
(x \vee y) \vee z &= x \vee (y \vee z)
\end{aligned}$$

These axioms imply the well-known properties of operators $\wedge, \vee, \neg$:

$$\begin{aligned}
1 \vee x &= 1 \\
x \vee x &= x \\
0 \wedge x &= 0 \\
x \wedge x &= x \\
\neg(x \vee y) &= (\neg x) \wedge (\neg y) \\
\neg(x \wedge y) &= (\neg x) \vee (\neg y) \\
\neg(\neg x) &= x
\end{aligned}$$

Since the 1950s it is known that any term (i.e. boolean formula) of the boolean algebra admits a normal form, which is called *the set of prime implicants* and can be produced algorithmically. Such algorithms have been produced by Quine [Qui52, Qui59] or by Slagle, Chang and Lee [SCL70]. But surprinsingly, attempts to find a Church-Rosser class rewrite system for Boolean algebra using a completion procedure failed to terminate in all experiments reported by [Hul80b, Hul80c, PS81]. Only in 1991, a formal proof on the non-existence of a convergent system for boolean algebra was given in [SA91].

In 1985, J. Hsiang observed that there exists however a convergent class rewrite system in an extended signature $\mathcal{F} = \{0, 1, \wedge, \vee, \neg, \oplus\}$, where $\oplus$ and $\wedge$ are operators from a boolean ring. A class rewrite system for boolean rings, where the conjunction $\wedge$ and the exclusive-or $\oplus$ are associative and commutative, is given by the following sets of rewrite rules and equalities, denoted BR/AC:

$$\begin{array}{rcl}
x \oplus 0 & \rightarrow & x \\
x \oplus x & \rightarrow & 0 \\
x \wedge 0 & \rightarrow & 0 \\
x \wedge 1 & \rightarrow & x \\
x \wedge x & \rightarrow & x \\
\neg x & \rightarrow & x \\
x \wedge (y \oplus z) & \rightarrow & (x \wedge y) \oplus (x \wedge z)
\end{array}$$

$$\begin{array}{rcl}
x \wedge y & = & y \wedge x \\
(x \wedge y) \wedge z & = & x \wedge (y \wedge z) \\
x \oplus y & = & y \oplus x \\
(x \oplus y) \oplus z & = & x \oplus (y \oplus z)
\end{array}$$

Using this class rewrite system, any formula of predicate calculus has a normal form. For instance

$$(p(x) \wedge p(x)) \oplus p(x) \oplus (q(y) \wedge r(x,y) \wedge 1) \oplus (p(x) \wedge 0) \oplus 1 \xrightarrow{*}_{BR/AC} q(y) \wedge r(x,y) \oplus 1.$$

In order to deal with Boolean algebras notations, one may add four more rules for defining implication $\Rightarrow$, equivalence $\equiv$, negation $\neg$ and disjunction $\vee$. A convergent class rewrite system for computing normal forms in Boolean algebras with the extended signature is as follows:

$$\begin{array}{rcl}
x \vee y & \rightarrow & (x \wedge y) \oplus x \oplus y \\
x \Rightarrow y & \rightarrow & (x \wedge y) \oplus x \oplus 1 \\
x \equiv y & \rightarrow & x \oplus y \oplus 1 \\
\neg x & \rightarrow & x \oplus 1 \\
x \oplus 0 & \rightarrow & x \\
x \oplus x & \rightarrow & 0 \\
x \wedge 0 & \rightarrow & 0 \\
x \wedge 1 & \rightarrow & x \\
x \wedge x & \rightarrow & x \\
x \wedge (y \oplus z) & \rightarrow & (x \wedge y) \oplus (x \wedge z)
\end{array}$$

$$\begin{array}{rcl}
x \wedge y & = & y \wedge x \\
(x \wedge y) \wedge z & = & x \wedge (y \wedge z) \\
x \oplus y & = & y \oplus x \\
(x \oplus y) \oplus z & = & x \oplus (y \oplus z)
\end{array}$$

Exercice 39 — The following counter-example and its proof are due to P. Narendran, M. Subrahmaniam and Q. Guo.
It is not the case that if R, A is terminating then R/A is terminating. Prove that the following class rewrite system provides a counter-example which proves this assertion, with A:

$$a + b = a + c$$

and R:

$$c \rightarrow b$$

where a, b and c are constants.
Answer: R/A is not terminating as is illustrated by the following loop:

$$a + b \longleftrightarrow_A a + c \rightarrow_R a + b.$$

Note that by definition of R, A, for any term t there exists no position ω in t such that $t_{|\omega} = \sigma(c)$, for any substitution σ, unless $t_{|\omega} = c$. Hence $\rightarrow_{R,A}$ is equivalent to $\rightarrow_R$. Since $\rightarrow_R$ is trivially terminating hence $\rightarrow_{R,A}$ is terminating.

7.4 Ordered class rewriting

Especially for the case of AC-theories, it is possible to combine both notions of ordered and class rewriting. As in ordered rewrite systems, the reduction ordering is made explicit, and as in class rewrite systems, it has to be compatible with the equivalence class.

Definition 7.19 An *ordered class rewrite system*, denoted $(E/A, >)$, is defined by a set of axioms A, a set of equalities E and a well-founded A-compatible reduction ordering $>$ total on A-equivalence classes of ground terms. A and E are assumed disjoint.

The class rewrite relation applies to a term if there exists a term in the same equivalence class modulo A that is reducible with $E^>$.

Definition 7.20 Given an ordered class rewrite system $(E/A, >)$, the term t $(E/A, >)$-*rewrites* to t', denoted $t \rightarrow_{E/A,>} t'$, if $t \overset{*}{\longleftrightarrow}_A u[\sigma(l)]_\omega$ and $t' \overset{*}{\longleftrightarrow}_A u[\sigma(r)]_\omega$, for some equation $(l = r) \in E$, some term u, some occurrence ω in u and some substitution σ such that $\sigma(l) > \sigma(r)$.

By construction $\rightarrow_{E/A,>}$ is terminating. A term irreducible for $\rightarrow_{E/A,>}$ is said in E/A-normal form. w.r.t. $>$. The $(E/A, >)$-normal form of a term t is denoted $t\downarrow_{E/A,>}$.

A-compatible reduction orderings do not exist when E is non-empty and A contains an axiom like idempotency $(x + x = x)$ where a lone variable occurs on one side and several times on the other. From an instance σ of an equality $(l = r) \in E$, a contradiction to well-foundedness of $>$ may be built, provided $\sigma(l) > \sigma(r)$:
$\sigma(l) \overset{*}{\longleftrightarrow}_A \sigma(l) + \sigma(l) > \sigma(r) + \sigma(l) \overset{*}{\longleftrightarrow}_A \sigma(r) + (\sigma(l) + \sigma(l)) > \sigma(r) + (\sigma(r) + \sigma(l)) \dots$
Other axioms that prevent the existence of an A-compatible reduction ordering are equalities like $(x * 0 = 0)$ where a variable occurs on one side and not on the other. Then $0 \longleftrightarrow_A \sigma(l) * 0 > \sigma(r) * 0 \longleftrightarrow_A 0$ provides the contradiction. Indeed, if such axioms are present, they must be considered as ordered equalities.

The rewrite relation $\rightarrow_{E/A,>}$ is not completely satisfactory from an operational point of view: even if E is finite and $\overset{*}{\longleftrightarrow}_A$ decidable, $\rightarrow_{E/A,>}$ may not be computable since equivalence classes modulo A may be infinite or not computable. For instance, the axiom $(-x = x)$ generates infinite equivalence classes. To avoid scrutiny through equivalence classes, the idea is to use a weaker relation on terms, called *ordered rewriting modulo A*, which incorporates A in the matching process, and uses the set $E^>$ of ordered instances of E.

A more operational definition of ordered rewriting modulo A is given below:

Definition 7.21 Given an ordered class rewrite system $(E/A, >)$, a term t $(E, A, >)$-*rewrites* to a term t', which is denoted by $t \rightarrow_{E,A,>} t'$ if there exists

- an equality $l = r$ of E,
- a position ω in t,
- a substitution σ, satisfying $t_{|\omega} \overset{*}{\longleftrightarrow}_A \sigma(l)$ and $\sigma(l) > \sigma(r)$

such that $t' = t[\omega \hookleftarrow \sigma(r)]$.

Note that $s \rightarrow_{E,A,>} t$ implies $s \overset{*}{\longleftrightarrow}_{E\cup A} t$ and $s > t$.

The corresponding Church-Rosser property crucially depends on the reduction ordering.

Definition 7.22 The class rewrite system $(E/A, >)$ is *Church-Rosser* on a set of terms $\mathcal{T}$ if

$$\overset{*}{\longleftrightarrow}_{E\cup A} \subseteq \overset{*}{\longrightarrow}_{E/A,>} \circ \overset{*}{\longleftrightarrow}_A \circ \overset{*}{\longleftarrow}_{E/A,>} .$$

The ordered rewriting relation $(E, A, >)$ defined on $\mathcal{T}$ is:

Church-Rosser modulo A	if	$\overset{*}{\longleftrightarrow}_{E\cup A} \subseteq \overset{*}{\longrightarrow}_{E,A,>} \circ \overset{*}{\longleftrightarrow}_A \circ \overset{*}{\longleftarrow}_{E,A,>}$
confluent modulo A	if	$\overset{*}{\longleftarrow}_{E,A,>} \circ \overset{*}{\longrightarrow}_{E,A,>} \subseteq \overset{*}{\longrightarrow}_{E,A,>} \circ \overset{*}{\longleftrightarrow}_A \circ \overset{*}{\longleftarrow}_{E,A,>}$
coherent modulo A	if	$\overset{*}{\longleftarrow}_{E,A,>} \circ \overset{*}{\longleftrightarrow}_A \subseteq \overset{*}{\longrightarrow}_{E,A,>} \circ \overset{*}{\longleftrightarrow}_A \circ \overset{*}{\longleftarrow}_{E,A,>}$
locally confluent modulo A	if	$\longleftarrow_{E,A,>} \circ \rightarrow_{E,>} \subseteq \overset{*}{\longrightarrow}_{E,A,>} \circ \overset{*}{\longleftrightarrow}_A \circ \overset{*}{\longleftarrow}_{E,A,>}$
locally coherent modulo A	if	$\longleftarrow_{E,A,>} \circ \longleftrightarrow_A \subseteq \overset{*}{\longrightarrow}_{E,A,>} \circ \overset{*}{\longleftrightarrow}_A \circ \overset{*}{\longleftarrow}_{E,A,>}$

The next theorem, adapted from [JK86c], relates the different properties.

Theorem 7.3 *The following properties of an ordered class rewrite system $(E/A, >)$, are equivalent on $\mathcal{T}$:*
- $(E, A, >)$ *is Church-Rosser modulo A.*
- $(E, A, >)$ *is confluent modulo A and coherent modulo A.*
- $(E, A, >)$ *is locally confluent and locally coherent modulo A.*
- $\forall t, t'$, $t \overset{*}{\longleftrightarrow}_{E\cup A} t'$ *iff* $t\downarrow_{E,A,>} \overset{*}{\longleftrightarrow}_A t'\downarrow_{E,A,>}$.

7.5 Conditional rewriting

Conditional rewrite systems arise naturally in algebraic specifications of data types, where they provide a way to handle partial operations and case analysis. They are also useful in theorem proving where they offer an alternative to Horn clauses.

The algebraic study of conditional rewrite rules, that are rules of the form $l \to r$ if $(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$, has been developed in many directions.

- Earlier approaches, as in [BK86, Kap84], consider such systems without any restrictive hypothesis on the rules. In *recursive rewriting* [Kap84, Kap87] the condition of a rule is evaluated first and the rule is applied only when the condition holds. Thus conditional rules introduce the additional complexity of recursively evaluating the conditions, which gives rise to a new termination problem.
- In *contextual rewriting* [Rém82] rules may be applied to terms without prior evaluation of the condition, which is appended to the environment as a context. This approach also is subject to an obvious non-termination risk; in general, the contextual part of a term may grow unbounded during a sequence of reductions.

With a general definition of conditional rules, simple questions, as "does a term t rewrite to t'" become clearly undecidable [Kap84]. Researches then focus on finding restrictions on the rules to yield a decidable rewriting relation.

- In order to avoid infinite recursive evaluation of the condition, the *hierarchical* approach [Dro83, PEE81, Rém82, ZR85] states a hierarchy on the rules and asks that, for each rule, the condition is evaluated at a lower level of the hierarchy.
- Termination of evaluation may be obtained via a reduction ordering to compare terms involved in the equality and in the condition of a conditional rewrite rule. In a *simplifying* system, every left-hand side of a rule is greater than both its right-hand side and its condition in some simplification ordering [Kap87]. Since such an ordering is well-founded, every term can be brought to a normal form via a simplifying system. Simplifying systems have been generalized to *reductive* systems [JW86], by replacing a simplification ordering with a more general reduction ordering.
- An even more general proposal is *decreasing* conditional rewrite systems [DOS87], that are systems where for each instance, the instantiated left-hand side is greater than the instantiated right-hand side and conditions. They have been extended in [BG89, DO90] to cover systems with variables in the conditions that do not appear in the left-hand side. The rewrite relation for decreasing systems is terminating and decidable, when the rewrite system consists has a finite number of rules.
- Considering a conditional equality as an equational Horn clause leads to use a general ordering on clauses to define *ordered* conditional rewriting. An ordered conditional equality is such that the equality is maximal w.r.t. conditions and the right-hand side is not greater than the left-hand side. This last approach is an instance of [BG91b].

7.5.1 Conditional rewrite systems

The formulas being considered are *conditional equalities*, written

$$l = r \text{ if } \Gamma$$

, where Γ is a conjunction of equalities. $\Gamma = (s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ and $l = r$ are respectively called the *condition* and the *conclusion* of the conditional equality (See Section 2.6 of Chapter 2).

In a *conditional rewrite rule*, the equality is oriented and denoted as

$$t \to s \text{ if } \Gamma.$$

Definition 7.23 A *conditional rewrite system* is a set of *conditional rewrite rules* of the form

$$l \to r \text{ if } (s_1 = t_1 \wedge \cdots \wedge s_n = t_n).$$

$\Gamma = (s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$, l and r are respectively called the *condition*, left-hand side and right-hand side of the conditional rewrite rule.

Using a conditional rewrite system R, a given rule could be applied to a term if its condition, instantiated by the matching substitution, is satisfied.

In standard rewrite systems, every variable occurring in a right-hand side of a rule also occurs in the left-hand side. This condition generalizes to conditional rewrite rules by requiring that every variable occurring either in the condition or in the right-hand side of a rule also occur in the left-hand side. This may appear as a strong restriction, and it is possible to define conditional rewriting in a very general way, as follows:

Definition 7.24 Given a conditional rewrite system R, a term t *rewrites* to a term t', which is denoted by $t \rightarrow_R t'$ if there exist:

1. a conditional rewrite rule $l \rightarrow r \text{ if } \Gamma$ of R,
2. a position ω in t,
3. a substitution σ, satisfying $t_{|\omega} = \sigma(l)$,
4. a substitution τ for the new variables such that $\tau(\sigma(\Gamma))$ holds,

and then $t' = t[\omega \hookleftarrow \tau(\sigma(r))]$.

With this definition, a conditional rewrite rule is applicable if there exists a substitution for the extra variables that makes the condition hold. Operationally this problem can be solved by narrowing. However, with extra variables, ground confluence no longer guarantees the completeness of the narrowing process [GM87]. Completeness can be restored at the price of a stronger confluence condition, called level-confluence. The interested reader can refer to [DO88, DO90] for more results on this subject.

Let us exemplify the use of extra variables on the definition of the *append* function of two lists:

Example 7.10 Consider the set of conditional rewrite rules:

$$\begin{array}{lcll} append(nil, y) & \rightarrow & y & \\ append(x, y) & \rightarrow & cons(u, z) & \text{if} \quad (x = cons(u, v) \wedge append(v, y) = z) \end{array}$$

We have

$$append(cons(a, nil), nil) \rightarrow cons(a, nil)$$

since $\{u \mapsto a, v \mapsto nil, z \mapsto nil\}$ is a solution modulo this definition of append of $(cons(a, nil) =^? cons(u, v) \wedge append(v, nil) =^? z)$.

Definition 7.24 is too general to be of practical use. This leads to distinguish more restrictive notions of conditional rewriting relations, according to the kind of evaluation chosen for the conditions. Definition 7.24 is then modified accordingly. First it is often required that every variable occurring either in the condition or in the right-hand side of a rule also occurs in the left-hand side.

$$\mathcal{V}ar(\Gamma) \cup \mathcal{V}ar(r) \subseteq \mathcal{V}ar(l).$$

Thus in the condition "there exists a substitution τ for the new variables such that $\tau(\sigma(\Gamma))$ holds," of Definition 7.24, τ becomes the identity and we are left to check that $\sigma(\Gamma)$ holds. Second, the evaluation of the conditions could be made more operational in different ways. For the different kinds of rewriting relations introduced below, we mention only the modified parts of Definition 7.24:

1. for *natural conditional rewriting*:
   $t \rightarrow_{R^{nat}} t'$ if
   (4) there exists a proof $\sigma(s_i) \overset{*}{\longleftrightarrow}_{R^{nat}} \sigma(t_i)$ for each instantiated component of the condition.
2. for *join conditional rewriting*:
   $t \rightarrow_{R^{join}} t'$ if
   (4) there exists a joinability proof $\sigma(s_i) \downarrow_{R^{join}} \sigma(t_i)$ for each instantiated component of the condition.
3. for *normal conditional rewriting*:
   $t \rightarrow_{R^{norm}} t'$ if
   (4) $\sigma(t_i)$ is a normal form of $\sigma(s_i)$, denoted by $\sigma(s_i) \xrightarrow{!}_{R^{norm}} \sigma(t_i)$, for each instantiated component of the condition.

For a better readability, the relations $\rightarrow_{R^{nat}}$, $\rightarrow_{R^{join}}$ and $\rightarrow_{R^{norm}}$ are denoted simply by R^{nat}, R^{join} and R^{norm}.

Example 7.11 [BK86] Consider the join conditional rewriting relation for the system:

$$\begin{array}{lll} f(a) & \rightarrow & a \\ f(x) & \rightarrow & c \quad \mathsf{if} \quad x = f(x) \end{array}$$

Since $a \downarrow f(a)$ we have $f(a) \rightarrow c$ by applying the second rule. Because $f(a) \downarrow_{R^{join}} f(f(a))$, we have also $f(f(a)) \rightarrow c$. But neither c nor $f(c)$ are reducible.

Example 7.12 Consider the normal conditional rewriting relation for the system:

$$\begin{array}{lllll} even(0) & \rightarrow & true & & \\ even(s(x)) & \rightarrow & odd(x) & & \\ odd(x) & \rightarrow & true & \mathsf{if} & even(x) = false \\ odd(x) & \rightarrow & false & \mathsf{if} & even(x) = true \end{array}$$

Then $even(s(0)) \rightarrow odd(0) \rightarrow false$ by applying first the second then the fourth rule.

In any case, the rewrite relation $\rightarrow_R$ associated to a conditional rewrite system has an inductive definition, which is fundamental for establishing properties of conditional rewrite systems.

Definition 7.25 Let R be a conditional rewrite system and R_i the rewrite system defined for $i \geq 0$ as follows:

$$\begin{array}{lll} R_0 & = & \{l \rightarrow r \mid l \rightarrow r \in R\} \\ R_{i+1} & = & R_i \cup \{\sigma(l) \rightarrow \sigma(r) \mid l \rightarrow r \text{ if } (s_1 = t_1 \wedge \cdots \wedge s_n = t_n) \in R, \\ & & \text{ and } \forall j = 1, \ldots, n, \; \sigma(s_j) \equiv_i \sigma(t_j)\} \end{array}$$

where $\equiv_i$ denotes $\overset{*}{\longleftrightarrow}_{R_i}$, $\downarrow_{R_i}$ or $\overset{!}{\longrightarrow}_{R_i}$.

Then:

Lemma 7.2 $t \rightarrow_R t'$ iff $t \rightarrow_{R_i} t'$ for some $i \geq 0$.

With this definition, an abstract reduction system can be associated to any conditional rewrite system, according to the evaluation chosen for the conditions. All definitions and properties of Chapter 4 are thus available for these abstract reduction systems.

Let us now compare the different definitions of conditional rewriting in more details. The main problem with natural conditional rewriting is that the applicability of a rule involves arbitrary proofs of equalities and so little is gained from this notion of rewriting. This is the reason for considering the more restrictive definition of join conditional rewriting. The notion of normal rewriting appears as even more restrictive. However the following results show that it has the same power as join rewriting.

First any join rewriting relation can be simulated by a normal rewriting relation on an enriched set of terms. Let R^{join} be generated by the set of conditional rewrite rules $R = \{l \rightarrow r \text{ if } (s_1 = t_1 \wedge \cdots \wedge s_n = t_n)\}$, expressed with function symbols $\mathcal{F}$; let R^{norm}_{ext} be the normal rewriting relation generated by the system R_{ext} obtained by replacing each rule $l \rightarrow r$ if $(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ by a rule of the form $l \rightarrow r$ if $(eq(s_1, t_1) = true \wedge \cdots \wedge eq(s_n, t_n) = true)$ where eq is a binary function symbol and $true$ is a constant, that do not belong to $\mathcal{F}$. R_{ext} additionally contains the rule $eq(x, x) \rightarrow true$.

In order to more precisely compare the deduction power of these different rewriting relations, the additional hypothesis of *decreasingness* is needed.

Definition 7.26 A conditional rewriting relation is *decreasing* if there exists a well-founded extension $>$ of the rewriting relation $\rightarrow$ which satisfies two additional properties:

- $>$ contains the proper subterm relation $\rhd$,
- for each rule $l \rightarrow r$ if $(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$, we have $\sigma(l) > \sigma(s_i)$ and $\sigma(l) > \sigma(t_i)$, for all substitutions σ and all indices i, $1 \leq i \leq n$.

Any conditional rewrite system has an underlying conditional equational system of the form defined in Definition 7.23 denoted by R^{eqn}. Indeed R^{eqn} is obtained by replacing $\rightarrow$ by $=$ in R. So $R^{eqn} \vdash p = q$ means that the equality $(p = q)$ can be deduced from the conditional system R^{eqn} by the rules of equational conditional deduction.

The next theorem summarizes the results presented in [DO90].

Theorem 7.4 *Let p and q be any terms, R be any conditional rewrite system R and R^{eqn} be its underlying conditional equational system. Then:*

- $$p \overset{*}{\longleftrightarrow}_{R^{nat}} q \text{ iff } R^{eqn} \vdash p = q.$$
- *If R^{join} is confluent,*
$$p \downarrow_{R^{join}} q \text{ iff } R^{eqn} \vdash p = q.$$
- *If R^{nat} is decreasing and confluent,*
$$p \downarrow_{R^{nat}} q \text{ iff } p \downarrow_{R^{join}} q.$$
- *If R_{ext} is the associated extended conditional rewrite system*
$$p \downarrow_{R^{join}} q \text{ implies } p \downarrow_{R_{ext}} q.$$
Conversely, if p and q do not contain eq and true,
$$p \downarrow_{R_{ext}} q \text{ implies } p \downarrow_{R^{join}} q.$$

Obviously, if R^{join} is confluent, then so is the relation R^{nat}, but the converse does not hold, because not all proofs in the condition of a natural rewriting proof can be transformed into joinability proofs.

Example 7.13 [Kap84] Consider the following system, where a, b, c, a', b' are constants and g, d are any terms.

$$\begin{array}{lcll} c & \rightarrow & a & \\ a' & \rightarrow & b & \\ b' & \rightarrow & a & \\ b' & \rightarrow & b & \\ g & \rightarrow & d & \mathsf{if} \quad c = a' \end{array}$$

Then $c \longleftrightarrow_{R^{nat}} a \longleftrightarrow_{R^{nat}} b' \longleftrightarrow_{R^{nat}} b \longleftrightarrow_{R^{nat}} a'$ and so $g \rightarrow_{R^{nat}} d$. But $c \downarrow_{R^{join}} a'$ is false and thus $g \rightarrow_{R^{join}} d$ is false, as well as $g \overset{*}{\longleftrightarrow}_{R^{join}} d$.

7.5.2 Decidability results

For terminating unconditional rewrite systems, with a finite number of rules, the joinability relation ($p \downarrow q$) is decidable. With conditional rewrite systems, joinability is not necessarily decidable, even for finite terminating conditional systems. In [Kap83, Kap84], an example is built of a conditional rewrite system such that the relation $\rightarrow_R$ is not decidable and such that the normal form of a term is not computable. Additional conditions must be added to get back decidable properties.

Theorem 7.5 *[DO90] If $\rightarrow_{R^{join}}$ is decreasing, the relations $\rightarrow_{R^{join}}$, $\overset{*}{\longrightarrow}_{R^{join}}$ and $\downarrow_{R^{join}}$ are all decidable.*

Proof: This is proved by induction with respect to the ordering that makes the conditional rewriting relation decreasing. □

The notion of decreasingness of a rewriting relation is not usable in practice. A property that can be checked on the conditional rewrite system is preferable. This leads to introduce the notion of reductive system [JW86] that generalizes the definition of simplifying system in [Kap87].

Definition 7.27 [JW86] A conditional rewrite system is *reductive* if there exists a well-founded reduction ordering $>$ such that for each rule $l \rightarrow r$ if $(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$, and for all substitutions σ:

(i) $\sigma(l) > \sigma(r)$ and

(ii) $\sigma(l) > \sigma(s_i)$ and $\sigma(l) > \sigma(t_i)$, for all indices i, $1 \leq i \leq n$.

Actually reductive systems capture the finiteness of evaluation of terms, as explained in [DO90] or in [Kap87]. Let the following predicate $\rightsquigarrow$ be defined on terms by $t \rightsquigarrow t'$ if there exist a conditional rewrite rule $l \to r$ if $(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ of R, a position ω in t, a substitution σ, satisfying $t_{|\omega} = \sigma(l)$, and such that $t' = \sigma(s_i)$ or $t' = \sigma(t_i)$ for some i, $1 \leq i \leq n$. Intuitively this means that evaluating t, while trying to apply the given rule, leads to evaluate t'. Ensuring termination of the evaluation procedure turns out to guarantee the termination of $\to_R \cup \rightsquigarrow$. This is done by imposing that the right-hand side and the condition of any rule are simpler than its left-hand side. The relation $\to_R \cup \rightsquigarrow$ corresponds to one step computation and its transitive closure $(\to_R \cup \rightsquigarrow)^+$ represents an arbitrary computation branch.

Proposition 7.4 Let R be a reductive conditional rewrite system. Then R^{join} is decreasing.

Proof: Let us define $>$ as the union $\to_{R^{join}} \cup \rightsquigarrow \cup \rhd$. Clearly $>$ is an extension of the rewriting relation $\to_{R^{join}}$ which satisfies the two additional properties:

- $>$ contains the proper subterm relation $\rhd$
- for each rule $l \to r$ if $(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$, $\sigma(l) \rightsquigarrow \sigma(s_i)$ and $\sigma(l) \rightsquigarrow \sigma(t_i)$, for all substitutions σ and all indices i, $1 \leq i \leq n$.

It remains to prove that $>$ is well-founded. This follows from the fact that the proper subterm relation is well-founded and commutes with both $\to_{R^{join}}$ and $\rightsquigarrow$. So the termination of $>$ amounts to the termination of $\to_{R^{join}} \cup \rightsquigarrow$. But since R is reductive, $t \to_{R^{join}} t'$ implies $t > t'$ and $t \rightsquigarrow t'$ implies $t > t'$. Since $>$ is well-founded, so is $\to_{R^{join}} \cup \rightsquigarrow$. $\Box$

7.5.3 Ordered conditional systems

As for unconditional equalities, it is possible to define a notion of ordered conditional rewriting, to deal with conditional systems where only some instances of the rules are reductive.

Definition 7.28 An *ordered conditional rewrite system* denoted $(C, >)$, is a set of conditional equalities C together with a reduction ordering $>$.

Let $>$ be a reduction ordering on terms which is assumed to be contained in some ordering total on ground terms. Remind that all recursive path orderings satisfy this condition. Let $\geq$ be defined by $s \geq t$ if $s > t$ or $s = t$.

From the conditional equalities and the ordering, a set of ordered instances has to be built. But we first need to have an appropriate (and most powerful) notion of ordered conditional equality. This requires the definition of an ordering on the different components of a conditional equality.

The multiset extension $>^{mult}$ is an ordering on equalities denoted by $>_{\mathcal{E}}$.

This ordering extends to conditional equalities considered as multisets of equalities as follows: First add $\perp$ as a new symbol that satisfies for every term t, $t > \perp$. Then associate to each occurrence of the equality $s = t$, its complexity $c(s = t)$ which is
- the multiset $\{\{s\}, \{t\}\}$ if $s = t$ is the conclusion of C,
- the multiset $\{\{s, \perp\}, \{t, \perp\}\}$ if $s = t$ occurs in the condition of C, These complexities are multisets of multisets of terms that may be compared using $(>^{mult})^{mult}$.

The complexity of a conditional equality C, denoted by $c(C)$ is the multiset of complexities $c(s = t)$ for any $s = t \in C$. In other words, each conditional equality

$$C = (l = r \text{ if } \bigwedge_{i=1,\ldots,n} s_i = t_i)$$

has the complexity

$$c(C) = \{\{\{s_1, \perp\}, \{t_1, \perp\}\}, ..., \{\{s_n, \perp\}, \{t_n, \perp\}\}, \{\{l\}, \{r\}\}\}.$$

This ordering gives rise to the notion of *ordered* conditional equality that generalizes the concept of ordered equality.

Definition 7.29 A conditional equality $c = (l = r \text{ if } \Gamma)$ is *ordered* if $r \not\geq l$ and $c(l = r)$ is strictly maximal in $c(C)$ (i.e. there is no other element in the multiset $c(C)$ greater or equal to $c(l = r)$)

We now get the notion of ordered instances.

Definition 7.30 Given an ordered conditional rewrite system $(E, >)$, an *ordered instance* of a conditional equality $(l \rightarrow r \text{ if } \Gamma) \in E$ (with respect to the ordering $>$), is an ordered conditional equality $(\sigma(l) \rightarrow \sigma(r) \text{ if } \sigma(\Gamma)$.

Thus a conditional equality can be used to rewrite a term only if its instance by the matching substitution is ordered.

The *ordered conditional rewriting* relation is just rewriting with ordered instances of conditional equalities.

The relation between ordered conditional equalities and reductive conditional rewrite rules is clarified in the next proposition.

Proposition 7.5 Assume that $>$ can be extended into a simplification ordering total on ground terms. If a conditional equality is ordered, then its ground instances define a decreasing conditional rewrite relation on ground terms. Conversely any reductive conditional rewrite rule is an ordered conditional equality.

Proof: Let $C = (l = r \text{ if } \bigwedge_{i=1,\ldots,n} s_i = t_i)$ be an ordered conditional equality. Let $\rightarrow$ the join conditional rewriting relation defined by the set of ground instances $(\sigma(l) = \sigma(r) \text{ if } \bigwedge_{i=1,\ldots,n} \sigma(s_i) = \sigma(t_i))$.

Let us also call $>$ the extension of $>$ on ground terms. If $>$ is a simplification ordering on ground terms, $>$ contains the proper subterm relation. For any ground substitution σ, $\sigma(l) > \sigma(r)$. So $\rightarrow \subseteq >$ on ground terms. Moreover saying that $c(l = r)$ is strictly maximal in $c(C) = \{\{\{s_1, \perp\}, \{t_1, \perp\}\}, ..., \{\{s_n, \perp\}, \{t_n, \perp\}\}, \{\{l\}, \{r\}\}\}$, implies that for all $i = 1, \ldots, n$, $\sigma(l) > \sigma(s_i)$ and $\sigma(l) > \sigma(t_i)$.

Conversely, if $l \rightarrow r \text{ if } \bigwedge_{i=1,\ldots,n} s_i = t_i$ is a reductive conditional rewrite rule w.r.t. a reduction ordering $>$, then the conditional equality $C = (l = r \text{ if } \bigwedge_{i=1,\ldots,n} s_i = t_i)$ satisfies for each substitution σ, $\sigma(l) > \sigma(r)$, and $\forall i = 1, \ldots n, \sigma(l) > \sigma(s_i), \sigma(l) > \sigma(t_i)$, so in particular, $\forall i = 1, \ldots n, l > s_i, l > t_i$. This implies that $l = r$ is maximal. $\square$

7.5.4 Horn clauses versus conditional rewrite rules

Programming with conditional equalities has the great advantage to combine functional and logic programming paradigms in a uniform framework. The relation between rewrite programs and Horn clause programs from a semantical point of view is clarified in [BH92]. Here we borrow an example of [DO90] in order to compare the two formalisms of Horn clauses and conditional rewrite rules on a simple example. In a Horn clauses language, the definition of the function *append* is given by the two clauses:

$$\begin{array}{lcl} null(x) & \Rightarrow & append(x, y) = y \\ \neg null(x) & \Rightarrow & append(x, y) = cons(car(x), append(cdr(x), y)) \end{array}$$

The additional definitions of *null*, *car* and *cdr* are also given as clauses:

$$\begin{array}{l} \Rightarrow null(nil) \\ \Rightarrow \neg null(cons(x, y)) \\ \Rightarrow car(cons(x, y)) = x \\ \Rightarrow cdr(cons(x, y)) = y \end{array}$$

In order to incorporate equality in the deduction mechanism for first-order logic, paramodulation must be added to resolution for the need of completeness. But then non-linear forward reasoning and backward chaining are required in this approach.

An alternative is to use function definitions as rewrite rules. For the *append* definition, the following set of rules could be written:

$$\begin{array}{lcl} append(x, y) & \rightarrow & if\ null(x)\ then\ y\ else\ cons(car(x), append(cdr(x), y)) \\ if\ true\ then\ x\ else\ y & \rightarrow & x \\ if\ false\ then\ x\ else\ y & \rightarrow & y \\ null(nil) & \rightarrow & true \\ null(cons(x, y)) & \rightarrow & false \\ car(cons(x, y)) & \rightarrow & x \\ cdr(cons(x, y)) & \rightarrow & y. \end{array}$$

It is important to note that unrestricted reduction using this system could lead to the following infinite derivation:

$$append(nil, nil) \rightarrow if\ null(nil)\ then\ nil\ else\ cons(car(nil), append(cdr(nil), nil))$$

$$\begin{aligned}
&\rightarrow\ if\ true\ then\ nil\ else\ cons(car(nil), append(cdr(nil), nil))\\
&\rightarrow\ if\ true\ then\ nil\ else\ cons(car(nil), if\ null(cdr(nil))\ then\ nil\\
&\qquad else\ cons(car(cdr(nil)), append(cdr(cdr(nil)), nil))).
\end{aligned}$$

In this specific case, instead of axiomatizing the *if then else* operator, the condition can be better expressed by mutually exclusive left-hand side patterns; then the three first rules are replaced by the next ones:

$$\begin{array}{lcl}
append(nil, y) & \rightarrow & y\\
append(cons(x, y), z) & \rightarrow & cons(x, append(y, z)).
\end{array}$$

But this is not always possible. A more general method is to transform a rewrite rule

$$f(x, y) \rightarrow if\ p(x, y)\ then\ r(x, y)\ else\ s(x, y)$$

into two conditional rewrite rules:

$$\begin{array}{lclll}
f(x, y) & \rightarrow & r(x, y) & \mathsf{if} & p(x, y) = true\\
f(x, y) & \rightarrow & s(x, y) & \mathsf{if} & p(x, y) = false
\end{array}$$

Applied to *append* this gives:

$$\begin{array}{lclll}
append(x, y) & \rightarrow & y & \mathsf{if} & null(x) = true\\
append(x, y) & \rightarrow & cons(car(x), append(cdr(x), y)) & \mathsf{if} & null(x) = false.
\end{array}$$

We get by this way a form very similar to the Horn clauses definition.

The transformations we have used above show the difficulties to reduce conditional to unconditional rewriting while preserving properties like termination. This is the subject of many studies well summarized in [Hin94].

7.6 Constrained rewriting

Following the tradition of logic programming [JL87] and higher-order unification [Hue72], constraints have been introduced in automated theorem proving to improve inference systems and deduction in several aspects. Some advantages in using deduction with constraints are

1. to make explicit every symbolic computation step, especially unification, orientation and typing.
2. to modularize deduction and in particular to design better controls by delaying complex problem solving.
3. to schematize (infinitely) many objects.
4. to get more expressive power.

These advantages may be better illustrated by a few examples where the concept of constraints is of interest.

Example 7.14 In these examples, constraints are equations ($t =^? t'$), disequations ($t \neq^? t'$), inequations: ($t >^? t'$) or membership constraints ($t \in^? s_0$) where s_0 is a sort. In order to emphasize the fact that the predicates used in constraints are interpreted in a specific way and that we are looking for solutions, we put a question mark as exponent in constraints. Indexes specify instead in which interpretation constraints are solved. For instance ($t =^?_\emptyset t'$) is an equation to solve in the empty theory, while ($t =^?_A t'$) is solved in the theory A.

We can then express

- A rule applying everywhere except in one point:

$$(-(x + y) \rightarrow (-x) + (-y) \parallel (x \neq^?_{ACIdentity} 0 \wedge y \neq^?_{ACIdentity} 0))$$

- Ordered rewriting:

$$(x + y \rightarrow y + x\ \parallel\ x >^?_\emptyset y)$$

- Rewriting modulo AC "à la Pedersen" where AC-equalities only occur below a variable position of the matched rewrite rule:
$$(x + y \to y \parallel x =^?_{AC} 0)$$

- A relation true on even natural numbers:
$$(P(x) \;\parallel\; x =^?_A 0)$$
with $A = \{s(s(x)) = x\}$ schematizes
$$P(0) \wedge P(s(s(0)) \wedge ... \wedge P(s^{2n}(0)) \wedge ....$$

- A meta-rule:
$$(f(g(\dot{x})) \to g(\dot{x}) \parallel \dot{x} \in^? g^n(f(x)))$$
that represents the infinite family $\{f(g^n(f(x))) \to g^n(f(x))\}$, generated by the divergent completion of one-rule system $f(g(f(x))) \to g(f(x))$.

- Some structure sharing:
$$(f(x, x, x, x) \;\parallel\; x =^?_\emptyset \mathit{bigterm})$$

- Infinite terms:
$$(h(x) \to x \;\parallel\; x =^?_\emptyset f(x))$$

- Order-sorted rewriting:
$$(f(x) \to a \parallel x \in^? s_0)$$
where $\mathcal{S} = \{s_0, s_1\}$, with $s_0 \leq s_1$, $\mathcal{F} = \{a, f\}$,
$$\begin{array}{rccc} a: & & \to & s_0 \\ f: & s_0 & \to & s_1 \end{array}$$

7.6.1 Constraints

A constraint is a first-order formula built on a first-order signature $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P})$ where $\mathcal{S}$ is a set of sort symbols, $\mathcal{F}$ a set of function symbols and $\mathcal{P}$ a set of predicate symbols. This signature is used to build elementary constraints. For instance, considering $\mathcal{P} = \{=, \neq, >, \in\}$, elementary constraints are equations: $(t =^?_\emptyset t')$ or $(t =^?_A t')$, disequations: $(t \neq^?_\emptyset t')$ or $(t \neq^?_A t')$, inequations: $(t >^?_\emptyset t')$ or $(t >^?_A t')$, or membership relations: $(t \in^? s_0)$.

Elementary constraints are then combined with usual first-order connectives and quantifiers. So the constraint language may also contain, beyond elementary constraints, conjunctions $c \wedge c'$, the empty conjunction $\mathbf{T}$ (always trivially true), negations $\neg c$, existential closures $(\exists x,\ c)$ where c, c' are constraints. Disjunctions $(c \vee c')$, falsity $\mathbf{F}$ (trivially unsatisfiable), universal closures $(\forall x,\ c)$, and implications $(c \Rightarrow c')$ can also be defined from the previous ones.

The definition of constraint languages adopted in this paper is an instance of the definition given in [Smo89, KKR90]. The main difference is that we restrict here to one interpretation, instead of considering a class of interpretations.

Let us briefly remind that, given a signature Σ, a first-order Σ-structure $\mathcal{K}$ is given by :
- a carrier K which is a collection of non-empty sets $(K_s)_{s \in \mathcal{S}}$,
- for each function symbol in $\mathcal{F}$ with a rank $f : s_1, \ldots, s_n \mapsto s$, a function $f_\mathcal{K}$ from $K_{s_1} \times \ldots \times K_{s_n}$ to K_s,
- for each predicate symbol except $=$ in $\mathcal{P}$ with a rank $p : s_1, \ldots, s_n$, a relation $p_\mathcal{K}$ on $K_{s_1} \times \ldots \times K_{s_n}$.
Whenever Σ contains the predicate symbol $=$, it will be interpreted as the equality relation in $\mathcal{K}$.

An assignment α is a mapping from $\mathcal{X}$ to K, that uniquely extends to an homomorphism $\underline{\alpha}$ from $\mathcal{T}(\Sigma, \mathcal{X})$ to $\mathcal{K}$. The restriction of an assignment α to a set of variables $V \subseteq \mathcal{X}$ is denoted by $\alpha_{|V}$. The set of all assignments is denoted by $ASS^{\mathcal{X}}_K$.

Definition 7.31 Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P})$ be a signature and $\mathcal{X}$ a set of variables. A *constraint language* $L_\mathcal{K}[\Sigma, \mathcal{X}]$ (or $L_\mathcal{K}$ for short) is given by :

- a set of *constraints* which are first-order formulae built over Σ and variables $\mathcal{X}$. The empty conjunction $\mathbf{T}$ is also a constraint. Constraints are syntactically distinguished by a question mark exponent on predicates. $\mathcal{V}ar(c)$ denotes the set of free variables of the constraint c.

- An interpretation $\mathcal{K}$ is a Σ-structure of domain K and a solution mapping that associates to each constraint the set of assignments $Sol_{\mathcal{K}}(c)$ defined as follows :
  - $Sol_{\mathcal{K}}(\mathbf{T}) = \{\alpha \in ASS_K^{\mathcal{X}}\}$,
  - $Sol_{\mathcal{K}}(t_1 =^? t_2) = \{\alpha \in ASS_K^{\mathcal{X}} \mid \underline{\alpha}(t_1) = \underline{\alpha}(t_2)\}$,
  - $Sol_{\mathcal{K}}(p^?(t_1, \ldots, t_m)) = \{\alpha \in ASS_K^{\mathcal{X}} \mid (\underline{\alpha}(t_1), \ldots, \underline{\alpha}(t_m)) \in p_{\mathcal{K}}\}$,
  - $Sol_{\mathcal{K}}(c \wedge c') = Sol_{\mathcal{K}}(c) \cap Sol_{\mathcal{K}}(c')$.
  - $Sol_{\mathcal{K}}(\neg c) = ASS_K^{\mathcal{X}} \backslash Sol_{\mathcal{K}}(c)$.
  - $Sol_{\mathcal{K}}(\exists x : c) = \{\alpha \in ASS_K^{\mathcal{X}} \mid \exists \beta \in ASS_K^{\mathcal{X}},\ \alpha_{|\mathcal{X}\backslash\{x\}} = \beta_{|\mathcal{X}\backslash\{x\}}$ and $\beta \in Sol_{\mathcal{K}}(c)\}$.

An assignment in $Sol_{\mathcal{K}}(c)$ is a *solution* of c in $L_{\mathcal{K}}$. A constraint c is *valid* in $L_{\mathcal{K}}$, written $L_{\mathcal{K}} \models c$, if any assignment is a solution of c in $L_{\mathcal{K}}$.

Abbreviations may be defined to write more complex constraints like : $c \vee c' = \neg(\neg c \wedge \neg c')$, $\forall x : c = \neg(\exists x : \neg c)$ and $\mathbf{F} = \neg(\mathbf{T})$. Two constraints c and c' are *equivalent* if $Sol_{\mathcal{K}}(c) = Sol_{\mathcal{K}}(c')$, which denoted $c \equiv_{\mathcal{K}} c'$.

Of particular interest for theorem proving, is the case where the interpretation $\mathcal{K}$ is isomorphic to a term algebra. Then the notion of a *symbolic solution* of a constraint c coincides with a substitution σ such that $\mathcal{K} \models \sigma(c)$, two symbolic solutions may be compared with the substitution ordering denoted here by $\leq_{\mathcal{K}}^{\mathcal{V}(c)}$, and the notion of complete set of solutions is available.

Definition 7.32 A *symbolic solution* of a $L_{\mathcal{K}}[\Sigma, \mathcal{X}]$-constraint c is a substitution σ such that $L_{\mathcal{K}} \models \sigma(c)$. The set of all symbolic solutions of c is denoted $SS_{\mathcal{K}}(c)$.

A substitution ϕ is an *instance* on $V \subseteq \mathcal{X}$ of a substitution σ, written $\sigma \leq_{\mathcal{K}}^{V} \phi$, if there exists some substitution μ such that $\forall x \in V,\ L_{\mathcal{K}} \models \phi(x) = \mu(\sigma(x))$.

Definition 7.33 A set of substitutions is a *complete set of solutions* of the $L_{\mathcal{K}}[\Sigma, \mathcal{X}]$-constraint c, denoted by $CSS_{\mathcal{K}}(c)$, if
(1) $\forall \sigma \in CSS_{\mathcal{K}}(c),\ \mathcal{D}om(\sigma) \cap \mathcal{VR}an(\sigma) = \emptyset$ (idempotency).
(2) $CSS_{\mathcal{K}}(c) \subseteq SS_{\mathcal{K}}(c)$ (correctness).
(3) $\forall \phi \in SS_{\mathcal{K}}(c),\ \exists \sigma \in CSS_{\mathcal{K}}(c),\ \sigma \leq_{\mathcal{K}}^{\mathcal{V}ar(c)} \phi$ (completeness).
When two substitutions of $CSS(c)$ cannot be compared with $\leq_{\mathcal{K}}^{\mathcal{V}ar(c)}$, the complete set of solutions $CSS_{\mathcal{K}}(c)$ is *minimal*.

The set $SS_{\mathcal{K}}(c)$ of symbolic solutions of the $L_{\mathcal{K}}$-constraint c is a complete set of solutions.

- An equational presentation given by a signature $(\mathcal{S}, \mathcal{F}, \mathcal{P})$ where the only predicate is $=$ and a set of equational axioms E, defines an *equational* constraint language where atomic constraints are *equations* over $\mathcal{T}(\mathcal{S}, \mathcal{F}, \mathcal{X})$. The standard interpretation is the quotient algebra $\mathcal{T}(\mathcal{S}, \mathcal{F}, \mathcal{X})/=_E$. A symbolic solution σ of a constraint c is an *E-unifier*. A complete set of solutions of a constraint c is denoted $CSS_E(c)$ or $CSU_E(c)$ since it is also a *complete set of E-unifiers*. For instance, if $\mathcal{F} = \{a, f\}$, $\mathcal{X} = \{v, x, y\}$ and E consists of the associativity and commutativity axioms for f, then $(f(v, x) =_{\emptyset}^{?} f(a, f(x, y)))$ is an equational constraint.

- A finite $(\mathcal{S}, \mathcal{F})$-algebra $\mathcal{A}$, together with a set of relations $\mathcal{P}_A$ on A, defines a constraint language $L_{\mathcal{A}}[\mathcal{S}, \mathcal{F}, \mathcal{P}]$ where $\mathcal{A}$ is the interpretation of interest. For simplicity, we assume that the finite functions defined on $\mathcal{A}$ are all defined on the same set and so $\mathcal{S}$ is reduced to one sort. In [KR92], constraints are solved in this language by embedding it a *primal constraint language* whose standard interpretation is isomorphic to a quotient term algebra.

7.6.2 Constrained equalities and rewrite rules

Using constraints, we are now ready to build formulas with constraints, especially equalities and rewrite rules. Such formulas are built by extending the signature $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P})$ with at least two new predicate symbols, namely $=$ for building equalities with constraints and $\rightarrow$ for rewrite rules with constraints. They are built in full generality on an extended signature Σ' and a superset of variables $\mathcal{X}'$.

Definition 7.34 Let $\Sigma \subseteq \Sigma'$ and $\mathcal{X} \subseteq \mathcal{X}'$. A *constrained equality*, denoted $(l = r \parallel c)$, is given by two terms l and r in $\mathcal{T}(\Sigma', \mathcal{X}')$ and a constraint c in $L_{\mathcal{K}}[\Sigma, \mathcal{X}]$.

The constrained equality $(l = r \parallel c)$ schematizes the following set of equalities on $\mathcal{T}(\Sigma', \mathcal{X}')$: $\mathcal{S}(l = r \parallel c) = \{\sigma(l) = \sigma(r) \mid \sigma \in SS_{\mathcal{K}}(c)\}$.

If there exists an ordering such that all these instances may be used from left to right, they are better represented by constrained rules.

Definition 7.35 Let $\Sigma \subseteq \Sigma'$ and $\mathcal{X} \subseteq \mathcal{X}'$. A *constrained rewrite rule*, denoted $(l \to r \parallel c)$, is given by two ordered terms l, r in $\mathcal{T}(\Sigma', \mathcal{X}')$ and a constraint c in $L_{\mathcal{K}}[\Sigma, \mathcal{X}]$. It is moreover assumed that $\mathcal{V}ar(c) \subseteq \mathcal{V}ar(l) \cup \mathcal{V}ar(r)$.

A constrained rewrite rule $(l \to r \parallel c)$ schematizes the following set of rewrite rules:
$\mathcal{S}(l \to r \parallel c) = \{ \sigma(l) \to \sigma(r) \mid \sigma \in SS_{\mathcal{K}}(c) \}$.

7.6.3 Rewriting with constraints

Let us first give a very general definition of rewriting with constraints in which the matching theory may be different from the theory in which constraints are solved.

Definition 7.36 Given a set CR of constrained rewrite rules with constraints in $L_{\mathcal{K}}[\Sigma, \mathcal{X}]$ and a set A of axioms, a term t $(CR, A, L_{\mathcal{K}})$*-rewrites* to a term t', which is denoted by $t \to_{CR,A,L_{\mathcal{K}}} t'$ if there exist a constrained rewrite rule $(l \to r \parallel c)$ of CR, a position ω in t, a substitution σ, satisfying $t_{|\omega} \overset{*}{\longleftrightarrow}_A \sigma(l)$, and $\sigma \in SS_{\mathcal{K}}(c)$. Then $t' = t[\sigma(r)]_{\omega}$.

With this definition, a constrained rewrite rule is applicable if there exists a substitution that matches the left-hand side and makes the constraint hold in the interpretation $\mathcal{K}$. In its full generality, this $(CR, A, L_{\mathcal{K}})$ rewrite relation allows the use of built-in constraint solvers in the structure $\mathcal{K}$, but this leads to combination problems in the matching and unification processes underlying rewriting and superposition with constrained rewrite rules. These matching and unification problems must be solved in a conservative extension of $\mathcal{K}$ taking into account axioms in A and all equalities valid in $\mathcal{K}$. This problem is addressed in [KR93].

Definition 7.37 Let CR be a set of constrained rewrite rules and $\mathcal{S}(CR)$ the schematized set of rewrite rules. The relation $(CR, A, L_{\mathcal{K}})$ is Church-Rosser modulo A on a set of terms $\mathcal{T}$ if
$\overset{*}{\longleftrightarrow}_{\mathcal{S}(CR) \cup A} \;\subseteq\; \overset{*}{\longrightarrow}_{CR,A,L_{\mathcal{K}}} \circ \overset{*}{\longleftrightarrow}_A \circ \overset{*}{\longleftarrow}_{CR,A,L_{\mathcal{K}}}$.

Example 7.15 Examples of ordered rewrite system given in [MN90] can actually be described by a constrained rewrite system. Assume given a reduction ordering $>$ total on ground terms and satisfying for all ground terms x, y, z:

$$\begin{array}{rcll} (x*y)*z & > & x*(y*z) & \\ x*y & > & y*x & \text{if} \quad x > y \\ x*(y*z) & > & y*(x*z) & \text{if} \quad x > y \end{array}$$

Then the following set CR is a constrained rewrite system, where $A = \emptyset$:

$$\begin{array}{l} (x*y)*z \to x*(y*z) \parallel \mathbf{T} \\ x*y \to y*x \parallel x >^?_{\emptyset} y \\ x*(y*z) \to y*(x*z) \parallel x >^?_{\emptyset} y \end{array}$$

If $>$ is the lexicographic path ordering and a, b, c are constants such that $c > b > a$:

$$b*(c*(b*a)) \to_{CR} b*(c*(a*b)) \to_{CR} b*(a*(c*b)) \to_{CR} a*(b*(c*b)) \to_{CR} a*(b*(b*c)).$$

This constrained rewrite system allows deciding the word problem for associativity and commutativity of $*$.

Example 7.16 Consider the set of constrained rewrite rules:

$$\begin{array}{rcl} append(nil, y) & \to & y \\ append(x, y) & \to & cons(u, z) \quad \parallel \quad (x =^?_{\emptyset} cons(u, v) \wedge append(v, y) =^?_{\emptyset} z) \end{array}$$

We have

$$append(cons(a, nil), nil) \to cons(a, append(v, nil))$$

with the match $(x \mapsto cons(a, nil))(y \mapsto nil)$, since $(u \mapsto a)(v \mapsto nil)(y \mapsto nil)(z \mapsto append(v, nil))$ is a solution of $(cons(a, nil) =^?_{\emptyset} cons(u, v) \wedge append(v, y) =^?_{\emptyset} z)$.

Reflexivity

$$\begin{array}{l} \mapsto\!\!\!\twoheadrightarrow \\ \langle t\rangle_A \rightarrow \langle t\rangle_A \ \| \ \mathbf{T} \end{array}$$

Congruence

$$\begin{array}{l} \langle t_1\rangle_A \rightarrow \langle t'_1\rangle_A \ \| \ c_1, \ldots, \langle t_n\rangle_A \rightarrow \langle t'_n\rangle_A \ \| \ c_n \\ \mapsto\!\!\!\twoheadrightarrow \\ \langle f(t_1, \ldots, t_n)\rangle_A \rightarrow \langle f(t'_1, \ldots, t'_n)\rangle_A \ \| \ c_1 \wedge \ldots \wedge c_n \\ \text{if } c_1 \wedge \ldots \wedge c_n \text{ satisfiable} \end{array}$$

Replacement

$$\begin{array}{l} \langle t_1\rangle_A \rightarrow \langle t'_1\rangle_A \ \| \ c_1 \ldots \langle t_n\rangle_A \rightarrow \langle t'_n\rangle_A \ \| \ c_n \\ \mapsto\!\!\!\twoheadrightarrow \\ \langle l(t_1, \ldots, t_n)\rangle_A \rightarrow \langle r(t'_1, \ldots, t'_n)\rangle_A \ \| \ c_1 \wedge \ldots \wedge c_n \wedge c(t_1, \ldots, t_n) \\ \text{if } \begin{array}{l} (l(x_1, \ldots, x_n) \rightarrow r(x_1, \ldots, x_n) \ \| \ c(x_1, \ldots, x_n)) \in CR \\ \text{and } c_1 \wedge \ldots \wedge c_n \wedge c(t_1, \ldots, t_n) \text{ satisfiable} \end{array} \end{array}$$

Transitivity

$$\begin{array}{l} \langle t_1\rangle_A \rightarrow \langle t_2\rangle_A \ \| \ c_1, \ \langle t_2\rangle_A \rightarrow \langle t_3\rangle_A \ \| \ c_2 \\ \mapsto\!\!\!\twoheadrightarrow \\ \langle t_1\rangle_A \rightarrow \langle t_3\rangle_A \ \| \ c_1 \wedge c_2 \\ \text{if } c_1 \wedge c_2 \text{ satisfiable} \end{array}$$

Figure 7.1: **CONSREW**: Constrained rewrite deduction

7.6.4 Comparison with conditional rewriting

The notion of constrained rewriting bears much similarity with conditional rewriting, especially with contextual rewriting. However, in conditional rewriting, occurrences of the same function symbol in conditions and in conclusion are usually interpreted in the same way. This is no more true with constrained rewriting, where the symbols in constraints are subject to special deduction rules. For instance, an equation $(f(s) =^?_\emptyset f(t))$ in a constraint may be decomposed into $(s =^?_\emptyset t)$. Such a transformation is in general not valid in the first-order theory which underlies the constrained formula. The difference between constrained and conditional rewriting also appears for instance in the following example of idempotent semigroups from [SS82a]. It is a noetherian confluent "conditional" system for the theory of an idempotent associative symbol $*$.

$$\begin{array}{rcl} x * x & \rightarrow & x \\ x * y * z & \rightarrow & x * z \ \text{ if } \ (x =^?_{ACI} z) \wedge (x * y =^?_{ACI} z) \end{array}$$

where ACI is the theory of an associative commutative idempotent symbol. So the equations in the condition are solved modulo the theory ACI of $*$, while the rules are used with matching modulo associativity. Since the theory of $*$ is different in the constraints, due to the commutativity axiom, we feel that this system is typically a constrained rewrite system. Actually this system does not fit into the classical frameworks for conditional term rewriting [KR89b].

7.6.5 A constrained rewriting logic

Following [Mes92], a general logic setting can be proposed to formalize constrained rewrite deduction. In the constrained rewriting logic sketched now, sentences are defined as constrained sequents of the form $(\langle t\rangle_A \rightarrow \langle t'\rangle_A \ \| \ c)$ where $t, t' \in \mathcal{T}(\Sigma, \mathcal{X})$, $\langle t\rangle_A$ denotes the A-equivalence class of t and $c \in L_\mathcal{K}$. The informal meaning of such sentences is that t' is derived from t if c is satisfied. A constrained rewrite theory is a set CR of constrained rewrite rules. Each rule $(l \rightarrow r \ \| \ c)$ has a finite set of variables $\mathcal{V}ar(l) \cup \mathcal{V}ar(r) \cup \mathcal{V}ar(c) = \{x_1, \ldots, x_n\}$ which are recorded in the notation $(l(x_1, \ldots, x_n) \rightarrow r(x_1, \ldots, x_n) \ \| \ c(x_1, \ldots, x_n))$. A theory CR entails the sequent $(\langle t\rangle_A \rightarrow \langle t'\rangle_A \ \| \ c)$, if it is obtained by the finite application of the deduction rules in Figure 7.1.

A sequent is a *one-step CR-rewrite*, if it can be derived from CR by application of the rules **Reflexivity**, **Congruence** and exactly one application of **Replacement**. The relation between a one step CR-rewrite and the constraint rewriting relation previously defined on terms is made precise in the following lemma.

Lemma 7.3 A sequent $\langle t\rangle_A \rightarrow \langle t'\rangle_A \ \| \ c$ is a one step CR-rewriting iff there exist a rule $(l \rightarrow r \ \| \ c)$ in CR, a substitution σ and a position ω of t such that $t \rightarrow^{\omega,\sigma,(l\rightarrow r \,\|\, c)}_{CR,A,L_\mathcal{K}} t'$.

Proof: Immediate induction on the form of the proof of the sequent $\langle t\rangle_A \rightarrow \langle t'\rangle_A \ \| \ c$. □

7.7 Conclusion

This chapter gathered several proposed extensions of rewriting that appeared to be of prime interest for theorem provers or for programming languages. The application of rewriting to the definition of operational semantics of logico-functional programming languages led to various other extensions not covered here, like order-sorted rewriting [KKM88], priority rewriting [Moh89], or graph rewriting [B$^+$87] .... In the domain of automated theorem proving, rewriting techniques are of primarily use in provers using demodulation or simplification inference rules to prune the search space. In this context, it appears that most of interesting proofs in mathematical structures, set and graph theory, or geometry, involve in their axiomatization some equalities which cannot be immediately used as rewrite rules. This motivates the introduction of several extensions, especially ordered rewriting, class rewriting and rewriting with constraints. This chapter has shown the evolution between them and their increasing power of expressivity. Although these three extensions are better motivated by theorem proving purposes, they also have promising applications in programming languages.

Chapter 8

Modular properties of rewrite systems

8.1 Introduction

For ascertaining properties of rewrite systems such as confluence or termination when there are many rewrite rules, it is obviously important to have results which state that a rewrite system has a property Prop if that system can be partitioned into smaller ssystems which have the property Prop. Two very simple counterexamples show there is no hope to apply this divide and conquer approach in general for the properties of confluence and termination. Consider for instance $R_1 = \{a \rightarrow b\}$, $R_2 = \{b \rightarrow a\}$ and $R_3 = \{a \rightarrow c\}$ where a, b, c are constants. Each of these systems is confluent and terminating, but $R_1 \cup R_2$ is not terminating and $R_1 \cup R_3$ is not confluent. In these examples however, function symbols are shared and a natural idea is to eliminate this case. The research on modularity for disjoint rewrite systems originated with Toyama [Toy87] who proved that the disjoint union of two confluent rewrite systems is confluent. In [Toy86], Toyama refuted the fact that the disjoint union of two terminating rewrite systems is terminating. His counterexample inspired the formulation by Rusinowitch [Rus87b] of sufficient syntactic restrictions on the rules, in terms of collapsing and duplicating rules, to keep the property of termination. These first results have been extended by Middeldorp [Mid89c], who also considered the case of conditional rewrite systems [Mid89a, Mid89d]. The disjointness assumption was relaxed in the case of constructor systems that are allowed to share constructors, while preserving the confluence and termination properties [MT91]. A survey of properties of rewrite systems preserved under (disjoint) unions can be found in [Mid90]. Most of the proofs for the results presented in this chapter can be found there, as well as results about the modular properties of weakly normalizing (conditional) rewrite systems that we do not consider here.

It must be clear however that properties of confluence and termination of the union of two abstract reduction systems, or two rewrite systems, have also been considered without the disjointness assumption. But then a commutation property between the considered abstract relations is needed to prove that the property is modular.

The results presented in this chapter are divided into three parts, according to the hypotheses put on the intersection of systems. Disjoint systems are considered first, then results are extended to constructor systems and other results on the union of systems are given last.

8.2 Modularity

Before stating any results, it is useful to precisely define the notion of modular property.

Definition 8.1 A given property Prop is *modular* if Prop is satisfied by R_1 and R_2 iff Prop is satisfied by the union of R_1 and R_2.

We are interested in studing the modularity of the properties of abstract reduction systems in general or of term rewrite systems in particular: local confluence, termination and confluence.

8.3 Disjoint systems

When R_1 and R_2 are rewrite systems built on disjoint signatures, their union is denoted $R_1 \oplus R_2$. All results given in this section assume disjoint signatures for the systems.

8.3.1 Confluence and local confluence

In the context of disjoint rewrite systems, the first modular property stated by Toyama is confluence. This result is important not only from an historical point of view, but also because many following results rely on this result.

Theorem 8.1 *[Toy87] Confluence is a modular property of rewrite systems.*

The proof relies on a commutation property between the reduction relation of R_1 and the reduction of R_2. More details on various commutation properties will be given in Section 8.5.

Concerning local confluence, the proof of modularity is easy by using the technique for proving local confluence from the computation of critical pairs, detailed in Chapter 16. Indeed there cannot exist critical pairs between two rewrite systems on disjoint sets of function symbols.

Theorem 8.2 *[Mid90] Local confluence is a modular property of rewrite systems.*

The generalization of these results to conditional rewrite systems needs to be careful.

Theorem 8.3 *[Mid90] Confluence is a modular property of natural conditional rewriting.*

However it is also proved in [Mid90] that local confluence is not a modular property of conditional rewrite systems.

8.3.2 Termination

The second interesting property of rewrite systems to consider is termination. However, termination is not a modular property, as shown in the next counterexamples.

Example 8.1 The rewrite system

$$f(a,b,x) \quad \rightarrow \quad f(x,x,x)$$

is terminating [Toy86], as well as

$$\begin{array}{rcl} g(x,y) & \rightarrow & x \\ g(x,y) & \rightarrow & y. \end{array}$$

However in the disjoint union of both systems, there is a cycle:

$$\begin{array}{rcl} f(g(a,b),g(a,b),g(a,b)) & \rightarrow & f(a,g(a,b),g(a,b)) \\ & \rightarrow & f(a,b,g(a,b)) \\ & \rightarrow & f(g(a,b),g(a,b),g(a,b)) \end{array}$$

Example 8.2 Another example, due to Drosten, of two confluent and terminating rewrite systems is as follows. Let R_1 be

$$\begin{array}{rcl} f(0,1,x) & \rightarrow & f(x,x,x) \\ f(x,y,z) & \rightarrow & 2 \\ 0 & \rightarrow & 2 \\ 1 & \rightarrow & 2 \end{array}$$

and R_2 be

$$\begin{array}{rcl} d(x,y,y) & \rightarrow & x \\ d(x,x,y) & \rightarrow & y. \end{array}$$

However The disjoint sum $R_1 \oplus R_2$ is not terminating because the term $f(d(0,1,1),d(0,1,1),d(0,1,1))$ has a cyclic derivation:

$$\begin{array}{rcl} f(d(0,1,1),d(0,1,1),d(0,1,1)) & \rightarrow & f(0,d(0,1,1),d(0,1,1)) \\ & \rightarrow & f(0,d(2,1,1),d(0,1,1)) \\ & \rightarrow & f(0,d(2,2,1),d(0,1,1)) \\ & \rightarrow & f(0,1,d(0,1,1)) \\ & \rightarrow & f(d(0,1,1),d(0,1,1),d(0,1,1)). \end{array}$$

Adding syntactic conditions on rewrite rules allow getting positive results.

Definition 8.2 A rewrite rule $l \to r$ is a *collapsing rule* if r is a variable.

A rewrite rule $l \to r$ is a *duplicating rule* if there exists a variable that has more occurrences in r than in l.

The results are summarized in the next theorem:

Theorem 8.4 *Let R_1 and R_2 be two terminating rewrite systems.*

1. *If neither R_1 nor R_2 contain collapsing rules, then $R_1 \oplus R_2$ is terminating.*
2. *If neither R_1 nor R_2 contain duplicating rules, then $R_1 \oplus R_2$ is terminating.*
3. *If one of the systems R_1, R_2 contains neither collapsing rules nor duplicating rules, then $R_1 \oplus R_2$ is terminating.*

Proof: (1) and (2) are proved in [Rus87b]. (3) is proved in [Mid89c]. □

8.3.3 Simple termination

More subtle results can be obtained when considering specific methods for proving termination. A first result confirms the observation that common classes of precedence-based simplification orderings exhibit a modular behaviour simply by combining the corresponding disjoint precedences.

Theorem 8.5 *[Gra92, KO90] Termination is modular for the class of (finite) disjoint systems whose termination is proved by simplification ordering.*

The proof presented in [Gra92] relies on the expression of (undecicable) sufficient conditions for the nontermination of the disjoint union of two rewrite systems. The restriction to finite system was required in [KO90].

8.3.4 Normal form and convergence

Remind that the unique normal form property of a rewrite system R means that any term has a unique normal form (but may also have infinite derivations).

Theorem 8.6 *[Mid89b] The unique normal form property is modular.*

The proof is based on the fact that every term rewrite system with unique normal forms can be conservatively extended to a confluent rewrite system with the same normal forms. This method also works for generalizing the result to natural conditional rewriting, but fails for join conditional rewriting.

If a restriction to left-linear rules is added, another positive result is obtained.

Theorem 8.7 *[TKB89] Let R_1 and R_2 be left-linear rewrite systems. Then $R_1 \oplus R_2$ is convergent iff both R_1 and R_2 are convergent.*

8.4 Constructor systems

A natural question is how to relax the disjointness assumption in the previous results. In [Mid91], A. Middeldorp and Y. Toyama addressed the problem of convergence of the union of constructor systems, that is systems in which all function symbols are constructors, except those at the top of left-hand sides of rewrite rules.

Definition 8.3 A *constructor system* $(\mathcal{C}, \mathcal{D}, R)$ is defined by a set of constructors $\mathcal{C}$, a set of defined functions $\mathcal{D}$ and a set of rewrite rules R, such that every left-hand side of any rule in R is of the form $f(t_1, ..., t_n)$ with $f \in \mathcal{D}$ and $t_1, ..., t_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$.

Two constructor systems $(\mathcal{C}_1, \mathcal{D}_1, R_1)$ and $(\mathcal{C}_2, \mathcal{D}_2, R_2)$ share constructors if $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{C}_1 \cup \mathcal{C}_2$ are pairwise disjoint.

The next example also illustrate some negative results.

Example 8.3 Consider the two constructor systems $(\mathcal{C}_1, \mathcal{D}_1, R_1)$ and $(\mathcal{C}_2, \mathcal{D}_2, R_2)$ where $\mathcal{C}_1 = \{s, a, b\}$, $\mathcal{D}_1 = \{f\}$,

$$R_1 : \begin{array}{rcl} f(x,x) & \rightarrow & a \\ f(x, s(x)) & \rightarrow & b \end{array}$$

$\mathcal{C}_2 = \{s\}$, $\mathcal{D}_2 = \{c\}$,

$$R_2 : \quad c \quad \rightarrow \quad s(c)$$

Each of them is confluent, but their union is not, since the term $f(c, s(c))$ rewrites to both a and b that are not joinable.

Exercise: Find two terminating constructor systems whose union is not terminating. (Example 8.1 provides an example)

However, constructor systems enjoy several modular properties.

Theorem 8.8 *[Mid91] The union of terminating and confluent (resp. locally confluent) constructor systems with shared constructors is terminating and confluent (resp. locally confluent).*

The theorem also holds for weak convergence, that is confluence and weak normalization.

This result yields the possibility to decompose some systems into parts sharing some function symbols and rewrite rules; the convergence property of the (non disjoint) union is then inferred from the convergence of the sub-systems.

A constructor system is convergent if it can be decomposed into convergent constructor systems. It is important to note that the notion of decomposition does not imply disjointness. This method is illustrated by the two following examples.

Example 8.4 [Mid91] Consider the constructor system R with constructor set $\{0, s\}$:

$$\begin{array}{rcl} 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x+y) \\ 0 \times x & \rightarrow & 0 \\ s(x) \times y & \rightarrow & (x \times y) + y \\ f(0) & \rightarrow & 0 \\ f(s(x)) & \rightarrow & f(x) + s(x). \end{array}$$

It can be decomposed into R_1

$$\begin{array}{rcl} 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x+y) \\ 0 \times x & \rightarrow & 0 \\ s(x) \times y & \rightarrow & (x \times y) + y \end{array}$$

and R_2

$$\begin{array}{rcl} 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x+y) \\ f(0) & \rightarrow & 0 \\ f(s(x)) & \rightarrow & f(x) + s(x). \end{array}$$

Both systems are easily shown convergent, so is their union.

Example 8.5 [Mid91] Consider the constructor system R with constructor set $\{0, s, true, false\}$:

$$\begin{array}{rcl} 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x+y) \\ 0 \times x & \rightarrow & 0 \\ s(x) \times y & \rightarrow & (x \times y) + y \\ fib(0) & \rightarrow & s(0) \\ fib(s(0)) & \rightarrow & s(0) \end{array}$$

$$\begin{array}{rcl} fib(s(s(x))) & \rightarrow & fib(s(x)) + fib(x) \\ x < 0 & \rightarrow & false \\ 0 < s(x) & \rightarrow & true \\ s(x) < s(y) & \rightarrow & x < y \\ true \wedge false & \rightarrow & false \\ false \wedge true & \rightarrow & false \\ x \wedge x & \rightarrow & x \end{array}$$

It can be decomposed into R_1 with constructors $C_1 = \{0, s\}$

$$\begin{array}{rcl} 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x + y) \\ 0 \times x & \rightarrow & 0 \\ s(x) \times y & \rightarrow & (x \times y) + y \end{array}$$

R_2 with constructors $C_2 = \{0, s\}$

$$\begin{array}{rcl} 0 + x & \rightarrow & x \\ s(x) + y & \rightarrow & s(x + y) \\ fib(0) & \rightarrow & s(0) \\ fib(s(0)) & \rightarrow & s(0) \\ fib(s(s(x))) & \rightarrow & fib(s(x)) + fib(x) \end{array}$$

R_3 with constructors $C_3 = \{0, s, true, false\}$

$$\begin{array}{rcl} x < 0 & \rightarrow & false \\ 0 < s(x) & \rightarrow & true \\ s(x) < s(y) & \rightarrow & x < y \end{array}$$

and R_4 with constructors $C_3 = \{true, false\}$

$$\begin{array}{rcl} true \wedge false & \rightarrow & false \\ false \wedge true & \rightarrow & false \\ x \wedge x & \rightarrow & x. \end{array}$$

All four systems are easily shown complete, so is their union.

Thess results are yet improved by Kurihara and Ohuchi. In their approach constructors are any symbol that does not appear at the top of a left-hand side of rules. But they do not impose further restrictions on the form of left-hand sides of rewrite rules as in Definition 8.3.

Theorem 8.9 *[KO92] The union of simply terminating systems with shared constructors is simply terminating.*

An interesting example is given by Kurihari and Ohuchi in [KO92], where other methods cannot apply.

Example 8.6 The two rewrite systems $R_1 = \{f(f(a, x), x) \rightarrow f(x, f(b, x))\}$ and $R_2 = \{g(b) \rightarrow g(a)\}$ share constructors a and b. Termination of R_1 can be proved using a recursive path ordering induced by a precedence where a is greater than b, while termination of R_2 can be proved using a recursive path ordering induced by a precedence where b is greater than a. Indeed this prevents the possibility to find a recursive path ordering combining the two precedences for proving termination of $R_1 \cup R_2$. Theorem 8.13 does not apply because the rule of R_1 is neither left- nor right-linear. Theorem 8.8 does not apply either because R_1 is not a constructor system. Of course, Theorem 8.9 applies and yields the simple termination of $R_1 \cup R_2$.

Unfortunately confluence is not modular when constructors are shared as demonstrated in the next example:

Example 8.7 [Hue80] Consider $R_1 = \{f(x, x) \rightarrow a,\ f(x, h(x)) \rightarrow b\}$ and $R_2 = \{c \rightarrow h(c)\}$ which are both confluent and share the constructor h. However in $R_1 \cup R_2$, the term $f(c, c)$ has two normal forms a and b.

From Theorem 8.9, it is easy to get modularity of simple convergent systems.

Theorem 8.10 *[KO92] Let R_1 and R_2 be two simply terminating and confluent rewrite systems with shared constructors. Then $R_1 \cup R_2$ is simply terminating and confluent.*

Proof: Indeed $R_1 \cup R_2$ is terminating and there is no critical pair between R_1 and R_2. $\Box$

8.5 Non-disjoint systems with commutation properties

When it is impossible to use the disjointness assumption, to decompose into constructor systems, or to share only constructors, it is useful to have results about modularity expressed at a more abstract level of relations and abstract reduction systems.

8.5.1 Confluence

In the very general framework of abstract reduction systems, one of the first results on Church-Rosser properties was obtained by Hindley [Hin64]Hindley and Rosen [Ros73]Rosen. It states that to get confluence without termination, some commutation property must hold.

Definition 8.4 Let $\langle \mathcal{T}, (\rightarrow_i)_{i \in I} \rangle$ be an abstract reduction system. For some $i, j \in I$, $\rightarrow_i$ *commutes* with $\rightarrow_j$ if $\leftarrow_i \circ \rightarrow_j \quad \subseteq \quad \rightarrow_j \circ \leftarrow_i$.

Lemma 8.1 [Hin64, Ros73] (Lemma of Hindley-Rosen)
Let $\langle \mathcal{T}, (\rightarrow_i)_{i \in I} \rangle$ be confluent abstract reduction systems such that for any $i, j \in I$, $\rightarrow_i$ commutes with $\rightarrow_j$. Then $\bigcup_{i \in I} \rightarrow_i$ is confluent.

This lemma may be rephrased as the statement of a modular property.

Lemma 8.2 Let $\langle \mathcal{T}_1, \rightarrow_1 \rangle$ and $\langle \mathcal{T}_2, \rightarrow_2 \rangle$ be two confluent abstract reduction systems such that $\rightarrow_1$ commutes with $\rightarrow_2$ and $\rightarrow_2$ commutes with $\rightarrow_1$. Then $\langle \mathcal{T}_1 \cup \mathcal{T}_2, \rightarrow_1 \cup \rightarrow_2 \rangle$ is confluent, i.e. confluence is a modular property of commuting abstract reduction systems.

Sufficient conditions for commutation are stated on abstract reduction relations in [Hin64, Sta75]. However the next example shows that commutation is not a necessary condition for confluence of the union of confluent abstract reduction systems.

Example 8.8 Let $\langle \mathcal{T}, (\rightarrow_i)_{i \in \{1,2\}} \rangle$ be defined by $\{a \rightarrow_1 b,\ b \rightarrow_1 c\}$ and $\{a \rightarrow_2 c\}$. $\langle \mathcal{T}, \rightarrow_1 \rangle$, $\langle \mathcal{T}, \rightarrow_2 \rangle$ and $\langle \mathcal{T}, (\rightarrow_i)_{i \in \{1,2\}} \rangle$ are confluent, although $\rightarrow_1$ and $\rightarrow_2$ clearly do not commute.

8.5.2 Termination

A sufficient condition useful to prove termination of the union $\langle \mathcal{T}_1 \cup \mathcal{T}_2, \rightarrow_1 \cup \rightarrow_2 \rangle$ of two abstract reduction systems is based in the idea that in every sequence of reduction, only finitely many steps of $\rightarrow_1$ can occur. This property is called relative termination.

Definition 8.5 Let $\mathcal{A}_1 = \langle \mathcal{T}_1, \rightarrow_1 \rangle$ and $\mathcal{A}_2 = \langle \mathcal{T}_2, \rightarrow_2 \rangle$ two abstract reduction systems. Then $\mathcal{A}_1/\mathcal{A}_2$ denotes $\langle \mathcal{T}_1 \cup \mathcal{T}_1, \xrightarrow{*}_2 \circ \rightarrow_1 \circ \xrightarrow{*}_2 \rangle$.

$\mathcal{A}_1$ is *relatively terminating* w.r.t. $\mathcal{A}_2$ if $\mathcal{A}_1/\mathcal{A}_2$ is terminating.

Then termination of $\mathcal{A}_1/\mathcal{A}_2$ allows proving termination of $\mathcal{A}_1 \cup \mathcal{A}_2$:

Lemma 8.3 [BD86a] The union of two terminating abstract reduction systems $\mathcal{A}_1$ and $\mathcal{A}_2$ is terminating iff $\mathcal{A}_1/\mathcal{A}_2$ is terminating.

This termination property can in turn be proved using a refined commutation property.

Definition 8.6 Let $\langle \mathcal{T}, (\rightarrow_i)_{i \in I} \rangle$ be an abstract reduction system. For some $i, j \in I$, $\rightarrow_1$ *quasi-commutes* over $\rightarrow_2$ if $\rightarrow_2 \circ \rightarrow_1 \subseteq \rightarrow_1 \circ (\rightarrow_1 \cup \rightarrow_2)^*$.

Proposition 8.1 [BD86a] Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two abstract reduction systems. If $\mathcal{A}_1$ is terminating and quasi-commutes over $\mathcal{A}_2$, then $\mathcal{A}_1/\mathcal{A}_2$ is terminating.

From Lemma 8.3 and Proposition 8.1, one easily get:

Theorem 8.11 *[BD86a] The union of two terminating abstract reduction systems $\mathcal{A}_1$ and $\mathcal{A}_2$ is terminating if $\mathcal{A}_1$ quasi-commutes over $\mathcal{A}_2$.*

From a more operational point of view, another idea to combine complex proofs of termination is to take the union of two terminating relations. Again, in order to state that the union of two well-founded-orderings is well-founded, a commutation property is needed. This is the case for reduction orderings in general.

Definition 8.7 Let consider two well-founded-orderings $>_{\mathcal{R}}$ and $>_{\mathcal{S}}$. $>_{\mathcal{S}}$ *commutes* over $>_{\mathcal{R}}$ if $>_{\mathcal{R}} \circ >_{\mathcal{S}} \subseteq >_{\mathcal{S}} \circ >_{\mathcal{R}}$.

Proposition 8.2 [DJ90a] If R and S are two relations on a set T, contained into two reduction orderings that commute, then $R \cup S$ is terminating on T.

Example 8.9 The union of $R = \{f(a) \rightarrow f(b)\}$ and $S = \{g(b) \rightarrow g(a)\}$ is terminating, by identifying the rewrite system and the associated rewrite relation.

This result is important since it allows combining a rewriting relation and various ordering on terms, which is especially useful for noetherian induction.

The union of a terminating rewriting relation with the strict subterm ordering $\lhd^{sub}$, the strict subsumption ordering $<$ or the strict encompassment ordering $\sqsubset$, is well-founded:

- If R is a terminating rewrite system, $\rightarrow_R \cup \lhd^{sub}$ is well-founded.
- If R is a terminating rewrite system, $\rightarrow_R \cup <$ is well-founded.
- If R is a terminating rewrite system, $\rightarrow_R \cup \sqsubset$ is well-founded.

Another possibility is to consider the property of weak termination and to restrict to innermost rewriting.

Theorem 8.12 *If two rewrite systems are weakly terminating with an innermost strategy, then their disjoint union is weakly terminating.*

Theorem 8.13 *Let R_1 and R_2 be two terminating rewrite systems such that R_1 is left-linear, R_2 is right-linear and there is no overlap between left-hand sides of R_1 and right-hand sides of R_2, then $R_1 \cup R_2$ terminates.*

8.6 Conclusion

A question left open in this chapter, when the disjointness property is relaxed, is how to check properties like commutation or quasi-commutation of abstract reduction systems. In the context of term rewrite systems, sufficient conditions have been proposed, based on linearity properties of rewrite rules and on checks of critical pairs for instance by [Ges90, RV80, Toy88]. An overview of their results can be found in [Mid90].

Another research area bears some similarity with the modularity results presented in this chapter. This is the problem of proving termination of systems obtained as the combination of convergent rewrite systems with the (typed) lambda-calculus or with the calculus of constructions. These problems are studied for instance in [GBT89, Oka89, Bar90].

Chapter 9

Implementing rewriting

9.1 Compiling rewriting

9.1.1 Sequentiality

9.1.2 Compilation into a functional language

9.2 Concurrent rewriting

Part III

Solving

Solving equations is ubiquitous in mathematics and the sciences. Solving equations on first-order terms emerged with Herbrand's work on proof theory [Her30] and was coined unification by Alan Robinson [Rob65]. Unification has received considerable attention since then, because it is at the heart of mechanizing mathematics and consequently a major piece in interpretors for logic programming languages. Invented by Alan Robinson, resolution was the first really effective mechanization of first-order logic. For the case of propositional logic, ground resolution allows to deduce the ground clause $C \vee D$ from two ground clauses $A \vee C$ and $\neg A \vee D$, where A denotes an atom and C and D denote ground clauses. The case of non-ground clauses is more difficult, since different atoms may have common ground instances. Unification bridges the conceptual gap between ground and non-ground atoms by computing a representative of all their common instances. Alan Robinson gave the first algorithm ever for computing such a representative, called it a most general unifier, and showed its uniqueness (up to an equivalence).

Following Alan Robinson, many people sought new, more efficient unification algorithms. Milestones here are [CB83, PW78, Hue76, MM82]. Corbin and Bidoit emphasized that the exponential complexity in Robinson's algorithm was due to the implementation of terms by trees, a fact already known by many people, including Robinson himself.[1] Simply using dags instead broke the complexity down to a quadratic one. Huet obtained an almost linear complexity by using Tarjan's Find-Union algorithm for managing equivalence classes (of unifiable subterms). Paterson and Wegman found a truly linear one by using a more elaborate data structure for terms. Its use, however, does not pay off in practice because of the important overhead of building the data structure itself. Finally, Martelli and Montanari pointed out that unification should be seen as a problem of solving equations. Their algorithm incorporates the detection of cycles along the way (rather than at the end) and performs well in practice. Some of these algorithms adapt to infinite rational terms [Hue76, Col84]. They can all be seen as deterministic implementations of a same set of non-deterministic rules, as recently sketched in [DJ90a]. Various data structures may be used to ease the implementation of the rules. We build on this point of view borrowed from Martelli and Montanari in section 3.2. Basic properties of unifiers are discussed by Lassez, Maher and Marriot [LMM88], and the logic of unification by Le Chenadec [LC89].

Attempts for mechanizing higher-order logic started soon after the discovery of resolution. If deduction can still be performed [Hue72], solving equations between higher-order terms, a problem called higher-order unification, becomes undecidable (actually semi-decidable) even for second-order terms. The main difference between the first-order and the higher-order case is the possibility of having infinitely many incomparable solutions. Milestones here are [Pie71, Hue76, SG89]. Huet proved undecidability of third-order unification (a result refined by Goldfarb who proved that second-order unification was already undecidable [Gol81]) and gave a fair procedure to enumerate a set of generators (called *preunifiers*) of all solutions in the general case.

In 1972 Plotkin gave a set of inference rules for mechanizing first-order theories based on *E-unification*, where two terms unify if they have a common instance, common up to E-equality [Plo72]. Similar to the higher-order case, he introduced the notion of *complete sets of E-unifiers*, that is generators of the whole set of unifiers by using instanciation and E-equality. He himself gave an algorithm for arithmetic theories, showed the existence of infinitely many such most general unifiers for associative theories, and conjectured that complete sets of most general unifiers may not always exist (intuitively, most general unifiers may be infinite terms). Milestones here are [Baa89a, FH86, Mak77, Yel87, TA87, BJSS88, Kir86, Fag84, Kir89a, Sti81, SS89]. Fages and Huet proved Plotkin's conjecture and showed that minimal complete sets were isomorphic. Baader later gave sufficient conditions for a minimal complete sets of unifiers not to exist. Algorithms for Presburger arithmetic were given first by Presburger and later by Shostak.[2] Stickel solved the AC-case, whose termination proof was much later completed by Fages. AC-unification relies upon solving linear homogeneous diophantine equations, a problem successfully addressed by Huet [Hue78], Lambert [Lam87a] and Fortenbacher [For83]. Stickel's algorithm was recently improved upon first by Kirchner [Kir89a] and then Boudet [Bou89]. Both algorithms need solving systems of linear homogeneous diophantine equations, a problem solved efficiently by Contejean and Devie [CD89]. Makanin solved the case of associativity and identity (words). A detailed study of Makanin's algorithm can be found in [Péc81, Abd87]. Arnborg and Tiden solved the case of left (or right) distributivity. Boudet, Jouannaud and Schmidt-Schauß solved the case of Boolean rings and Abelian groups. Kirchner solved the case of "syntactic" theories by generalizing Martelli and Montanari. Schmidt-Schauß solved the general case of a combination of two disjoint theories that separately admit an unification algorithm, improving upon Yelick and Kirchner.

On the dark side, the solvability of Diophantine equations, that is, polynomial equations over the integers, was shown to be undecidable by Matijasevič [Mat70, DMR76].[3] Simpler cases of undecidable unification

[1]See Knight's unification survey [Kni89] for interesting historical remarks about the discovery of improved unification algorithms.

[2]Presburger arithmetic can be equationally axiomatized, which justifies its place in this list.

[3]An axiomatization of the corresponding equational theory is given in section 10.1.

problems are associativity and distributivity, and associativity, commutativity and distributivity [Sza82, SS82b]. Distributivity (left and right) has been a challenging simple equational theory whose unification problem was unknown until Schmidt-Schauß solve it positively [SS98].

Of course, E-unifiability is semi-decidable for recursively enumerable E. Paramodulation (without functional reflexivity axioms) is one improvement over the obvious "British-museum" method of interleaving the production of substitutions with the search for equational proofs. Enumerating a complete set of unifiers in an arbitrary equational theory is called *universal unification* by Siekmann [Sie89]. Fay [Fay79] was the first to give a non-trivial (non necessarily terminating) procedure for universal unification, called *narrowing*, and to prove its completeness for any theory presented by a finite convergent set of rewrite rules. Hullot [Hul80a] improved upon Fay by characterizing cases where a modified procedure terminates (an example is given by the theory of quasi-groups). A rule-based version of narrowing (Fay's version) is due to Nutt, Réty and Smolka [NRS89]. Gallier and Snyder [GS89] solved the general case, giving a set of rules enumerating complete set of E-unifiers for an arbitrary theory E.

Universal unification has been proved useful for embedding equations into PROLOG [GM86], as well as for the general understanding of E-unification itself.

Type systems introduced new unification problems in the last five years. The possibility of having subtypes, record types, overloading and polymorphism raised similar questions as above, since part of these new features can be equationally axiomatized. This is one area where the research is very active now, since many problems are either open or amenable to drastic improvements. Major works here are [CD85, Wal84, SS86a, MGS87, SAK89, Kir88].

With constraint logic programming [Col89, JL87] and, more recently, constrained theorem proving [KK89], unification had to be generalized so as to cover inequalities besides equalities, as well as arbitrary quantifiers and special predicate symbols (besides equality). This is again a subject under active development, and a whole paper is devoted to it by Comon [Com91b].[4]

Previous surveys on unification include [Sie89, Kni89]. Recent developments in unification theory are given in [Kir90]. Our notations and definitions are consistant with [DJ90a], where a whole section is devoted to unification. We concentrate on unification in this survey, although related problems such as matching [Bür89], semi-unification [Hen89] and rigid unification [GRS89] share many properties with unification.

Higher-Order Unification

Higher-order unification is the problem of solving equations in the typed lambda calculus, i.e., of computing higher-order substitutions for the free variables of the terms of the equation $s =^? t$, which make s and t equal modulo the (α and β) conversion rules of the calculus. This problem is, of course, related to semantic first-order unification.

Higher-order unification is the key to higher-order theorem proving and higher-order logic programming. As an example, it is the main mechanism for generalizing resolution to higher-order logic [Hue72, Hue73a, JP76]. It is also crucial for program synthesis and program transformation [HM88, HL78a] or type inference in polymorphic languages [Pfe88, LC89]. It is the basic mechanism used in λ-PROLOG [MN86].

The main results on higher-order unification have been obtained during the seventies. Huet [Hue76, Hue73b, Hue75] has shown the undecidability of third order unification. He has also defined a restricted form of unification called pre-unification which is used in refutational methods for higher-order theorem proving. Goldfarb [Gol81] shows that even second-order unification is undecidable, and Farmer has recently proved that monadic second-order unification was decidable [Far88]. Let us also mention the early work by Pietrzykowski and Jensen [Pie71, JP76].

Recently Snyder and Gallier gave a general overview of the work in this field and showed how Huet's procedure can be described and proved using transformation rules [SG89]. This highlights similarities and differences between first- and higher-order unification by expressing the fundamental elementary unification steps in the same way.

Higher-order unification is surveyed in [GS90, JK91]. Taking advantage of the transformation of λ-terms into combinators, Dougherty [Dou90] gives an alternative to Huet's approach.

[4]The search for decision procedures for various constraint languages actually goes back far away in the past. For example, Fourier investigated linear constraints on the reals.

Chapter 10

Unification of equational problems

We will now define what equational unification is all about. But to give such a definition is not so easy since it should be simple and allow convincing soundness and completeness proofs. This is not the case with many of the definitions of equational unification which are given in the litterature. The reason is that there are two contradicting goals: as far as generality is concerned, the definition should not depend upon a particular case, such as syntactic unification. On the other hand, since our approach is based on transforming problems in simpler ones having the same set of solutions, the definition must allow to get a clear, simple and robust definition of equivalence of two unification problems but also to be able to define what simpler problem means.

10.1 Solutions and unifiers

One source of potential trouble in defining equational problems and their solutions and unifiers is that some variables may simply appear or go away when transforming a given unification problem into a simpler one. As a consequence, our definition of a solution to a unification problem should not only care of the variables occurring in the simplified problem, but also of the variables which have appeared at the intermediate steps. The idea that the so-called "new variables" are simply existentially quantified variables appeared first in Comon [Com88b] although quantifiers had already been introduced by Kirchner and Lescanne [KL87] both in the more general context of disunification.

An other problem, is the need of managing the so-called "new variables" which may be necessary for expressing the most general unifiers. Building upon Plotkin [Plo72] and Huet [Hue76], Hullot [Hul80c] introduced the concept of variables *away from* a set given set of variables $\mathcal{W}$. Intuitively, $\mathcal{W}$ is a set of variables which must be *protected* from a possible capture because of the current environment in which unification is used e.g. a completion procedure.

Since we are giving a general framework for unification problems, we need to define the solutions of an equation between terms in $\mathcal{T}(\mathcal{F},\mathcal{X})$ in an arbitrary $\mathcal{F}$-algebra $\mathcal{A}$ which will be in our case the free algebra modulo some set of equational axioms. The solution to a unification problem in some algebra $\mathcal{A}$ should not, of course, give a value to the bound variables, the value of which should be given by the semantics of existential quantification. This can easily be done, if solutions are homomorphisms, by restricting the application of the homomorphism to the free variables.

Definition 10.1 Let $\mathcal{F}$ be a set of function symbols, $\mathcal{X}$ be a set of variables, and $\mathcal{A}$ be an $\mathcal{F}$-algebra. A $\langle\mathcal{F},\mathcal{X},\mathcal{A}\rangle$*-unification problem* (*unification problem* for short) is a first-order formula without negations nor universal quantifiers whose atoms are $\mathbf{T},\mathbf{F}$ and $s =^?_{\mathcal{A}} t$, where s and t are terms in $\mathcal{T}(\mathcal{F},\mathcal{X})$. We call an *equation* on $\mathcal{A}$ any $\langle\mathcal{F},\mathcal{X},\mathcal{A}\rangle$-unification problem $s =^?_{\mathcal{A}} t$ and *multiequation* any multiset of terms in $\mathcal{T}(\mathcal{F},\mathcal{X})$.

Equational problems will be written as a disjunction of existentially quantified conjunctions:

$$\bigvee_{j\in J} \exists \overrightarrow{w_j} \bigwedge_{i\in I_j} s_i =^?_{\mathcal{A}} t_i.$$

When $|J| = 1$ the problem is called a *system.* Variables $\overrightarrow{w}$ in a system $P = \exists\overrightarrow{w} \bigwedge_{i\in I} s_i =^?_{\mathcal{A}} t_i$ are called *bound*, while the other variables are called *free.* Their respective sets are denoted by $\mathcal{BV}ar(P)$ and $\mathcal{V}ar(P)$.

The superscripted question mark is used to make clear that we want to solve the corresponding equalities, rather than to prove them.

$x+0$	$\rightarrow$	x	$0+x$	$\rightarrow$	x
$x*0$	$\rightarrow$	0	$0*x$	$\rightarrow$	0
$\mathit{prede}(\mathit{succe}(x))$	$\rightarrow$	x	$\mathit{succe}(\mathit{prede}(x))$	$\rightarrow$	x
$\mathit{opp}(0)$	$\rightarrow$	0	$\mathit{opp}(\mathit{opp}(x))$	$\rightarrow$	x
$x+\mathit{opp}(x)$	$\rightarrow$	0	$\mathit{opp}(x)+x$	$\rightarrow$	0
$\mathit{opp}(\mathit{prede}(x))$	$\rightarrow$	$\mathit{succe}(\mathit{opp}(x))$	$\mathit{opp}(\mathit{succe}(x))$	$\rightarrow$	$\mathit{prede}(\mathit{opp}(x))$
$\mathit{succe}(x)+y$	$\rightarrow$	$\mathit{succe}(x+y)$	$x+\mathit{succe}(y)$	$\rightarrow$	$\mathit{succe}(x+y)$
$x+\mathit{prede}(y)$	$\rightarrow$	$\mathit{prede}(x+y)$	$\mathit{prede}(x)+y$	$\rightarrow$	$\mathit{prede}(x+y)$
$\mathit{opp}(x+y)$	$\rightarrow$	$\mathit{opp}(y)+\mathit{opp}(x)$	$x*\mathit{succe}(y)$	$\rightarrow$	$(x*y)+x$
$\mathit{succe}(x)*y$	$\rightarrow$	$y+(x*y)$	$(x+y)+z$	$\rightarrow$	$x+(y+z)$
$\mathit{opp}(y)+(y+z)$	$\rightarrow$	z	$x+(\mathit{opp}(x)+z)$	$\rightarrow$	z
$\mathit{prede}(x)*y$	$\rightarrow$	$\mathit{opp}(y)+(x*y)$	$x*\mathit{prede}(y)$	$\rightarrow$	$(x*y)+\mathit{opp}(x)$

Figure 10.1: BasicArithmetic: Basic arithmetic axioms.

Example 10.1 With obvious sets $\mathcal{X}$ and $\mathcal{F}$ and for an $\mathcal{F}$-algebra $\mathcal{A}$, $\{\exists z\, f(x,a) =^?_{\mathcal{A}} g(f(x,y), g(z,a)) \wedge x =^?_{\mathcal{A}} z\}$ is a system of equations where the only bound variable is z and the free variables are x and y.

Definition 10.2 A $\mathcal{A}$-*solution* (for short a solution when $\mathcal{A}$ is clear from the context) to a $\langle\mathcal{F},\mathcal{X},\mathcal{A}\rangle$-unification system $P = \exists\overrightarrow{w}\ \bigwedge_{i\in I} s_i =^?_{\mathcal{A}} t_i$ is a homomorphism h from $\mathcal{T}(\mathcal{F},\mathcal{X})$ to $\mathcal{A}$ such that there exists an homomorphism h' from $\mathcal{T}(\mathcal{F},\mathcal{X})$ to $\mathcal{A}$ satisfying $h'(x) = h(x)$ for all $x \in \mathcal{X} - \overrightarrow{w}$ and $h'(s_i) = h'(t_i)$ for all $i \in I$.

A $\mathcal{A}$-solution to a $\langle\mathcal{F},\mathcal{X},\mathcal{A}\rangle$-unification problem $D = \bigvee_{j\in J} P_j$, where all the P_j are unification systems, is a homomorphism h from $\mathcal{T}(\mathcal{F},\mathcal{X})$ to $\mathcal{A}$ such that h is solution of at least one of the P_j.

We denote by $\mathcal{S}ol_{\mathcal{A}}(D)$ the set of solutions of D in the algebra $\mathcal{A}$. Two $\langle\mathcal{F},\mathcal{X},\mathcal{A}\rangle$-unification problems D and D' are said to be $\mathcal{A}$-*equivalent* if $\mathcal{S}ol_{\mathcal{A}}(D) = \mathcal{S}ol_{\mathcal{A}}(D')$, this is denoted $D' \Leftrightarrow_{\mathcal{A}} D$.

Note how the above definition takes care of the existentially quantified variables by forgetting the respective values of the homomorphism h, allowing these variables to take whatever value is necessary to satisfy all equations.

Finding solutions to a unification problem in an arbitrary $\mathcal{F}$-algebra $\mathcal{A}$ is impossible in general and when it is possible it is often difficult. For example, solving equations in the algebra $\mathcal{T}(\mathcal{F})/E$, where E is the BasicArithmetic theory given by the set of equational axioms described in Figure 10.1 is actually the problem of finding integer solutions to polynomial equations with integer coefficients. This is known as Hilbert's tenth problem, shown to be undecidable by Matijasevič [Mat70, DMR76].

Fortunately, we will see that the existence of solutions is decidable for many algebras of practical interest. However, there are in general infinitely many solutions to a unification system P. A first step towards the construction of a finite representation of these solutions is the notion of a unifier, which is meant as describing sets of solutions:

Definition 10.3 A $\mathcal{A}$-*unifier* of an $\langle\mathcal{F},\mathcal{X},\mathcal{A}\rangle$-unification system

$$P = \exists\overrightarrow{w}\ \bigwedge_{i\in I} s_i =^?_{\mathcal{A}} t_i$$

is a substitution σ such that

$$\mathcal{A} \models \exists\overrightarrow{w}\ \bigwedge_{i\in I} \sigma_{|\mathcal{X}-\overrightarrow{w}}(s_i) = \sigma_{|\mathcal{X}-\overrightarrow{w}}(t_i).$$

A $\mathcal{A}$-unifier of a $\langle\mathcal{F},\mathcal{X},\mathcal{A}\rangle$-unification problem $D = \bigvee_{j\in J} P_j$, where all the P_j are $\langle\mathcal{F},\mathcal{X},\mathcal{A}\rangle$-unification systems, is a substitution σ such that σ unifies at least one of the P_j.

We denote by $\mathcal{U}_{\mathcal{A}}(D)$ the set of unifiers of D. This is abbreviated $\mathcal{U}(D)$ when $\mathcal{A}$ is clear from the context. Similarly when clear from the context $\mathcal{A}$-unifiers are called unifiers and $\langle\mathcal{F},\mathcal{X},\mathcal{A}\rangle$-unification is called unification.

It is important to note that, as it is usual in logic (see for example [Gal86]), only free variables are substituted. Thus for example, applying the substitution $\{(z \mapsto f(a))\}$ to the formula $\exists z,\ f(z) =^? x$ results in this formula itself. Bound variables are taken care of by the semantics as defined in Definition 2.22 and indeed, unifiers are substitutions of the free variables of the formulas in such a way that it becomes valid as a universaly quantified existential formula.

The above definition is important, since it allows to define unifiers for equations whose solutions range over an arbitrary $\mathcal{F}$-algebra $\mathcal{A}$. The relationship between solutions and unifiers is not, however, quite as strong as we would like it to be. Of course, interpreting a unifier in the algebra $\mathcal{A}$ by applying an arbitrary homomorphism yields an homomorphism that is a solution. That all solutions are actually homomorphic images of unifiers is not true in general, but happens to be true in term generated algebras (allowing in particular free algebras generated by a given set of variables).

To illustrate the difference between solutions and unifiers, let us consider the set of symbols $\mathcal{F} = \{0, s, *\}$ and the $\mathcal{F}$-algebra $\mathbf{R}$ whose domain is the set of real numbers. Then $h(x) = \sqrt{2}$ is a solution of the equation $x * x =^?_{\mathbf{R}} s(s(0))$, although no $\mathbf{R}$-unifier exists for this equation, since the square root cannot be expressed in the syntax.

Proposition 10.1 If $\mathcal{A}$ is a term generated $\mathcal{F}$-algebra then a unification system P has $\mathcal{A}$-solutions iff it admits $\mathcal{A}$-unifiers.

Proof: Assume that σ is a $\mathcal{A}$-unifier of the system $P = \exists \overrightarrow{w}\ \bigwedge_{i \in I} s_i =^?_{\mathcal{A}} t_i$ then by definition

$$\mathcal{A} \models \exists \overrightarrow{w}\ \sigma_{|\mathcal{X} - \overrightarrow{w}}(s_i) = \sigma_{|\mathcal{X} - \overrightarrow{w}}(t_i)$$

(where the universal quantifier is not explicitly mentioned). This means, by definition of validity, that for every assignment ν of the variables in $\mathcal{X} \setminus \overrightarrow{w}$, $\nu.\sigma$ is a $\mathcal{A}$-solution of P.

Conversely, if μ is a $\mathcal{A}$-solution of the system $P = \exists \overrightarrow{w}\ \bigwedge_{i \in I} s_i =^?_{\mathcal{A}} t_i$, there exists homomorphisms h and h' from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ to $\mathcal{A}$ such that for all $i \in I$, $h'(s_i) = h'(t_i)$ and $\forall x \in \mathcal{X} \setminus \overrightarrow{w}$, $h(x) = h'(x)$. Since $\mathcal{A}$ is term generated (c.f. Definition 2.23), this means that: $\forall a \in \mathcal{A}, \exists t \in \mathcal{T}(\mathcal{F})$ such that, $a = \iota(t)$ where ι is the interpretation under consideration for $\mathcal{A}$. Thus $\forall x \in \mathcal{D}om(h') \cap \mathcal{X}, \exists u_x \in \mathcal{T}(\mathcal{F})$ such that $h'(x) = \iota(u_x)$. Let σ be the ground substitution such that for all $x \in \mathcal{D}om(h') \cap \mathcal{X}$, $\sigma : x \mapsto u_x$. Then by construction: $\iota\sigma(s_i) = \iota\sigma(t_i)$, which means since these equality involves only ground terms that, $\mathcal{A} \models \sigma(s_i) = \sigma(t_i)$ for all i in I, and thus σ is a $\mathcal{A}$-unifier of P. $\Box$

Notice that from the previous proof it becomes clear that unifiers represent in general, because of variable instantiation, several (possibly infinite) solutions.

Example 10.2 In the BasicArithmetic example, if we consider the equation $x =^? s(y)$, then $(x \mapsto 1, y \mapsto 2)$ is one of its $\mathbf{N}$-solution. Its corresponding $\mathbf{N}$-unifier is $(x \mapsto \mathit{succe}(0), y \mapsto \mathit{succe}(\mathit{succe}(0)))$. The $\mathbf{N}$-unifier $(x \mapsto \mathit{succe}(y))$ also represents the previous $\mathbf{N}$-solution by simply valuating x to $\mathit{succe}(0)$ and y to $\mathit{succe}(\mathit{succe}(0))$.

In the following, we are restricting our attention the special but fundamental case of the free algebras, initial algebras and their quotient. For these algebras the above property is satisfied since they are by construction term generated. As an important consequence of the previous proposition, two unification problems are equivalent iff they have the same sets of unifiers, an alternative definition that we are adopting in the remainder.

Definition 10.4 For a set of equational axioms E built on terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, an equation to be solved in $\mathcal{A} = \mathcal{T}(\mathcal{F}, \mathcal{X})/E$ is denoted by $=^?_E$. The equivalence of equational problems is denoted by $\Leftrightarrow_E$ and $\mathcal{A}$-unification is called *E-unification*.

Since a unification system is a term whose outermost symbol is the existential quantifier, its body part, i.e., the conjunction of equations occurring in the system, is actually a subterm. Rewriting this subterm to transform the unification system into a simpler one hides the existential quantifier away. As a consequence, one may simply forget about the existential quantifier for most practical purposes. This is true for syntactic unification, where existential quantification is not needed as long as the transformation rules are chosen among the set given in Chapter 3. This is not, however, true of all algorithms for syntactic unification: Huet [Hue76] for example uses an abstraction rule introducing new variables in a way similar to the one used for combination problems in Chapter 11.

Example 10.3 The equation $\{x + y =^?_{\mathcal{A}} x + a\}$ is equivalent to the equation $\{y =^?_{\mathcal{A}} a\}$, since the value of x is not relevant: our definition allows dropping useless variables. Note that this is not the case if unifiers are substitutions whose domain is restricted to the variables in the problem, a definition sometimes used in the litterature, since $\{x \mapsto a, y \mapsto a\}$ would be a unifier of the first problem but not of the second.

Example 10.4 $P = \{x + (y * y) =^?_{\mathcal{A}} x + x\}$ is equivalent to $P' = \exists z\ \{x + z =^?_{\mathcal{A}} x + x, z =^?_{\mathcal{A}} y * y\}$ whereas it is not equivalent to $P'' = \{x + z =^?_{\mathcal{A}} x + x, z =^?_{\mathcal{A}} y * y\}$. In P'', z does not get an arbitrary value, whereas it may get any value in P, and in P' as well, since the substitution cannot be applied to the bound variable z.

Exercice 40 — Assuming the symbol + commutative, prove that $x + f(a, y) =^? g(y, b) + f(a, f(a, b))$ is equivalent to $(x =^? g(y, b) \wedge f(a, y) =^? f(a, f(a, b))) \vee (x =^? f(a, f(a, b)) \wedge f(a, y) =^? g(y, b))$.
Answer: One can easy check that the definition of equivalence (on unifiers) is satisfied.

The key to our definition is that, as usual in logic, substitutions do not replace bound variables. This question of handling "new" variables appearing during the unification process has often been ignored. When recognized, it has been solved in different ways. An alternative, slightly more complicated definition of solutions is given in [Com88b]. Another alternative to the solution developed here is given in [Kir85a], where a weaker notion than equivalence is defined, called extension, which allows one to handle the case of different sets of variables in related unification problems. This leads, however, to more complex and technical proofs. A third alternative [RB86, RB85, Gog89, RS86, Baa89b, Baa91] is simply to get rid of the variables by using a categorical point of view. However, this does not allow to build unification algorithms directly from their description, a major goal in our approach.

Let us finally mention that for any finitely presented equational theory $\mathcal{TH}(E)$, the set of E-unifiers is recursively enumerable. This is due to the fact that one can enumerate all the substitutions and for each of them check, in finite time, if it is an E-unifier (see Section 2.4.6).

10.2 Generating sets

We have seen that unifiers schematize solutions in the case of term generated algebras. Let us now consider schematizing sets of unifiers. This will be done using the notion of complete set of unifiers that we will also define and then study from an abstract point of view.

10.2.1 Complete sets of unifiers

Unifiers are representations of solutions but are still infinitely many in general. We may take advantage of the fact that any instance of a unifier is itself a unifier to keep a set of unifiers minimal with respect to instantiation.

In order to express it in general, we need to introduce a slightly more general concept of equality and subsumption as follows.

Definition 10.5 Let $\mathcal{A}$ be an $\mathcal{F}$-algebra. We say that two terms s and t are $\mathcal{A}$-*equal*, written $s =_{\mathcal{A}} t$ if $h(s) = h(t)$ for all homomorphisms h from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ into $\mathcal{A}$. A term t is an $\mathcal{A}$-*instance* of a term s, or s is *more general* than t in the algebra $\mathcal{A}$ if $t =_{\mathcal{A}} \sigma(s)$ for some substitution σ; in that case we write $s \leq_{\mathcal{A}} t$. The relation $\leq_{\mathcal{A}}$ is a quasi-ordering on terms called $\mathcal{A}$-*subsumption*.

As in Section 2.4.4 subsumption is easily lifted to substitutions:

Definition 10.6 We say that two substitutions σ and τ are $\mathcal{A}$-*equal* on the set of variables $V \subseteq \mathcal{X}$, written $\sigma =^V_{\mathcal{A}} \tau$ if $\sigma(x) =_{\mathcal{A}} \tau(x)$ for all variables x in V. A substitution σ is *more general* for the algebra $\mathcal{A}$ on the set of variable V than a substitution τ, written $\sigma \leq^V_{\mathcal{A}} \tau$, if there exists a substitution ρ such that $\rho\sigma =^V_{\mathcal{A}} \tau$. The relation $\leq^V_{\mathcal{A}}$ is a quasi-ordering on substitutions called $\mathcal{A}$-*subsumption*. V is omitted when equal to $\mathcal{X}$.

The above definitions specialize to the notion of subsumption defined in Section 2.4.4 when the algebra $\mathcal{A}$ is equal to $\mathcal{T}(\mathcal{F}, \mathcal{X})$.

We are in fact mostly interested in generating sets of the set of $\mathcal{A}$-unifiers of an equational problem. The definition of such a generating set that we are now presenting is issued from the definition given first by G. Plotkin [Plo72] followed by G. Huet [Hue76] and J.-M. Hullot [Hul80c].

Definition 10.7 Given an equational problem P, $CSU_{\mathcal{A}}(P)$ is a *complete set of unifiers* of P for the algebra $\mathcal{A}$ if:

(i) $CSU_{\mathcal{A}}(P) \subseteq \mathcal{U}_{\mathcal{A}}(P)$, (correctness)

(ii) $\forall \theta \in \mathcal{U}_{\mathcal{A}}(P)$, $\exists \sigma \in CSU_{\mathcal{A}}(P)$ such that $\sigma \leq^{\mathcal{V}ar(P)}_{\mathcal{A}} \theta$, (completeness)

(iii) $\forall \sigma \in CSU_{\mathcal{A}}(P), \mathcal{R}an(\sigma) \cap \mathcal{D}om(\sigma) = \emptyset$. (idempotency)

$CSU_{\mathcal{A}}(P)$ is called a *complete set of most general unifiers* of P in $\mathcal{A}$, and written $CSMGU_{\mathcal{A}}(P)$, if:

(iv) $\forall \alpha, \beta \in CSMGU_{\mathcal{A}}(P), \alpha \leq_{\mathcal{A}}^{\mathcal{V}ar(P)} \beta$ implies $\alpha = \beta$. (minimality)

or in other words: any two substitutions of $CSU_{\mathcal{A}}(P)$ are not comparable for the quasi-ordering $\leq_{\mathcal{A}}^{\mathcal{V}ar(P)}$. Furthermore $CSU_{\mathcal{A}}(P)$ is said *outside the set of variables* W such that $\mathcal{V}ar(P) \subseteq W$ when:

(v) $\forall \sigma \in CSU_{\mathcal{A}}(P), \mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(P)$ and $\mathcal{R}an(\sigma) \cap W = \emptyset$. (protection)

Notice that unifiers are compared only on the problem variables (i.e., $\mathcal{V}ar(P)$), a fundamental restriction as pointed out in particular by F. Baader in [Baa91]. As shown in Lemma 2.1 the conditions (idempotency) as well as (protection) in the above definition insure that the unifiers are idempotent.

One may wonder why the notion of being most general is defined with respect to unifiers, rather than to unified terms, since practice often uses the latter. This is so because substitutions compose, making the design of algorithms easier. On the other hand, different unifiers in a complete set of most general unifiers may yield the same set of instantiated terms, up to $\mathcal{A}$-equivalence. This is a practical concern that one should keep in mind.

Complete sets of most general unifiers are very important, in particular when they are finite, since they describe the generally infinite set of solutions of the problem under consideration. We will further elaborate on this notion in the next sections.

In the next chapters we mainly investigate *semantic unification* (also called *equational unification*), when $\mathcal{A}$ is the quotient algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})/E$, for some a priori given set E of equational axioms, or the typed instance of these algebras.

Although *higher-order unification* could be viewed as a case of semantic unification in a quotient set by appropriately axiomatizing the λ-calculus, we can adopt the traditional λ-calculus view that unifiers are simply higher-order substitutions. In this case, $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the algebra of higher-order terms, and $\mathcal{A}$ its quotient under $\beta\eta$-interreducibility. The above definition of complete sets of unifiers hold for all these cases but we will not investigate further higher-order unification in this book.

Exercice 41 — Give a description of all the $\emptyset$-solutions of the equation $x =^? y$. Then give a complete set of $\emptyset$-solutions outside $\{x, y\}$. How does the elements of this set compare to the unifier $\alpha = (x \mapsto y)$? Is your complete set of solutions minimal?

Answer: 1. $\{\sigma | \sigma(x) = \sigma(y) = t, t \in \mathcal{T}(\mathcal{F}, \mathcal{X}) - \{x, y\}\} \cup \{x \mapsto y\} \cup \{y \mapsto x\}$ 2. It is enough to take $\sigma = \{(x \mapsto z), (y \mapsto z)\}$, where z is a variable different from$x$ et y. We have then $(z \mapsto y).\sigma = \alpha[\{x, y\}]$.

A complete set of $\mathcal{A}$-unifiers satisfying (i) and (ii) clearly always exists, just take the set of all $\mathcal{A}$-solutions. Let us now check that the limitation on the set of variables to get complete set of unifiers outside W is not in general a limitation. This will imply that the idempotency restriction is not either a limitation in general.

Proposition 10.2 Let W be a set of variables such that $X - W$ is denumerable and P an equational problem such that $\mathcal{V}ar(P) \subseteq W$. If Σ is a complete set of $\mathcal{A}$-unifiers of P satisfying only the conditions (i) and (ii) of the definition above then there exists a complete set of unifiers of P outside W having the same number of elements than Σ.

Proof: Let

$$\Sigma' = \{\xi\delta \mid \delta \in \Sigma \text{ and } \xi \in Perm \text{ and } \xi : W \cap \mathcal{R}an(\delta) \mapsto X - (W \cup \mathcal{R}an(\delta))\}.$$

In others words, ξ renames the variables of $W \cap \mathcal{R}an(\delta)$ outside $W \cup \mathcal{R}an(\delta)$. Note that this is possible because $X - W$ has been supposed denumerable and since $\mathcal{R}an(\delta)$ is finite. Let us check that Σ' is a complete set of unifiers:

1. By definition, each element of Σ' is outside W.
2. It is clear that $\Sigma' \in \mathcal{U}_{\mathcal{A}}(P)$.
3. Finally let us show the completeness of Σ' : $\forall \alpha \in \mathcal{U}_{\mathcal{A}}(P), \exists \delta \in \Sigma$ such that

$$\begin{array}{ll} & \delta \leq_{\mathcal{A}}^{\mathcal{V}ar(P)} \alpha \\ \Leftrightarrow & \exists \rho, \rho\delta =_{\mathcal{A}}^{\mathcal{V}ar(P)} \alpha \\ \Leftrightarrow & \exists \rho, \rho\xi^{-1}\xi\delta =_{\mathcal{A}}^{\mathcal{V}ar(P)} \alpha \\ \Leftrightarrow & \xi\delta \leq_{\mathcal{A}}^{\mathcal{V}ar(P)} \alpha \end{array}$$

which prove that Σ' satisfy all the conditions to be a complete set of unifiers outside W.

□

This result shows that under the natural condition of having enough variables, there exists always a complete set of unifiers outside a given set of variables containing the variables of the problem. The condition of protection for the variables in W is thus only a technical way to simplify the proofs. A consequence of the previous result is also that if there is enough variables, it is not necessary to check the idempotency condition in Definition 10.7, since it can be insured as in the previous proof.

Using the result above, let us now show how a complete set of unifiers can be computed for a disjunction of systems from the complete sets of its system components.

Lemma 10.1 Let $D = \bigvee_{j\in J} P_j$ a disjunction of systems. Then $\bigcup_{j\in J} CSU_{\mathcal{A}}(P_j)$ is a complete set of $\mathcal{A}$-unifiers of D.

Proof: We just have to check properties (i) and (ii).

(i), by definition of the set of solutions of a disjunction we have: $\bigcup_{j\in J} CSU_{\mathcal{A}}(P_j) \subseteq \mathcal{U}_{\mathcal{A}}(D)$.

(ii): Let σ be a $\mathcal{A}$-unifier of D. By definition there exists a system P_j which solutions set contains σ and this allows to conclude that σ will be generated by an element of $CSU_{\mathcal{A}}(P_j)$. □

Because of this result, we will be interested mainly in solving systems rather that disjunctions.

Unfortunately this cannot be extended to minimal complete sets of unifiers. Take for example: $D = S_1 \vee S_2$ with $S_1 = \{x =^? y\}$ and $S_2 = \{x =^? a \wedge y =^? b\}$. Then the respective minimal complete sets of unifiers are:

$$\begin{aligned} CSMGU(D) &= \{x \mapsto z, y \mapsto z\} \\ CSMGU(S_1) &= \{x \mapsto z, y \mapsto z\} \\ CSMGU(S_2) &= \{x \mapsto a, y \mapsto b\}, \end{aligned}$$

which shows that $CSMGU(D) \neq CSMGU(S_1) \cup CSMGU(S_2)$.

10.2.2 Abstract properties of generating sets

We have just seen how the set of all the unifiers of a given problem can be generated from one of its subset, if possible minimal. In fact many properties of generating sets are only due to the structure of the quasi-order on substitutions and not to the substitutions themself. It is why it is quite natural to explore first the properties of generating sets of posets. The results will then have immediate consequences on $\mathcal{A}$-unifiers up to $\equiv_{\mathcal{A}}$ the subsumption equivalence modulo $\mathcal{A}$. For more details on the concepts that we are now using on orderings, refer to Section 4.2 and to [Bou70] or [BM67, Bir67].

Definition 10.8 A subset G of E is a *minimal generating set* of the poset $(E, \leq)$ if:

1. $\forall \alpha \in E, \exists \sigma \in G$ such that $\sigma \leq \alpha$, (completeness)
2. $\forall \alpha, \beta \in G, \alpha \leq \beta \Rightarrow \alpha = \beta$. (minimality)

A poset $(E, \leq)$ is of *type zero* if E have no minimal generating set.

Exercice 42 — Show that a subset G generating the poset $(E, \leq)$ is minimal iff G have no proper subset that generates E.
Answer:

Lemma 10.2 Let $(E, \leq)$ be a poset. Then two minimal generating subsets G_1 and G_2 of E are equal.

Proof: G_2 a generating set thus:

$$\forall \alpha \in G_1, \exists \sigma \in G_2 \text{ such that } \sigma \leq \alpha,$$

similarly G_1 is a generating set thus:

$$\forall \sigma' \in G_2, \exists \alpha' \in G_1 \text{ such that } \alpha' \leq \sigma'.$$

We have:

$$\forall \alpha \in G_1, \exists \sigma' \in G_2, \alpha' \in G_1, \text{ such that } \alpha' \leq \sigma' \leq \alpha$$

and since G_1 is minimal $\alpha = \alpha'$ and since $(E, \leq)$ is a poset $\alpha = \sigma'$. □

A consequence for minimal complete sets of $\mathcal{A}$-unifiers of a unification problem P is that they are all isomorphic up to $\equiv_{\mathcal{A}}^{\mathcal{V}ar(P)}$ the subsumption equivalence modulo $\mathcal{A}$. For a direct proof, see [FH86].

In fact for posets, the minimal generating sets have a nice characterization:

Lemma 10.3 Let $(E, \leq)$ be a poset and M be the set of all $\leq$-minimal elements of E.

1. If G is a minimal generating set of E then $G = M$.
2. If M is a generating set then M is a minimal generating set of E.

Proof: 1. ($G \subseteq M$): Let G be a minimal generating set of E and let $g \in G$ such that g is not $\leq$-minimal. Then there exists $x \in E$ such that $x < g$. Since G is a generating set, there exists $g' \in G$ such that $g' \leq x < g$, but this contradicts the minimality of G.
($M \subseteq G$): If $m \in M$, since G is a generating set, there exists $g \in G$ such that $g \leq m$. But m being a minimal element implies $m = g$.
2. If M is a generating set, since minimal elements are not comparable by definition, M also minimal. □

As a consequence, we get the following characterization of the type zero posets:

Corollary 10.1 The set M of minimal elements of the poset $(E, \leq)$ is generating iff M is a minimal generating set iff $(E, \leq)$ is not of type zero.

Finally let us state the characterizations due to F. Baader [Baa89a] :

Theorem 10.1 *Let $(E, \leq)$ be a poset. Let us consider the following properties:*

1. *$(E, \leq)$ is of type zero.*
2. *There exists an element w of E such that for all elements u in E, if $u \leq w$ then there exists $v \in E$ such that $v \leq u$ et $v \neq u$.*
3. *There exists a descending chain in E:*
$$\ldots \leq u_3 \leq u_2 \leq u_1$$
without lower bound in E and having the following property:
$$\forall u \in E, u \leq u_n \Rightarrow \exists w \in E \text{ such that } w \leq u \text{ and } w \leq u_n.$$
4. *There exists a strictly decreasing chain in E:*
$$\ldots < u_3 < u_2 < u_1$$
such that $\{u_1, u_2, u_3, \ldots\}$ is a generating set.
5. *There exists a decreasing chain in E without lower bound in E.*

Then the following relations hold:

- $1 \Leftrightarrow 2$
- $4 \Rightarrow 3 \Rightarrow 1 \Rightarrow 5$
- $5 \not\Rightarrow 1 \not\Rightarrow 3 \not\Rightarrow 4$

Proof: Let M be the set of minimal elements in E.
$1 \Rightarrow 2$. By the previous results (1) implies that M is not a generating set. Let w be a term not generated by M. Any u in E such that $u \leq w$ is thus not minimal and there exists v such that $v \leq u$.
$2 \Rightarrow 1$. Assume that M is a generating set. Then for any w there exists u in M such that $u \leq w$. But because of property (2), there exists v in E such that $v \leq u$ and $v \neq u$, which contradicts the minimality of u.
$1 \Rightarrow 5$. Assume (5) false. Then all decreasing chains in E have a lower bound in E. By application of Zorn's lemma (see page 54) there exists a minimal element in E and thus M is a generating set and (1) does not hold.
The proof of the other implications is left to the reader and can be found in [Baa89a]. □

10.2.3 Application to minimal complete sets of unifiers

For an equational theory E and a unification problem P, the previous abstract results on posets can be applied to the poset $(\mathcal{U}_{\mathcal{A}}(P)/\equiv_E^{\mathcal{V}ar(P)}, \leq_E^{\mathcal{V}ar(P)})$, yielding in particular:

Corollary 10.2 When a minimal complete set of E-unifiers of a unification problem P exists, it is unique up to subsumption equivalence. More precisely, for two minimal complete sets of E-unifiers Σ_1 and Σ_2 of P, there exists a bijection ϕ such that:

$$\begin{array}{rcl}\phi : \Sigma_1 & \rightarrow & \Sigma_2 \\ \sigma_1 & \mapsto & \sigma_2 \equiv \phi(\sigma_1).\end{array}$$

The next question is to determine if there exist theories having unification problems P such that $(\mathcal{U}_{\mathcal{A}}(P)/\equiv_E^{\mathcal{V}ar(P)}, \leq_E^{\mathcal{V}ar(P)})$ is of type zero, i.e. such that P has no minimal complete set of E-unifiers. This was conjectured by G.Plotkin in his seminal paper [Plo72] but, the first example of such a situation is due to F. Fages and G. Huet ten years later [FH83].

Proposition 10.3 [FH86] Let FH be the following equational theory:

$$\mathsf{FH} = \left\{ \begin{array}{l} f(0,x) = x \\ g(f(x,y)) = g(y). \end{array}\right.$$

Then the equation $g(x) =^?_{\mathsf{FH}} g(0)$ has no minimal complete set of FH-unifiers.

We will prove this result by applying the case $4 \Rightarrow 1$ in Theorem 10.1. Thus we have to show that there exists a strictly decreasing and generating chain of FH-unifiers. This will be the subject of the next two Lemmas. Before giving them, let us first remark that the term rewriting system $\overrightarrow{\mathsf{FH}} = \{f(0,x) \rightarrow x, g(f(x,y)) \rightarrow g(y)\}$ is terminating and confluent.

Lemma 10.4 Let:

$$\begin{array}{rcl}\sigma_0 & = & \{x \mapsto 0\} \\ \sigma_i & = & \{x \mapsto f(x_i, \sigma_{i-1}(x))\} \quad (0 < i)\end{array}$$

then $\Sigma = \{\sigma_i | i \in \mathbf{N}\}$ is a complete set of FH-unifiers for the equation $g(x) =^?_{\mathsf{FH}} g(0)$ and $\forall i \in \mathbf{N}, \sigma_{i+1} \leq^{\{x\}}_{\mathsf{FH}} \sigma_i$

Proof: We prove the correctness by induction on i.

For $i = 0$, it is clear.

For $0 < i$, we have $\sigma_i(g(x)) = g(\sigma_i(x)) = g(f(x_i, \sigma_{i-1}(x))) =_{\mathsf{FH}} g(\sigma_{i-1}(x)) = \sigma_{i-1}(g(x)) =_{\mathsf{FH}} g(0)$ by induction hypothesis.

Let $\sigma \in \mathcal{U}(g(x) =^?_{\mathsf{FH}} g(0))$, completeness is proved by structural induction on the form of the normal form t of $\sigma(x)$ for $\overrightarrow{\mathsf{FH}}$.

By definition we have $g(t) =_{\mathsf{FH}} g(0)$.

- If t is a variable, a constant or $g(t')$ then clearly t should be 0, in which case $\sigma_0 \leq^{\{x\}}_{\mathsf{FH}} \sigma$.
- If $t = f(t', t'')$ then $g(t'') =_{\mathsf{FH}} g(0)$ and the result follows.

Let us now prove that:

$$\ldots \sigma_i <^{\{x\}}_{\mathsf{FH}} \ldots \sigma_1 <^{\{x\}}_{\mathsf{FH}} \sigma_0$$

- $\sigma_{i+1} \leq^{\{x\}}_{\mathsf{FH}} \sigma_i$ since:

$$\begin{array}{rl}\{x_{i+1} \mapsto 0\}\sigma_{i+1} & = \{x_{i+1} \mapsto 0, x \mapsto f(0, \sigma_i(x))\} \\ & =_{\mathsf{FH}} \{x_{i+1} \mapsto 0, x \mapsto \sigma_i(x)\}\end{array}$$

  thus $\{x_{i+1} \mapsto 0\}\sigma_{i+1} =^{\{x\}}_{\mathsf{FH}} \sigma_i$.
- We can finally show by contradiction that $\sigma_i \not\leq^{\{x\}}_{\mathsf{FH}} \sigma_{i+1}$.

□

Notice that in fact the previous example is a matching problem and that the theory is not regular and collapse. Fages and Huet give also an example of regular collapse theory of type zero. But in this case, the problem under consideration should be a unification problem since:

Proposition 10.4 Let E be a regular set of axioms, then any non-trivialy redondant complete set of matches is minimal.

The FH example relies on building an equational theory such that an equation has for complete set of E-unifiers a strictly decreasing chain of substitutions. More complicated exemples are provided by varieties of idempotent semigroup [Baa87] that are almost all of type zero. In particular idempotent semigroup are of type zero [Baa86a, SS86b]. Together with decidability results, This leads to a unification based classification of equational theories that we are presenting after giving more details on the decidability results for unification.

10.3 (Un)-Decidability of unification

Equational unification and matching are in general undecidable since there exist equational theories that have undecidable word problems.

What is more disturbing is that very simple theories have undecidable E-unification problem. Let us review some of them.

Proposition 10.5 Let DA be the theory built over the set of symbols $\mathcal{F} = \{a, *, +\}$ and consisting in the axioms:

$$\begin{cases} x + (y + z) &= (x + y) + z \\ x * (y + z) &= (x * y) + (y * z) \\ (x + y) * z &= (x * z) + (x * z). \end{cases}$$

Unification is undecidable in DA.

Proof: It is done in [Sza82] using a reduction to Hilbert's tenth problem. □

Another simple theory with undecidable unification problem is D_lAU_r, consisting in the associativity of $+$, the left distributivity of $*$ with respect to $+$ and a right unit element 1 satisfying $x * 1 = x$ [TA87].

In fact, decidability of unification is even quite sensitive to "new" constants. H.-J. Bürckert shows it by encoding the previous DA theory using new constants. This shows in particular that there exists equational theories for which unification is decidable but matching is not [Bür89]. Another quite simple example has been found by A. Bockmayr:

Proposition 10.6 [Boc92] The unification problem for the set of axioms:

$$\mathsf{ISG} = \begin{cases} (x * y) * z &= x * (y * z) \\ i(i(x)) &= x \\ (x * i(x)) * x &= x \\ (i(x) * x) * (i(y) * y) &= (i(y) * y) * (i(x) * x) \end{cases}$$

of *inverse semigroups* is decidable in the signature $\mathcal{F} = \{*, i, a\}$ but is undecidable for $\mathcal{F}' = \{*, i, a, b\}$.

Since left (or right) distributivity has a decidable unification problem [TA87] and as does associativity as well, it is challenging to know which are the minimal undecidable theories consistent with Peano arithmetic. A first answer has been given for DA by Szabó and for D_lAU_r by Arnborg and Tiden: they proved that every theory T consistent with Peano arithmetic and such that $DA \subseteq T$ or $D_lAU_r \subseteq T$ has an undecidable unification problem.

The decidability of unification for classes of theories is also a very challenging problem. For example, variable permutative theories have an undecidable unification problem, as shown in [NO90], refining a result of [SS90b]. Even in theories represented by a canonical term rewriting system (which is, as we have seen in Part Rewriting, a strong requirement) the unification problem is undecidable:

Proposition 10.7 [Boc87]

In the equational theory BasicArithmetic presented by the canonical term rewriting system $\overrightarrow{\mathsf{BasicArithmetic}}$, the unification and matching problems are undecidable.

Proof: (*i*) $\overrightarrow{\mathsf{BasicArithmetic}}$ is confluent and noetherien as it can be checked by a completion program like REVE.

(*ii*) The normal form of any ground term t computed by $\overrightarrow{\mathsf{BasicArithmetic}}$ is the usual arithmetic value i.e. is of the form $p^n(0)$ or $s^m(0)$.

(*iii*) Let us now assume that unification is decidable modulo $\overrightarrow{\mathsf{BasicArithmetic}}$. Then in particular the matchability of any term t to 0 is decidable. We can assume without restriction that the substitution μ such that $\mu(t) = 0$ is ground (otherwise either it is not a match or it can be reduced using $\overrightarrow{\mathsf{BasicArithmetic}}$). Since such a term t is a representation of a multivariate integer polynomial this would mean that Hilbert's tenth problem would be decidable, which has been proven to be false [Mat70, DMR76]. □

10.4 A Classification of Theories with Respect to Unification

Since we have seen that minimal complete sets of E-unifiers are isomorphic whenever they exist, a classification of theories based on their cardinality makes sense, as pioneered by Szabó and Siekmann [SS84, Sza82, SS82b]. But in doing so, we should be careful with the fact that solving one single equation is not general enough a question, as shown by the following result:

Proposition 10.8 There exists equational theories E such that all single equation have minimal complete set of E-unifiers, but some system of equations are of type zero i.e. have no minimal complete set of E-unifiers.

Proof: In [BHSS90] it is shown that the equational theory:

$$\mathsf{BHSS} = \begin{cases} f_1(g_1(x)) & = & g_2(f_1(x)) \\ f_2(g_1(x)) & = & g_2(f_2(x)) \\ f_3(g_1(x)) & = & g_2(f_3(x)) \\ f_4(g_1(x)) & = & g_2(f_4(x)) \\ f_1(k_1(x)) & = & f_2(k_1(x)) \\ f_3(k_2(x)) & = & f_4(k_2(x)) \\ k_1(h(x)) & = & k_2(h(x)) \\ g_1(k_2(h(l(x)))) & = & k_2(h(x)) \end{cases}$$

has the required property. □

Thus it makes sense to define the type of an equational theory based on the cardinality of minimal complete sets of E-unifiers for equation systems, when they exist.

Let P be a system of equations in an equational theory E, and let $CSMGU_E(P)$ be a complete set of most general E-unifiers of P, whenever it exists. E-unification is said to be:

U-based if $CSMGU_E(P)$ exists for all problems P (the class of U-based theories is denoted by $\mathcal{U}$),

U-unitary if $E \in \mathcal{U}$ and $|CSMGU_E(P)| \leq 1$ for all P,

U-finitary if $E \in \mathcal{U}$ and $|CSMGU_E(P)|$ is finite for all P,

U-infinitary if E is U-based but not finitary,

U-nullary if E is not U-based,

U-undecidable if it is undecidable whether a given unification problem has solutions.

We drop the "U-" when it is clear from the context that we are interested in a unification property. We use $0, 1, \omega$ and ∞ to denote the respective types of nullary, unary, finitary and infinitary theories.

Syntactic unification is unitary as we have seen in Section 3.2 and so is unification in boolean rings [MN89], see Section 13.2. Commutative unification is finitary, as we will see next in Section 10.5.4. So is also associative-commutative unification, see Section 13.1. Associative unification is infinitary [Plo72], take for example the equation $x + a =^?_{\mathsf{A}(+)} a + x$ which independant $\mathsf{A}(+)$-unifiers are $\{x \mapsto a\}, \{x \mapsto a + a\}, \{x \mapsto a + (a + a)\}, \cdots$. We have seen that the theory FH is nullary.

One can wonder if this classification can be enhanced by allowing U-finitary theories with only 2 most general elements and 3 and 4..., but this is hopeless due to the following result:

Proposition 10.9 [BS86] In any given U-finitary but non U-unitary theory, there exists an equation the complete set of unifiers of which has more that n elements for any given natural number n.

Finally, given a finite presentation of a theory E, its position in the unification hierarchie is undecidable, i.e. it is undecidable whether E is U-unitary, U-finitary, U-infinitary or U-nullary [Nut89].

The Table 10.1 summarizes part of our current knowledge about unification in equational theories.

Name	Type	Deci	Main references
$\emptyset$	1	yes	[Her30, Rob65, Rob71, Hue76, MM82, PW78, CB83, Fag83]. Subject of Section 3.2.
$A(f)$	∞	yes	The decidability has been proved in [Mak77]. Related works are [Abd87, AP90, Jaf90, Plo72], [Péc81, Sie75, LS75].
$C(f)$	ω	yes	[Sie79, Her87, Kir86]. Subject of Section 10.5.4.
$I(f)$	ω	yes	First studied in [RS78]. Hullot [Hul80a] derives an algorithm by narrowing.
$A(f), C(f)$	ω	yes	[Sti76, Sti75, Sti81, LS76, Fag84, HS85, Kir89a, Kir85a]. Subject of Section 13.1.
$A(f), I(f)$	0	yes	Studied in [SS82b, Baa86a, SS86b]. The variety of bands has been completely investigated in [Baa86b].
$C(f), I(f)$	ω	yes	[RS78, JKK83]
$A(f)$, $C(f)$, $I(f)$, $U(f,1)$	ω	yes	[BB88, Dro92]
$A(+)$, $C(+)$, $U(+,0)$, $E(h,+)$, $UE(h,0)$	?	no	[Nar96]
$A(f)$, $C(f)$, $E(h,f)$	?	no	[BB88, Dro92]
$Dr(f,g)$	1	yes	[AT85, TA87]
$Dl(f,g)$	1	yes	[AT85, TA87]
$D(f,g)$	∞	open	[Sza82] give a first study of the theory.
$D(f,g), A(g)$	∞	no	[Sza82, page 150]
$D(f,g)$, $A(g)$,$A(f)$	∞	no	[Sza82, page 151]
$D(f,g)$, $C(f)$, $C(g)$	∞	open	[Sza82]
$D(f,g)$, $A(f)$, $A(g)$, $I(f)$, $I(g)$	open	yes	[Sza82]
$Dl(f,g)$, $A(g)$, $C(g)$	?	no	[Nar96]
$D(f,g)$, $A(g)$, $C(g)$	∞	no	[Sza82]
$Dl(f,g)$, $A(g)$, $Ur(f,1)$	∞	no	[AT85, TA87]
$H(f,*,+)$	1	yes	[Vog78]
$H(f,*,+), A(+)$	∞	yes	[Vog78]
$H(f,*,+)$, $A(+)$, $C(+)$	ω	yes	[Vog78]
$E(h,*)$	1	yes	[Vog78]
$E(h,*)$, $A(*)$, $C(*)$	∞	no	First studied in [Vog78], the undecidability result is proved in [Nar96]
$Cr(f)$	ω	yes	[Pla93, Kir85a, Jea80]
QG	ω	yes	[Hul80c]

continued on next page

continued from previous page			
Name	Type	Deci	Main references
AG	ω	yes	[Lan79b, LBB84]
BR	1	yes	[MN89]
$Minus$	ω/∞	yes	Depending of the arity of the symbols, the type may be infinite [KK82, Kir84a].
BST	ω/∞	yes	Depending of the arity of the symbols, the type may be infinite [KK82].

Table 10.1: Some results on equational unification

10.5 Transforming equational problems

10.5.1 A Rule-Based Approach to Unification

Following [Her30, MM82], we view unification as a step by step process of transforming unification problems until a *solved form* is obtained, from which a representation of all unifiers can be easily extracted. These rules can be designed to be either deterministic or non-deterministic. In the first case, no matter which rule is applied, an equivalent unification problem is obtained. In the latter case, several transformation rules may apply concurrently on the same input problem to obtain an equivalent set of unification problems. This last situation is the way [JK91] deals with disjunctions of systems. We take here the first solution which have the advantage to explicit all the choices at once and the drawback to induce a more compact description of the transformation rules. In both cases, these transformational approaches, because of the clear distinction made between the transformation rules and their actual use (the control), have the following key advantages:

1. It eases the design (and the understanding as well) of (complex) unification algorithms, by providing a systematic guide which consists in:
   - Choose the intended solved forms. This characterizes which equational problems actually correspond to most general unifiers or more generaly to the desired simplified form of equational problems.
   - Determine the transformation rules. For each possible equational problem which is not in solved form, write transformation rules replacing this set with an equivalent one.
   - Determine the appropriate control. This will determine which application of the transformation rules is intended in order to reach the solved forms.
2. It eases correctness proofs by providing a systematic guide for proving the following fundamental properties:
   - Soundness; that is to show that each rule preserves the set of unifiers.
   - Completeness; that is to show that normal forms (with respect to the transformation rules) are indeed solved forms. This is usually quite easy, since the rules are actually designed to achieve this purpose.
   - Termination; one has to show that the rules terminate for some specific control or class of controls or class of input.
   - Fairness; that is to show that normal forms are indeed reached for the chosen control.

Termination happens to be generally the difficult part of most proofs.

10.5.2 Solved forms for Unification Problems

We will now need a slightly more general form of solved form than the one introduced for syntactic unification but the reader will recognize the same ideas concerning tree and dag solved forms. Existential quantifiers are used in both, but may be omited when they are not necessary.

Definition 10.9 A *tree solved form* is any conjunction of equations:

$$\exists \vec{z},\ x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$$

such that $\forall 1 \leq i \leq n, x_i \in \mathcal{X}$ and:

$$\begin{array}{lll} (i) & \forall 1 \leq i < j \leq n & x_i \neq x_j, \\ (ii) & \forall 1 \leq i, j \leq n & x_i \notin \mathcal{V}ar(t_j), \\ (iii) & \forall 1 \leq i \leq n & x_i \notin \vec{z}, \\ (iv) & \forall z \in \vec{z}, \exists 1 \leq j \leq n & z \in \mathcal{V}ar(t_j). \end{array}$$

Given a unification problem P, we say that $\exists \vec{z},\ x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$ is a tree solved form for P if it is a tree solved form equivalent to P and all variables free in $\exists \vec{z},\ x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$ are free variables of P.

In the above definition, the first condition checks that a variable is given only one value, while the second checks that this value is a finite term. The third and fourth conditions check that the existential variables are useful, i.e., that they contribute to the value of the other variables.

Tree solved forms have the following important property which is a straighforward extention of Lemma 3.8:

Lemma 10.5 Let $\mathcal{A}$ be an $\mathcal{F}$-algebra. A unification problem P with tree solved form:

$$P = \exists \vec{z},\ x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$$

has, up to $\mathcal{A}$-subsumption equivalence, a unique most general idempotent unifier $\{x_1 \mapsto t_1, \cdots, x_n \mapsto t_n\}$ in $\mathcal{A}$ which is denoted μ_P.

Regardless in which algebra $\mathcal{A}$-solutions are to be computed, tree solved forms have a unique most general idempotent unifier representing the set of all solutions in $\mathcal{A}$. When dealing with tree solved forms, the mention of the algebra of interest is therefore superfluous.

We refer to the Theorem 3.3 for establishing the canonicity of tree solved forms.

We now extend the notion of dag solved form given for syntactic unification.

Definition 10.10 A *dag solved form* is any set of equations

$$\exists \vec{z}\ , x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$$

such that $\forall 1 \leq i \leq n, x_i \in \mathcal{X}$ and:

$$\begin{array}{lll} (i) & \forall 1 \leq i < j \leq n & x_i \neq x_j, \\ (ii) & \forall 1 \leq i \leq j \leq n & x_i \notin \mathcal{V}ar(t_j), \\ (iii) & \forall 1 \leq i \leq n & t_i \in \mathcal{X} \Rightarrow x_i,\ t_i \notin \vec{z}, \\ (iv) & \forall z \in \vec{z}, \exists 1 \leq j \leq n & z \in \mathcal{V}ar(t_j). \end{array}$$

Given a unification problem P, we say that $\exists \vec{z}\ , x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$ is a dag solved form for P if it is a dag solved form equivalent to P and all variables free in $\exists \vec{z}\ , x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$ are free variables of P.

The second condition forbids cycles. The third condition forbids that an existentially quantified variable be equal to a free variable. Moreover, if a free variable x is mapped to another free variable y, then x may not appear in any other right hand side. The last condition requires that any existentially quantified variable contributes to the value of a (free by transitivity) variable. In some sense, both conditions together describe the usefulness of quantified variables. Note that some of the x_i may be existentially quantified in dag solved forms.

Of course, a tree solved form for P is a dag solved form for P. Dag solved forms save space, since the value of the variable x_j need not be duplicated in the t_i for $i \leq j$. Conversely, a dag solved form yields a tree solved form by replacing x_i by its value t_i in all t_j such that $j < i$ and removing the remainding unnecessary existentially quantified variables. Formally the job is done using the quantifier elimination rule together with the following transformation rule:

$$\begin{array}{lll} \textbf{Eliminate} & P \wedge x =^? s & \\ \mapsto\!\!\!\!\!\!\rightarrow & \{x \mapsto s\}P \wedge x =^? s & \text{if } x \notin \mathcal{V}ar(s), s \notin \mathcal{X}, x \in \mathcal{V}ar(P) \end{array}$$

Dag2Tree: Transformation of dag to tree solved forms

As a consequence we get as an extention of Lemma 3.12:

Lemma 10.6 A unification problem $P = \exists\vec{z}\ , x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$ in dag solved form has, up to $\mathcal{A}$-subsumption equivalence, a unique most general idempotent unifier $\sigma = \sigma_n \cdots \sigma_2\sigma_1$, where $\sigma_i = \{x_i \mapsto t_i\}$.

The occur check relation is then defined in the same way as for syntactic unification problems, see Definition 3.6.

In the following, we refer without further precision to the most general unifier associated to a particular solved form by either one of the above lemmas.

10.5.3 Equivalence

Let us state here the most commonly used tranformations that preserve the set of solutions.

Proposition 10.10 Let E be a set of equational axioms built on terms of TFX. Then one can replace any subterm t in an equational problem P with an E-equal term without changing the set of E-unifiers of P.

In particular rewriting by some term rewriting system that is a sub-theory of E preserves the set of E-unifiers.

One quite important set of rules preserve equivalence of equational problems; they are the rules that allow manipulating the connectors $\wedge$ and $\vee$. The most commonly used such rules are described in Figure 10.2.

Proposition 10.11 All the rules in **Simplification** preserve the set of $\mathcal{A}$-unifiers, for any $\mathcal{F}$-algebra $\mathcal{A}$.

10.5.4 The commutativity example

Let us terminate this chapter with the easy but useful example of the description and proof of a unification algorithm for the commutativity of a symbol $+$.

As in the free case, unification algorithms try to transform a unification problem into a set of equivalent unification problems in solved form by using appropriate rules.

We now give the rules for a commutative theory: assume that $\mathcal{F} = \{a, b, \ldots, f, g, \ldots, +\}$ and that $+$ is commutative, the others being free. Then a set of rules for unification in this theory can be easily obtained by adding a *mutation* rule to the set **Dag-Unify**, which describes the effect of the commutativity axiom. We call the resulting set of rules **CommutativeUnification**: it is described in Figure 10.3.

We see how easy it is here to obtain a set of rules for unification modulo commutativity from the set of rules for syntactic unification. Note that there is no need of using existential quantifiers here.

In order to study this set of transformation rules we are following on purpose the same approach as we have done for **SyntacticUnification**.

First of all let us prove that all these rules are sound i.e. preserve the set of unifiers.

Lemma 10.7 All the rules in **CommutativeUnification** are sound.

Proof: The main difference with syntactic unification is of course the rule **ComMutate** which soundness is an obvious consequence of the commutativity of $+$. □

Definition 10.11 A *commutative unification procedure* is any sequence of application of the transformation rules in **CommutativeUnification** on a finite set of equations P.

In fact a strategy of application of the rules in **CommutativeUnification** determines a unification procedure. Some are complete, some are not. Let us first show that a brute force fair strategy is complete.

Theorem 10.2 *Starting with a unification problem P and using the above rules in* **CommutativeUnification** *repeatedly until none is applicable results in* **F** *iff P has no C-unifier, or else it results in a finite disjunction of tree solved form:*

$$\bigvee_{j \in J} x_1^j =^?_{\mathsf{C}} t_1^j \wedge \cdots \wedge x_n^j =^?_{\mathsf{C}} t_n^j$$

having the same set of C-unifiers than P. Moreover:

$$\Sigma = \{\sigma^j | j \in J \text{ and } \sigma^j = \{x_1^j \mapsto t_1^j, \ldots, x_n^j \mapsto t_n^j\}\}$$

is a complete set of C-unifiers of P.

Associativity- $\wedge$	$(P_1 \wedge P_2) \wedge P_3$	$=$	$P_1 \wedge (P_2 \wedge P_3)$
Associativity- $\vee$	$(P_1 \vee P_2) \vee P_3$	$=$	$P_1 \vee (P_2 \vee P_3)$
Commutativity- $\wedge$	$P_1 \wedge P_2$	$=$	$P_2 \wedge P_1$
Commutativity- $\vee$	$P_1 \vee P_2$	$=$	$P_2 \vee P_1$

Trivial	$P \wedge (s =^? s)$	$\to$	P
AndIdemp	$P \wedge (e \wedge e)$	$\to$	$P \wedge e$
OrIdemp	$P \vee (e \vee e)$	$\to$	$P \vee e$
SimplifAnd1	$P \wedge \mathbf{T}$	$\to$	P
SimplifAnd2	$P \wedge \mathbf{F}$	$\to$	$\mathbf{F}$
SimplifOr1	$P \vee \mathbf{T}$	$\to$	$\mathbf{T}$
SimplifOr2	$P \vee \mathbf{F}$	$\to$	P
Distrib	$P \wedge (Q \vee R)$	$\to$	$(P \wedge Q) \vee (P \wedge R)$
Propag	$\exists \vec{z} : (P \vee Q)$	$\to$	$(\exists \vec{z} : P) \vee (\exists \vec{z} : Q)$
Elimin0	$\exists z : P$	$\to$	P
			if $z \notin \mathcal{V}ar(P)$
Elimin1	$\exists z : z =^? t \wedge P$	$\to$	P
			if $z \notin \mathcal{V}ar(P) \cup \mathcal{V}ar(t)$

Figure 10.2: **Simplification**: Rules for connectors simplification

Proof: By Lemma 3.13 all the rules considered preserve the set of C-unifiers. We shall now prove that the process stops and that the normal forms are indeed tree solved forms.

This last point follows immediately from a step by step inspection of the different cases, we left it as an exercice to the reader.

The most difficult point is to prove that the process terminates. As in the syntactic case this is not completely obvious since the rule **Eliminate** makes the terms bigger but **Decompose** and **ComMutate** decrease their size. Moreover we have now to consider disjunction of systems and to define accordingly the complexity of a disjunction of systems.

The complexity of one equation is defined as for syntactic unification:

$$I(s =^?_{\mathsf{C}} t) = (\max(|s|, |t|), type(s, t))$$

and the complexity of a system is defined as:

$$\begin{array}{rcl} I(\mathbf{F}) & = & (0, \emptyset) \\ I(s_1 =^?_{\mathsf{C}} t_1 \wedge \ldots \wedge s_n =^?_{\mathsf{C}} t_n) & = & (N, \{I(s_1 =^?_{\mathsf{C}} t_1), \ldots, I(s_n =^?_{\mathsf{C}} t_n)\}) \end{array}$$

where N is the number of unsolved variables, i.e. of variables that are not solved. We compare the complexities lexicographically, using the standard ordering on naturals for the first component and the multiset ordering for the second component. Finally the complexity of a finite disjunction of systems $D = \bigvee_{j \in J} P^j$ is the multiset of the complexities of its systems: $I(D) = \{I(P^j) | j \in J\}$ compared by the multiset extention of the ordering on system complexities.

We will now check that for **CommutativeUnification**, each application of a rule decreases the complexity of the disjunction of systems on which it is applied.

We have seen when proving **SyntacticUnification** that all the rules but **ComMutate** decrease the complexity of a system thus of a disjunction.

The last case is **ComMutate**. As **Decompose**, it may increase the number of solved variables but in any case it decreases the second component since it replaces an equation by strictly smaller ones, thus a system is replaced by the disjunction of two other ones whose complexity is strictly smaller, thus the complexity of the whole disjunction decreases too.

Finally, all the σ^j are solutions of P since they are mgu for a tree solved form in the disjunction of tree solved forms issued from P. That Σ is a complete set of C-unifiers results from the definition of the set of unifiers of a disjunction. □

Since we have proved that the whose set of rule terminates, we can envisage as for syntactic unification complete restrictions of it. Let us first define useful subsets of the rules in **CommutativeUnification**. We introduce the set of rules:
TreeComUnify =
{**Delete, Decompose, ComMutate, Conflict, Coalesce, Check, Eliminate**}
and
DagComUnify =
{**Delete, Decompose, ComMutate, Conflict, Coalesce, Check*, Merge**}.

Corollary 10.3 Starting with a unification problem P and using the rules **TreeComUnify** repeatedly until none is applicable, results in **F** iff P has no C-unifier, or else it results in a finite disjunction of tree solved form:

$$\bigvee_{j\in J} x_1^j =^?_{\mathsf{C}} t_1^j \wedge \cdots \wedge x_n^j =^?_{\mathsf{C}} t_n^j$$

having the same set of C-unifiers than P. Moreover:

$$\Sigma = \{\sigma^j | j \in J \text{ and } \sigma^j = \{x_1^j \mapsto t_1^j, \ldots, x_n^j \mapsto t_n^j\}$$

is a complete set of C-unifiers of P.

Proof: This is a clear consequence of Theorem 10.2. In fact **Merge** and **Check*** are useless for getting the tree solved forms. Termination is of course not affected when the set of rules is restricted. □

Exercice 43 — Apply the set of rules **TreeComUnify** to the following unification problem $x + g(x + y) =^?_{\mathsf{C}} g(y + z) + g(h(z))$.
Answer:
We can also forbid the application of the **Eliminate** rule, in which case we get dag solved forms:

Corollary 10.4 Starting with a unification problem P and using the rules **DagComUnify** repeatedly until none is applicable, results in **F** iff P has no C-unifier, or else it results in a finite disjunction of dag solved form:

$$\bigvee_{j\in J} x_1^j =^?_{\mathsf{C}} t_1^j \wedge \cdots \wedge x_n^j =^?_{\mathsf{C}} t_n^j$$

having the same set of C-unifiers than P. Moreover:

$$\Sigma = \{\sigma^j | j \in J \text{ and } \sigma^j = \{x_n^j \mapsto t_n^j\} \cdots \{x_1^j \mapsto t_1^j\}\}$$

is a complete set of C-unifiers of P.

Proof: This is also a clear consequence of Theorem 10.2. One can check easily that the normal forms are indeed dag solved forms and termination is not affected. □

By removing **Eliminate**, we get a set of rules for solving equations over infinite trees, exactly as for syntactic unification.

10.5.5 Complexity of Commutative Unification

This first study of a non trivial equational unification algorithm allows us to state the questions that designers will reach when studying equational unification:

1. Are the images of a given term, under the application of the substitutions in a minimal complete set of unifiers, independant for the subsumption ordering?
2. Are the complete sets of unifiers, obtained under a given set of transition rules, minimal?
3. What is the complexity of the cardinality of the minimal complete set of unifiers with respect to the input system?
4. What is the complexity of the decidability of equational unification?
5. What is the complexity of the enumeration of minimal complete set of unifiers?

To the first question, the answer is negative: When applied to the equation $s =^?_{\mathsf{C}} t$, commutative unification may generate redundant unified terms $\sigma(s)$, even if the substitutions σ are most general. This is the case with the simple problem $x + y =^?_{\mathsf{C}} a + b$, whose complete set of most general unifiers is

$$\{\{x \mapsto a,\ y \mapsto b\},\ \{x \mapsto b,\ y \mapsto a\}\},$$

since $a + b$ and $b + a$ are equivalent under commutativity.

Even worst, there seems to be no general solution to question 2 even in the case of commutativity since in general the unification algorithm generated by the rules above may return non minimal complete sets of unifiers, as for the problem $x + y =^? (a + b) + (b + a)$. Of course any post processing consisting to check for redondancy after the generation of a CSU is applicable but since this require to check all the elements of a CSU two by two, this will be quite expensive. This kind of redundancy of complete sets of most general commutative unifiers has been studied by [Sie79].

The number of minimal complete set of commutative unifiers can be quite large. For example the matching problem

$$(x_1 + x_2) + (x_3 + x_4) =^? (a + b) + (c + d)$$

has 4! independant unifiers and there exists problems with $2 * n$ leaves having a CSMGU of n! elements. Thus the cardinality of CSMGU can be exponential in the size of the input problem.

Delete $P \wedge s =^?_{\mathsf{C}} s$

$\twoheadrightarrow$

P

Decompose $P \wedge f(s_1, \dots, s_n) =^? f(t_1, \dots, t_n)$

$\twoheadrightarrow$

$P \wedge s_1 =^? t_1 \wedge \dots \wedge s_n =^? t_n$

if $f \neq +$

ComMutate $P \wedge s_1 + s_2 =^?_{\mathsf{C}} t_1 + t_2$

$\twoheadrightarrow$

$$P \wedge \left(\begin{array}{l} s_1 =^?_{\mathsf{C}} t_1 \wedge s_2 =^?_{\mathsf{C}} t_2 \\ \vee \\ s_1 =^?_{\mathsf{C}} t_2 \wedge s_2 =^?_{\mathsf{C}} t_1 \end{array} \right)$$

Conflict $P \wedge f(s_1, \dots, s_n) =^?_{\mathsf{C}} g(t_1, \dots, t_p)$

$\twoheadrightarrow$

F

if $f \neq g$

Coalesce $P \wedge x =^?_{\mathsf{C}} y$

$\twoheadrightarrow$

$\{x \mapsto y\}P \wedge x =^?_{\mathsf{C}} y$

if $x, y \in \mathcal{V}ar(P)$ *and* $x \neq y$

Check* $P \wedge \ x_1 =^?_{\mathsf{C}} s_1[x_2] \wedge \dots$
$\dots \wedge x_n =^?_{\mathsf{C}} s_n[x_1]$

$\twoheadrightarrow$

F

if $s_i \notin \mathcal{X}$ *for some* $i \in [1..n]$

Merge $P \wedge x =^?_{\mathsf{C}} s \wedge x =^?_{\mathsf{C}} t$

$\twoheadrightarrow$

$P \wedge x =^?_{\mathsf{C}} s \wedge s =^?_{\mathsf{C}} t$

if $0 < |s| \leq |t|$

Check $P \wedge x =^?_{\mathsf{C}} s$

$\twoheadrightarrow$

F

if $x \in \mathcal{V}ar(s)$ *and* $s \notin \mathcal{X}$

Eliminate $P \wedge x =^?_{\mathsf{C}} s$

$\twoheadrightarrow$

$\{x \mapsto s\}P \wedge x =^?_{\mathsf{C}} s$

if $x \notin \mathcal{V}ar(s), s \notin \mathcal{X}, x \in \mathcal{V}ar(P)$

Figure 10.3: **CommutativeUnification**: Rules for commutative unification

Chapter 11

Modular semantic unification

In the context of logic programming and deduction with constraints, the need for combining constraint solving in specific theories frequently appears. For instance, in first-order theorem proving, free constants and function symbols are generated during skolemization and it is known that unification with constants or general unification, where new constants or free function symbols are added to the signature, must be carefully distinguished from elementary unification [Bür86]. Combination problems also appear in constraint logic programming, when different kinds of constraints coexist and must be solved in appropriate domains for which a constraint solving process is already available. We focuss in this paper on the combination of symbolic constraint solvers that compute solutions which are substitutions defined on an appropriate set of terms.

Unification in an equational theory is a special case of symbolic constraint solving, for which the combination problem has already been addressed and can be stated as follows: given two unification algorithms in two (consistent) equational theories E_1 on a set of terms $\mathcal{T}(\mathcal{F}_1, \mathcal{X})$ and E_2 on $\mathcal{T}(\mathcal{F}_2, \mathcal{X})$, how to find a unification procedure for $E_1 \cup E_2$ on $\mathcal{T}(\mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{X})$? Combining unification algorithms was initiated in [Her86, Kir85a, Tid86, Yel87] where syntactic conditions on the axioms of the theories to be combined were assumed. Combination of arbitrary theories with disjoint sets of symbols is considered in [Bou90b, SS89, BS92]. The general idea of unification in a combination of theories consists in first breaking an equational problem into sub-problems that are pure in the sense that they can be solved in one component of the combination. Indeed a same variable could then be solved differently in each theory. To avoid this problem, a variable is considered as a constant in one theory while solved in the other. This motivates the need for each theory, of a unification algorithm taking into account additional free constants in the signature. In general, recombining the solutions obtained in each theory presents another difficulty due to cycles that may occur, for instance if x_1 is instantiated to $f(x_2)$ in the first theory and x_2 by $g(x_1)$ in the second. This problem is solved thanks to a linear restriction, that is an ordering on variables that must also be taken into account by the unification algorithms.

11.1 Combination problem for unification

The problem attacked in this chapter is how to combine two unification process defined on two different equational theories in order to get a unification procedure in the union of the two theories. Let us formalise some definitions and notations.

Definition 11.1 Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two mono-sorted first-order signatures, E_1 and E_2 be sets of $\mathcal{F}_1$ and $\mathcal{F}_2$ axioms respectively. The *combined theory* is defined by the quotient term algebra $\mathcal{T}(\mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{X})/=_{E_1 \cup E_2}$.

All along this section, we assume the following hypotheses.

Hypotheses 1
1) The first-order signatures $\mathcal{F}_1$ and $\mathcal{F}_2$ are finite and disjoint.
2) The theories defined by E_1 and E_2 are consistent.

We assume given a unification algorithms in each (consistent) equational theory and want to reuse them for designing a unification procedure for $E_1 \cup E_2$ on $\mathcal{T}(\mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{X})$. Given an heterogeneous equational problem whose terms are in $\mathcal{T}(\mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{X})$, the most natural idea is to split the problem into subproblems which only contains symbols from one signature. Such terms are called pure.

Definition 11.2 Let $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$. A term t of $\mathcal{T}(\mathcal{F} \cup \mathcal{X})$ is *i-pure* (for $i = 1, 2$) if t contains only function symbols from $\mathcal{F}_i$ and possibly variables of $\mathcal{X}$.

An equation $(s =^? t)$ is i-pure if s and t are.

A term with its top symbol in $\mathcal{F}_i$ is called *i-term*. *Alien* subterms of a i-term are j-(sub)terms such that each prefix symbol is in $\mathcal{F}_i$.

Example 11.1 Let $\mathcal{F}_1 = \{f\}$ and $\mathcal{F}_2 = \{g\}$. The term $f(y, g(z, g(x)))$ is a 1-term and $g(z, g(x))$ is an alien subterm.

The equations $(g(z, g(x)) =^? z)$, $(g(z, g(x)) =^? g(y, y))$ are 2-pure, The equation $(x =^? z)$ is both 1-pure and 2-pure.

Some other notions must be defined.

Definition 11.3

- The set of alien positions of a term t is

$$AlienPos(t) = \{\omega \neq \epsilon \mid t(\omega) \notin \mathcal{F}_i \cup \mathcal{X} \text{ et } \forall \omega' \in \mathcal{P}os(t),\ \omega' < \omega \Rightarrow t(\omega') \in \mathcal{F}_i,\ i = 1, 2\}.$$

- The set of alien subterms of a term t is $AST(t) = \{t_{|\omega} \mid \omega \in AlienPos(t)\}$.
- The number of layers of theories $ht(t)$ of a term t is 0 if $AST(t) = \emptyset$, else $ht(t) = 1 + \max_{s \in AST(t)} ht(s)$.

We first focus on a simple case, where the three steps necessary to perform unification in the combined theory are detailled. The first step is to split the initial heterogeneous equational problem Γ into two subproblems Γ_1 and Γ_2 respectively 1-pure and 2-pure. This is performed thanks to an operation called abstraction. The second step consists in solving each subproblem in its component. The third step is the combination of the obtained solutions. This is the more delicate point since conflicts between theories and cycles may occur at this step.

11.2 Combination of simple theories

In simple theories [Kir85a], the difficulty due to cycles may be avoided, since cycles have by definition no solution.

Definition 11.4 A theory E is *simple* if for any term t the problem $x =^?_E t[x]_\omega$ with $\omega \neq \epsilon$ has no solution.

The empty theory is simple, as well as C and AC.

This hypothesis implies strong syntactic restriction of the axioms of E.

Proposition 11.1 A simple theory is regular and collapse-free.

Proof: If E is collapse, then there exists an E-equality $x =_E t[x]$ and the identity substitution is solution of $x =^? t[x]$. If E is not regular, then there exists an E-equality $l =_E r[x]$ with $x \notin \mathcal{V}(l)$ and $\{x \mapsto l\}$ is solution of $x =^? r[x]$. □

The converse does not hold since the theory $\{a = f(a)\}$ is not simple, although regular and collapse-free. The problem of stating whether an arbitrary theory belongs to the class of simple theories has been shown undecidable [BHSS90].

11.2.1 Abstraction

As already said, the first step is to split a problem Γ into pure subproblems. So an equation is decomposed into a conjunction of pure equations by introducing new equations of the form $x =^? t$ where t is an alien subterm and x is a variable that does not appear yet. This is formalised thanks to the notion of abstraction.

Definition 11.5 A *variable abstraction* is a one-to-one mapping π from the quotient set of terms $(T(\mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{X})/E_1 \cup E_2$ to a subset of $\mathcal{X}$.

The term t^{π_i}, called *i-abstraction* of the term t, is inductively defined as follows:
- if $t = x \in \mathcal{X}$ then $t^{\pi_i} = x$,
- if $t = f(s_1, \ldots, s_n)$ and $f \in \mathcal{F}_i$ then $t^{\pi_i} = f(s_1^{\pi_i}, \ldots, s_n^{\pi_i})$
- if $t = f(s_1, \ldots, s_n)$ and $f \notin \mathcal{F}_i$ then $t^{\pi_i} = \pi(t)$.

Example 11.2 Let $\mathcal{F}_1 = \{f\}$ and $\mathcal{F}_2 = \{g\}$. The 1-abstraction of the term $t = f(g(x), y)$ is the term $f(z, y)$. Its 2-abstraction is a variable z'.

Given a substitution σ, σ^{π_i} denotes its i-abstraction defined by $\sigma^{\pi_i}(x) = (\sigma(x))^{\pi_i}$ for any variable x in $\mathcal{D}om(\sigma)$.

The next lemma states some useful and easy properties of i-abstraction.

Lemma 11.1

- t^{π_i} is a term in $\mathcal{T}(\mathcal{F}_i, \mathcal{X})$.
- If t is i-pure, then $(\sigma(t))^{\pi_i} = \sigma^{\pi_i}(t)$ for any substitution σ in $\mathcal{T}(\mathcal{F}, \mathcal{X})$.

Iterating abstraction as long as possible yields two pure equational problems that should be solved in each component.

11.2.2 Solving in one component

The method which consists in solving a i-pure equation with respect to the equational theory E_i is obviously correct: we get substitutions which are unifiers since $=_{E_i}$ is included in $=_E$. Care must be taken that this method is also complete: each unifier must be an instance of at least one of these substitutions. This method has been shown complete for the combination of disjoint regular and collapse-free equational theories in [Yel87].

Lemma 11.2 Let E_1 and E_2 be collapse-free. If $s \longleftrightarrow_{E_1 \cup E_2} t$, then $s^{\pi_i} \overset{*}{\longleftrightarrow}_{E_1 \cup E_2} t^{\pi_i}$.

Proof: Assume that s is an i-term. If the $\longleftrightarrow_{E_1 \cup E_2}$ step occurs in an alien subterm of s, then $s^{\pi_i} = t^{\pi_i}$, since the subterm obtained by replacement remains an alien subterm for t, thanks to the assumption that E_1 and E_2 are collapse-free. If the $\longleftrightarrow_{E_1 \cup E_2}$ step occurs at a position above all alien positions in s, then this an $\longleftrightarrow_{E_i}$ step, where alien subterms of s only occur in the instanciation. So the same $\longleftrightarrow_{E_i}$ step applies from s^{π_i} to t^{π_i} and t is a i-term since E_i is collapse-free. So we also get $s^{\pi_j} = t^{\pi_j} \in \mathcal{X}$ for $j \neq i$. □

A replacement of equal by equal in a i-pure term is done via an E_i axiom to get an i-pure term.

Lemma 11.3 Let E_1 and E_2 be regular and collapse-free. If $s \in T(\mathcal{F}_i, \mathcal{X})$ and $s \longleftrightarrow_{E_1 \cup E_2} t$, then $t \in T(\mathcal{F}_i, \mathcal{X})$ and $s \longleftrightarrow_{E_i} t$.

Proof: Since E_j is collapse-free, the $\longleftrightarrow_{E_1 \cup E_2}$ step cannot be a $\longleftrightarrow_{E_j}$ step. In addition no alien subterm may be introduced in t by a $\longleftrightarrow_{E_i}$ step since E_i is regular, so t is i-pure. □

From Lemmas 11.2 and 11.3, it follows that to any equational proof in $E_1 \cup E_2$ corresponds a proof in E_i on abstracted terms.

Lemma 11.4 Let E_1 and E_2 be regular and collapse-free. For any terms s and t,

$$s =_{E_1 \cup E_2} t \Leftrightarrow s^{\pi_i} =_{E_1 \cup E_2} t^{\pi_i} \Leftrightarrow s^{\pi_i} =_{E_i} t^{\pi_i}.$$

This allows to prove completeness of solving in each component, expressed as follows:

Proposition 11.2 Let E_1 and E_2 be regular and collapse-free. For any s and t i-pure terms and any substitution σ,

$$\sigma(s) =_{E_1 \cup E_2} \sigma(t) \Leftrightarrow \sigma^{\pi_i}(s) =_{E_i} \sigma^{\pi_i}(t).$$

Proof: Since the terms s and t are i-purs, $(\sigma(s))^{\pi_i} = \sigma^{\pi_i}(s)$ and $(\sigma(t))^{\pi_i} = \sigma^{\pi_i}(t)$. □

The substitution σ is an instance of σ^{π_i} since $\sigma =_{E_1 \cup E_2} \sigma^{\pi_i}\pi^{-1}$.

Once the problems Γ_1 and Γ_2 are solved in their respective theories, it must be checked that from two solutions issued from each componant, a solution can be built in the union of theories.

11.2.3 Combination of solutions

In the case of simple theories a conjunction of solved forms either is already a solved form in the union, or has no solution because it contains a conflict of theories or a cycle.

If a variable is instantiated in both theories simultaneously, then the problem has no solution, thanks to the next result:

Proposition 11.3 If E_1 and E_2 are collapse-free, there exists no equality $s =_{E_1 \cup E_2} t$ with $s(\epsilon) \in \mathcal{F}_1$ and $t(\epsilon) \in \mathcal{F}_2$.

Proof: If $s =_{E_1 \cup E_2} t$ with $s(\epsilon) \in \mathcal{F}_1$ and $t(\epsilon) \in \mathcal{F}_2$, then $s^{\pi_1} =_{E_1} t^{\pi_1}$ with $t^{\pi_1} \in \mathcal{X}$ since $t(\epsilon) \in \mathcal{F}_2$. The theory E_1 would then be collapse, which contradicts the hypothesis. □

If the problem contains a cycle, then it has no solution, since the disjoint union of simple theories is yet a simple theory.

Proposition 11.4 Let E_1 and E_2 be regular and collapse-free. If E_1 and E_2 are two simple theories, then $E_1 \cup E_2$ is simple.

Proof: We prove by induction on the number n of layers of theories in t, that an equation $x =^?_{E_1 \cup E_2} t[x]_\omega$ has no solution. Assume that there exists a substitution σ such that $\sigma(x) =_{E_1 \cup E_2} \sigma(t)[\sigma(x)]_\omega$.

- If $n = 0$, then t is i-pure and $\sigma(x) =_{E_1 \cup E_2} \sigma(t)[\sigma(x)]$ implies $\sigma^{\pi_i}(x) =_{E_i} \sigma^{\pi_i}(t)[\sigma^{\pi_i}(x)]$, which contradicts the hypothesis on E_i.
- Otherwise, on may assume without loss of generality that $\sigma(x)(\epsilon), t(\epsilon) \in \mathcal{F}_i$ by Proposition 11.3. Then thanks to Lemma 11.4, $\sigma^{\pi_i}(x) =_{E_i} (\sigma(t)[\sigma(x)]_\omega)^{\pi_i}$.

  If there is no alien position between ϵ and ω, then $\sigma^{\pi_i}(x) =_{E_i} \sigma^{\pi_i}(t)[\sigma^{\pi_i}(x)]$, which contradicts the hypothesis on E_i.

  If there is an alien position between ϵ and ω, then some variable y abstracts a term containing $\sigma(x)$ as strict subterm. The same variable must also abstract an alien subterm of $\sigma(x)$ since E_i is regular and collapse-free. As a consequence, there exists a solution to an equation $y =^? u[y]$, where u is a term that has stricly less layers of theories than t. By induction hypothesis, this is impossible.

□

As a consequence, a cycle occurring in a problem which is a conjunction of solved forms has no solution. Actually assuming that theories are simple is a little too strong to obtain the fact that a cycle has no solution. It is enough to assume that both theories are regular and collapse-free. The algorithm given below actually works for the union of regular and collapse-free theories [Bou90a].

11.2.4 Unification algorithm for the union of two regular and collapse-free theories

The algorithm has strong similarities with unification in empty theory without replacement. An equational problem $\Gamma = \exists \vec{x} : P$, or more precisely its unquantified part P is transformed by the following rules as described below:

- Rules Variable Abstraction and Impure Equation transform the problem into a conjunction of pure problems.
- The rule E_i-Res solves the i-pure sub-problem in the theory E_i,
- The rule Conflict is similar to the rule with the same name for the empty theory, except that here it applies if the terms have their top symbols in different theories.
- The rule Cycle is identical to the rule with the same name for the empty theory.

Let Γ be the initial unification problem and $\Gamma_1 \wedge \Gamma_2$ the conjunction of problems respectively 1-pure and 2-pure obtained by repeated application of rules Variable Abstraction and Impure Equation.

Proposition 11.5 Applying to a unification problem Γ the rules Variable Abstraction et Impure Equation with an arbitrary control terminates and returns a unification problem $\exists \vec{x} : \Gamma_1 \wedge \Gamma_2$ equivalent to Γ.

Variable Abstraction

$$\frac{P \wedge s =^? t}{\exists x : P \wedge s =^? t[x]_\omega \wedge x =^? t_{|\omega}} \quad \text{if } \begin{cases} \omega \in AlienPos(t) \\ x \text{ is a new variable} \end{cases}$$

Impure Equation

$$\frac{P \wedge s =^? t}{\exists x : P \wedge x =^? s \wedge x =^? t} \quad \text{if } \begin{cases} s \in T(\mathcal{F}_1, \mathcal{X}) \backslash \mathcal{X},\ t \in T(\mathcal{F}_2, \mathcal{X}) \backslash \mathcal{X} \\ x \text{ is a new variable} \end{cases}$$

E_i-Res$^+$

$$\frac{P_i}{\exists \vec{x_i} : \hat{\sigma}_i} \quad \text{if } \sigma_i \in CSU_{E_i}(P_i) \text{ and } \vec{x_i} = \mathcal{V}(\hat{\sigma}_i) \backslash \mathcal{V}(P_i).$$

VarRep

$$\frac{P \wedge x =^? y}{P\{x \mapsto y\} \wedge x =^? y} \quad \text{if } x, y \in \mathcal{V}(P)$$

Conflict

$$\frac{P \wedge x =^? s \wedge x =^? t}{\mathbf{F}} \quad \text{if } s(\epsilon) \in \mathcal{F}_1, t(\epsilon) \in \mathcal{F}_2$$

Cycle

$$\frac{P \wedge x_1 =^? t_1[x_2]_{p_1} \wedge \cdots x_n =^? t_n[x_1]_{p_n}}{\mathbf{F}} \quad \text{if } p_i \neq \epsilon \text{ for some } i \in \{1, \ldots, n\}$$

Figure 11.1: Set of rules $\mathcal{RS}$ for unification in the union of disjoint regular and collapse-free theories

A normal form for {Variable Abstraction, Impure Equation} is a unification problem without heterogeneous equations. Considering the new variables as existencially quantified variables, Variable Abstraction and Impure Equation obviously preserve the sets of solutions.

The rules of $\mathcal{RS}$ in Figure 11.1 transform a problem into an equivalent one according to Propositions 11.3, 11.4 and 11.5. Il s'agit encore de vérifier qu'une forme normale est une forme séquentiellement résolue (et réciproquement).

Lemma 11.5 ([Bou90a]) If the rules of of $\mathcal{RS}$ applied with an arbitrary control to a problem Γ terminate, then the result is a disjonction of unification problems $\exists \vec{x} : P$ where P is in tree solved form.

Theorem 11.1 *If E_1 and E_2 are two simple (or regular and collapse-free) theories on disjoint signatures, then $E_1 \cup E_2$-unification is finitary iff E_i-unification ($i = 1, 2$) is finitary.*

11.3 General combination of unification with disjoint signatures

In the previous section, strong conditions were assumed on the theories to be combined. The goal is now to deal with the general case of disjoint signatures but without restriction of the form of axioms. In this context, new problems appear:

- an alien subterm does not always remain alien in a step of replacement of equal by equal,
- a i-pure term may be equal to a j-term, $i \neq j$, thanks to a collapse axiom,
- a new alien subterm may be introduced by replacement of equal by equal.

In order to provide an operational way to work in the equational theory $E_1 \cup E_2$, the idea is to impose directionality on the use of equations. The following construction was already used in [Bou90b, Bou90a, BS92].

Let us define $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$. Assume given a simplification ordering, total on terms of $\mathcal{T}(\mathcal{F} \cup \mathcal{X})$, denoted by $>$. It can be built by taking the union of two simplification orderings on each signature. Both E_1 and E_2 can be turned into ordered rewrite systems w.r.t. $>$ using ordered completion [BDP89]. Let

$$E_i^> = \{\sigma(l) \to \sigma(r) \mid l =_{E_i} r,\ \sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F} \cup \mathcal{X}), \sigma(l) > \sigma(r)\}.$$

$E_1^>$ and $E_2^>$ are convergent on $\mathcal{T}(\mathcal{F} \cup \mathcal{X})$ by construction. We restrict hereafter to the cases where their union is convergent too: this is the case if $\mathcal{F}_1$ and $\mathcal{F}_2$ are disjoint, because there is no critical pair between $E_1^>$ and $E_2^>$ and since simple termination is a modular property. The other interesting case is when the two signatures share only a finite number of constants [Rin92, Rin93]. A third case worth considering would be the case where $E_1^>$ and $E_2^>$ share constructors. Again simple termination plus confluence is modular [KO92].

Let us state more precisely hypotheses and results needed in the following.

11.3.1 Properties of the ordered rewrite system

In order to prove that solving a pure equation in the related component of the combination is correct and complete, we need the confluence and termination of the ordered system $E_1^> \cup E_2^>$.

Hypothesis 2 *$>$ is a simplification ordering total on $\mathcal{T}(\mathcal{F} \cup \mathcal{X})$ such that variables are minimal for $>$.*

With these hypotheses, we can state:

Proposition 11.6 If $l \to r \in E_1^> \cup E_2^>$ and $l(\epsilon) \in \mathcal{F}_i$ then $l \to r \in E_i^>$.

Proof: Consider the rule $\psi(g) \to \psi(d)$ in $E_i^>$ such that the top symbol of $\psi(g)$ is in $\mathcal{F}_j$ and $g \in \mathcal{T}(\mathcal{F}_i, \mathcal{X})$ then g is necessarily a variable x. Since E_i is consistent, the variable x occurs necessarily in d. Therefore $\psi(x)$ is a subterm of $\psi(d)$. Since $>$ is a simplification ordering, which satisfies the so-called subterm property, we have $\psi(g) = \psi(x) < \psi(d)$ which leads to a contradiction. □

Proposition 11.7 $E_1^> \cup E_2^>$ is convergent on $\mathcal{T}(\mathcal{F} \cup \mathcal{X})$.

Proof: Termination: Let s and t be terms in $\mathcal{T}(\mathcal{F} \cup \mathcal{X})$. If $s \to_R t$ then $s \to_{E_1^>} t$ or $s \to_{E_2^>} t$. In both cases, $s > t$ where $>$ is nœtherian on $\mathcal{T}(\mathcal{F} \cup \mathcal{X})$.

Confluence: Since R is terminating, it is sufficient to prove that R is locally confluent. Two cases must be considered (other ones are obvious).

First, we have a peak $s \leftarrow_{E_i^>} t \to_{E_i^>} u$, where (s, u) are instances of a critical pair (l, r), which is also a theorem of E_i. Hence $s = \psi(l), u = \psi(r)$ and $s \to u$ or $u \to s$ is a rule in $E_i^>$.

Second, we have a peak $s \leftarrow_{E_i^>} t \to_{E_j^>} u$. Let us assume without loss of generality that the rule in $E_i^>$ is applied at the position ϵ. Thus $t = \psi(g)$ and $s = \psi(d)$ where $g =_{E_i} d$ and $g, d \in \mathcal{T}(\mathcal{F}_i, \mathcal{X})$. According to Proposition 11.6, the rule in $E_j^>$ is necessarily applied at a position with a symbol in $\mathcal{F}_j$. Hence there is a variable $x \in \mathcal{V}(g)$ at a position ω, a substitution ψ' and a position υ such that $u = \psi(g)[\psi'(r)]_{\omega \cdot \upsilon}$ with $\psi(x)_{|\upsilon} = \psi'(l)$ and $\psi'(l) \to \psi'(r) \in E_j^>$. Let σ be the substitution defined by $\sigma(y) = \psi(y)$ for $y \neq x$ and $\sigma(x) = \psi(x)[\psi'(r)]_\upsilon$. Then

$$s \xrightarrow{*}_{\psi'(l) \to \psi'(r)} \sigma(s) \longleftrightarrow_{\epsilon, \sigma(g) \longleftrightarrow \sigma(d)} \sigma(u) \xleftarrow{*}_{\psi'(r) \leftarrow \psi'(l)} u.$$

□

Let us now define R as the rewrite system included into $E_1^> \cup E_2^>$ and defined by

$$R = \bigcup_{i=1}^{2} \{\sigma(l) \to \sigma(r) \mid l =_{E_i} r,\ \sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F} \cup \mathcal{X}),\ \sigma(l) > \sigma(r),$$

$$\sigma(x) \text{ is } E_1^> \cup E_2^>\text{-normalized for any } x \in \mathcal{V}(r) \backslash \mathcal{V}(l)\}.$$

The restriction to normalized instances for variables in $\mathcal{V}(r) \backslash \mathcal{V}(l)$ ensures that the reduction of a term does not introduce new reducible alien subterms.

Corollary 11.1 Given two terms s and t in $\mathcal{T}(\mathcal{F} \cup \mathcal{X})$, $s =_{E_1 \cup E_2} t$ if and only if $s \downarrow_R = t \downarrow_R$.

Now deciding an i-pure equality in $E_1 \cup E_2$ can be performed by normalization using the ordered system R built from $E_1^>$ and $E_2^>$. The R-normal form of a term $\sigma(t)$ where t is i-pure and σ is a R-normalized substitution, is obtained by applying only rewrite steps with rules from $E_i^>$ and alien subterms occurring only in the substitution part. It is then possible to do a similar proof with axioms from E_i on terms where alien subterms have been replaced with new variables. Indeed equal subterms have to be replaced by the same variable. This is the purpose of variable abstraction described in the next section.

11.3.2 Abstraction

Again an heterogeneous equational problem is decomposed into a conjunction of pure subproblems by introducing new equations of the form $x =^? t$ where t is an alien subterm in an equation and x is a new variable. Since we now choose $t \downarrow_R$ as the representant of an equivalence class modulo $E_1 \cup E_2$, the notion of abstraction has to be modified accordingly.

Definition 11.6 A *variable abstraction* is a one-to-one mapping π between the set of R-normalized terms $\mathcal{T} \downarrow_R = \{u \downarrow_R \mid u \in \mathcal{T}(\mathcal{F} \cup \mathcal{X}) \text{ and } u \downarrow_R \in \mathcal{T}(\mathcal{F} \cup \mathcal{X}) \backslash \mathcal{X}\}$ and a subset of $\mathcal{X}$. π^{-1} denotes the substitution with a possibly infinite domain which corresponds to the inverse of π.

The term t^{π_i}, called *i-abstraction* of the term t, is inductively defined as follows:
• if $t = x \in \mathcal{X}$ then $t^{\pi_i} = x$,
• if $t = f(s_1, \ldots, s_n)$ and $f \in \mathcal{F}_i$ then $t^{\pi_i} = f(s_1^{\pi_i}, \ldots, s_n^{\pi_i})$
• else if $t \downarrow_R \notin \mathcal{X}$ then $t^{\pi_i} = \pi(t \downarrow_R)$ else $t^{\pi_i} = t \downarrow_R$.

Example 11.3 Let us consider $\mathcal{F}_1 = \{1, \times\}$, $\mathcal{F}_2 = \{\top, +\}$, $E_1 = \{x \times 1 = x\}$, $E_2 = \{x + \top = \top\}$ and the heterogeneous term $t = (y \times (y + \top)) \times 1$. Its 1-abstraction t^{π_1} is $(y \times \top) \times 1$ since $(y + \top) \downarrow_R = \top$. Its 2-abstraction t^{π_2} is z since $t \downarrow_R = y \times \top$.

The next lemma states another property of i-abstraction.

Lemma 11.6 If t is R-normalized then $t = \pi^{-1}(t^{\pi_i})$. So, if σ is R-normalized then $\sigma = \pi^{-1}\sigma^{\pi_i}$, which means $\sigma^{\pi_i} \leq^{\mathcal{X}}_{\mathcal{T}(\mathcal{F},\mathcal{X})} \sigma$.

11.3.3 Solving in one component

As before solving a i-pure equation with respect to the equational theory E_i is obviously correct, but completeness must be checked with a more sophisticated method, but similar ideas.

Lemma 11.7 Let s be a i-term where alien subterms are R-normalized. Then

- $s \rightarrow_R t$ such that t is either a j-term R-normalized or a i-term where alien subterms are R-normalized.
- $s^{\pi_i} =_{E_i} t^{\pi_i}$.

Proof: By assumption, all alien subterms are in normal form. Then, if a rule in R may be applied, it is necessarily at a position with a non-constant symbol in $\mathcal{F}_i$. According to Proposition 11.6, this rule is in $E_i^{>}$. Thanks to the definition of R, no new alien subterm is introduced during a rewriting step. So irreducibility of alien subterms is preserved during the rewrite step. Since alien subterms always belong to instantiated parts of rules, the same proof holds, now with "replacement of equals for equals", when alien subterms are substituted with (new) variables. This step corresponds to the application of a variable abstraction. Hence $s^{\pi_i} =_{E_i} t^{\pi_i}$. □

Corollary 11.2 If s is a i-term where alien subterms are R-normalized then $s^{\pi_i} =_{E_i} (s \downarrow_R)^{\pi_i}$.

Proof: Thanks to Lemma 11.7 and nœtherian induction on $\rightarrow_R$. □

Corollary 11.3 For any i-pure term s and any R-normalized substitution σ, $\sigma^{\pi_i}(s) = (\sigma(s) \downarrow_R)^{\pi_i}$.

Proof: Alien subterms of $\sigma(s)$ are R-normalized and $\sigma^{\pi_i}(s) = (\sigma(s))^{\pi_i}$. □

We are now able to prove that any unifier in $E_1 \cup E_2$ of a i-pure equation $(s =^? t)$ corresponds to a unifier in E_i.

Proposition 11.8 Let s and t be two i-pure terms and σ a R-normalized substitution. Then

$$\sigma(s) =_{E_1 \cup E_2} \sigma(t) \Leftrightarrow \sigma^{\pi_i}(s) =_{E_i} \sigma^{\pi_i}(t).$$

Proof: ($\Leftarrow$) is obvious (correctness): $\sigma^{\pi_i}(s) =_{E_i} \sigma^{\pi_i}(t) \Rightarrow \pi^{-1}(\sigma^{\pi_i}(s)) =_{E_i} \pi^{-1}(\sigma^{\pi_i}(t))$. This equality is identical to $\sigma(s) =_{E_i} \sigma(t)$. Then, we just argue that a E_i-theorem is also a E-theorem. This is the only assertion we need for proving that, for a i-pure equation, a E_i-unifier is also a E-unifier.

Let us prove ($\Rightarrow$) (completeness). If $\sigma(s) =_{E_1 \cup E_2} \sigma(t)$ then $\sigma(s) \downarrow_R = \sigma(t) \downarrow_R$ and so,

$$\sigma^{\pi_i}(s) =_{E_i} (\sigma(s) \downarrow_R)^{\pi_i} = (\sigma(t) \downarrow_R)^{\pi_i} =_{E_i} \sigma^{\pi_i}(t).$$

□

Note that $\sigma^{\pi_i} \leq^{\mathcal{X}}_{E_1 \cup E_2} \sigma$. Hence a complete set of E_i-unifiers of a i-pure equational problem $(s =^? t)$ is a complete set of $(E_1 \cup E_2)$-unifiers this equation.

11.3.4 Combination of solutions

By repeatedly applying this transformation, an heterogeneous equational problem Γ is transformed into the conjunction of a pure subproblems Γ_1 and Γ_2. The solving process should be applied on Γ_i, but some new difficulties may be considered:

• Abstraction produces pure equational problems in each theory by introducing new variables to split terms. These new variables are shared by the two theories and may further be instantiated in both of them. So, all possible choices for instantiating a variable in a theory have to be considered. When a variable is instantiated in i, it is considered as a constant in $j \neq i$. Unification in each component now needs more than the assumed unification algorithm, since new free constant symbols of the signature have to be taken into account. It is known that unification and general unification (unification with additional free constant symbols) are in general not equivalent.

• The problem introduced by abstraction is that two distinct variables may be introduced that actually denote two equal or equivalent terms. So care must be taken that distinct variables could be identified with a solution as in the next example.

Example 11.4 Let us consider the combination of the 2-elements Boolean algebra with two free symbols a, f. The equational problem $(x + y =^? x) \wedge (x =^? f(z)) \wedge (y =^? f(a))$, where x and y are free constants in the Boolean equation has no solution. But if x and y are variables introduced by abstraction, they may be indeterministically instantiated in any theory. Then the equational problem $(x =^? y) \wedge (x + y =^? x) \wedge (x =^? f(z)) \wedge (y =^? f(a))$, where x, y are identified and only y is considered as a free constant in the Boolean equation, is equivalent to $(x =^? y) \wedge (y + y =^? y) \wedge (y =^? f(z)) \wedge (y =^? f(a))$ and then to the solved form $(x =^? y) \wedge (y =^? f(z)) \wedge (z =^? a)$.

Thus, for the sake of completeness, before solving an equational problem in E_i, the problem must be first split into a disjunction of problems obtained by variable identification. A variable identification is just a substitution whose range is a set of variables.

Definition 11.7 An *identification* on a set of variables V is an idempotent substitution ξ such that $\mathcal{D}om(\xi) \subseteq V$ and $\mathcal{R}an(\xi) \subseteq V$. The set of all identifications on V is denoted by ID_V.

• An additional problem occurs due to the fact that cycling equations between the two languages may appear and must be solved. For instance, if $(x_1 =^? t_1[x_2])$ is solved in the first theory (where x_2 is considered as a free constant symbol) and $(x_2 =^? t_2[x_1])$ is solved in the second (where x_1 is considered as a free constant symbol), their propagation yields a cycle. This problem is avoided by a priori choosing a linear ordering on the set $V \cup C$ of all variables and constants introduced in the problem. Then to each constant a is associated a set of variables $V_a = \{x \mid x \in V \text{ and } x < a\}$. Solving a problem of unification with linear restriction is finding unifiers σ s.t. $\forall x, a$ with $x \in V_a$, then a does dot occur in $\sigma(x)$. So in each theory a unification algorithm with linear restriction is used to solve the equational problem Γ_i.

Definition 11.8 Let $<$ be a linear ordering on $V_1 \oplus V_2$ the disjoint union of two finite sets of variables and an equational problem Γ_i such that $\mathcal{V}(\Gamma_i) \subseteq V_1 \oplus V_2$. A *$E_i$-unifier with linear restriction* of Γ_i is a substitution σ such that

- $\forall x_j \in V_j\ , \sigma(x_j) = x_j$,
- $\forall x_j \in V_j,\ \forall x_i \in V_i,\ x_j \notin \sigma(x_i)$ if $x_i < x_j$.

The set of all these unifiers is denoted by $SS^<_{E_i}(\Gamma_i, V_j)$. Variables in $V_j, j \neq i$ are said *frozen.*

Definition 11.9 A set of substitutions is a *complete set of E_i-unifiers with linear restriction* of the problem Γ_i, denoted by $CSS^<_{E_i}(\Gamma_i, V_j)$, if

1. $\forall \sigma \in CSS^<_{E_i}(\Gamma_i, V_j),\ \mathcal{D}om(\sigma) \cap \mathcal{VR}an(\sigma) = \emptyset$ and $\mathcal{D}om(\sigma) \subseteq \mathcal{V}(\Gamma_i)$.
2. $CSS^<_{E_i}(\Gamma_i, V_j) \subseteq SS^<_{E_i}(\Gamma_i, V_j)$.
3. For any $\phi \in SS^<_{E_i}(\Gamma_i, V_j)$, there exists $\sigma \in CSS^<_{E_i}(\Gamma_i, V_j)$ such that $\sigma \leq^{\mathcal{V}(\Gamma_i)}_{E_i} \phi$.

Two solutions in each component must be combined to get a solution in the combined language. A combined solution is obtained from two partial solutions by transforming a dag solved form (the union of the two parts) into a tree solved form where replacement has been performed.

Definition 11.10 The *combined solution* $\sigma = \sigma_1 \odot \sigma_2$ of $\Gamma_1 \wedge \Gamma_2$ obtained from $\sigma_1 \in SS^{<}_{E_1}(\Gamma_1, V_2)$ and $\sigma_2 \in SS^{<}_{E_2}(\Gamma_2, V_1)$ is defined as follows: let x be a variable with theory index i and $\{y_1, \ldots, y_n\}$ be the set of (smaller) variables with theory index $j \neq i$ occurring in $\sigma_i(x)$. Then

$$\sigma(x) = \sigma_i(x)[y_k \hookleftarrow \sigma(y_k)]_{k=1,\ldots,n}.$$

Proposition 11.9 (Correctness) A combined solution $\sigma = \sigma_1 \odot \sigma_2$ of $\Gamma_1 \wedge \Gamma_2$ where $\sigma_1 \in SS^{<}_{E_1}(\Gamma_1, V_2)$ and $\sigma_2 \in SS^{<}_{E_2}(\Gamma_2, V_1)$ is a unifier of $\Gamma_1 \wedge \Gamma_2$ in $E_1 \cup E_2$.

Proof: By assumption, we have $E_1 \models \sigma_1(\Gamma_1)$ and $E_2 \models \sigma_2(\Gamma_2)$, so $E_1 \cup E_2 \models \sigma_1(\Gamma_1)$ and $E_1 \cup E_2 \models \sigma_2(\Gamma_2)$. Then, by construction, $\sigma_i \leq^{\mathcal{V}(\Gamma_1 \wedge \Gamma_2)}_{E_1 \cup E_2} \sigma_1 \odot \sigma_2$ for $i = 1, 2$. So, we have also $E_1 \cup E_2 \models (\sigma_1 \odot \sigma_2)(\Gamma_1)$ and $E_1 \cup E_2 \models (\sigma_1 \odot \sigma_2)(\Gamma_2)$. □

We first give a decidability result: its proof states that for any solution of $\Gamma_1 \wedge \Gamma_2$ in $E_1 \cup E_2$, there exist an identification and a linear restriction for which one can find a solution in each component.

Proposition 11.10 (Completeness) If σ is a R-normalized unifier of $\Gamma_1 \wedge \Gamma_2$ in $E_1 \cup E_2$, then there exist

- $\xi \in ID_{\mathcal{V}(\Gamma_1 \wedge \Gamma_2)}$,
- a linear restriction $<$ on $V_1 \oplus V_2 = \mathcal{V}(\xi(\Gamma_1 \wedge \Gamma_2))$,
- $\sigma_1 \in SS^{<}_{E_1}(\xi(\Gamma_1), V_2),\ \sigma_2 \in SS^{<}_{E_2}(\xi(\Gamma_2), V_1)$

such that $(\sigma_1 \odot \sigma_2) \circ \xi \leq^{\mathcal{V}(\Gamma_1 \wedge \Gamma_2)}_{E_1 \cup E_2} \sigma$.

Proof: We adopt the method used in [BS92] for proving that a unifier in the combined theory provides a unifier with linear restriction in each component theory. Let σ be a R-normalized unifier of $\Gamma_1 \wedge \Gamma_2$. We may assume without loss of generality that $\sigma(x) \neq \sigma(y)$ for two different variables x and y in $\mathcal{V}(\Gamma_1 \wedge \Gamma_2)$. Otherwise x and y have to be first identified by ξ. The variable abstraction π may be chosen as follows: $\pi(\sigma(x)) = x$ for $x \in \mathcal{V}(\Gamma_1 \wedge \Gamma_2)$. In this way, an alien subterm $\sigma(x)$ is abstracted by x. The subterm ordering is used for the linear ordering: $x < y$ if $\sigma(x)$ is a strict subterm of $\sigma(y)$. Let us now consider the substitutions σ_1 and σ_2 defined as follows:

$$\sigma_i = \{x \mapsto \sigma^{\pi_i}(x)\}$$

and the linear restriction on $V_1 \oplus V_2$ with $V_i = \mathcal{D}om(\sigma_i)$. We must verify that σ_1 and σ_2 are unifiers. By assumption, we have

$$E_1 \cup E_2 \models \sigma(\Gamma_1) \wedge \sigma(\Gamma_2)$$

and so

$$E_1 \models \sigma^{\pi_1}(\Gamma_1) \text{ and } E_2 \models \sigma^{\pi_2}(\Gamma_2),$$

according to Proposition 11.8, which means

$$E_1 \models \sigma_1(\Gamma_1) \text{ and } E_2 \models \sigma_2(\Gamma_2),$$

since $\sigma^{\pi_i}(\Gamma_i) = \sigma_i(\Gamma_i)$. These unifiers satisfy the linear restriction: if $y \in \sigma_i(x)$ $(i = 1, 2)$ with $y \in V_j$ $(j \neq i)$ then $\sigma(y)$ is a subterm of $\sigma(x)$. Since $\sigma(x) \neq \sigma(y)$, we have $y < x$. □

Corollary 11.4 Unification in the combined theory $E_1 \cup E_2$ is decidable if unification with linear restriction is decidable in E_1 and E_2.

The last part of the proof amounts to show that combining unifiers in complete sets of s unifiers provides a complete set of unifiers of $\Gamma_1 \wedge \Gamma_2$.

Proposition 11.11 The set of combined solutions of $\Gamma_1 \wedge \Gamma_2$

$$\{\sigma = \sigma_1 \odot \sigma_2 \mid \sigma_1 \in CSS^{<}_{E_1}(\Gamma_1, V_2),\ \sigma_2 \in CSS^{<}_{E_2}(\Gamma_2, V_1)\}$$

is a complete set of combined solutions.

Proof: A similar proof is developped in [BS92]. Let $\sigma' = \sigma'_1 \odot \sigma'_2$ with $\sigma'_i \in SS^{<}_{E_i}(\Gamma_i, V_j)$. There exists $\sigma_i \in CSS^{<}_{E_i}(\Gamma_i, V_j)$ such that $\sigma_i \leq^{V_i}_{E_i} \sigma'_i$. We must show that $\sigma \leq^{V_1 \oplus V_2}_{E_1 \cup E_2} \sigma'$. Let γ_1, γ_2 be the two substitutions verifying

$$\sigma'_i =^{V_i}_{E_i} \gamma_i \sigma_i$$

for $i, j = 1, 2$ and $i \neq j$. These substitutions do not instantiate variables in $V_1 \oplus V_2$: $\mathcal{D}om(\gamma_1) \cap (V_1 \oplus V_2) = \emptyset$ and $\mathcal{D}om(\gamma_2) \cap (V_1 \oplus V_2) = \emptyset$. Moreover, we can assume without loss of generality that γ_1 (resp. γ_2) is idempotent and instantiate two disjoint sets of variables, that is $\gamma_1\gamma_1 = \gamma_1$, $\gamma_2\gamma_2 = \gamma_2$ and $\mathcal{D}om(\gamma_1) \cap \mathcal{D}om(\gamma_2) = \emptyset$.

Let us now prove by nœtherian induction on $<$ that

$$\forall y \in V_1 \oplus V_2, \sigma'(y) =_{E_1 \cup E_2} \gamma_1\gamma_2\sigma(y).$$

This hypothesis holds for the minimal variable z in V_i with $i = 1, 2$ since

$$\sigma'(z) = \sigma'_i(z) =_{E_i} \gamma_i\sigma_i(z) =_{E_1 \cup E_2} \gamma_1\gamma_2\sigma_i(z) = \gamma_1\gamma_2\sigma(z).$$

When this hypothesis holds for all variables $y < x$, the definition of a combined solution states that

$$\begin{array}{rcl}
\sigma'(x) & = & \sigma'_i(x)[y_k \hookleftarrow \sigma'(y_k)]_{k=1,\ldots,n} \\
 & = & \gamma_i\sigma_i(x)[y_k \hookleftarrow \sigma'(y_k)]_{k=1,\ldots,n} \\
 & =_{E_1 \cup E_2} & \gamma_i\sigma_i(x)[y_k \hookleftarrow \gamma_1\gamma_2\sigma(y_k)]_{k=1,\ldots,n} \\
 & = & \gamma_1\gamma_2(\sigma_i(x)[y_k \hookleftarrow \sigma(y_k)]_{k=1,\ldots,n}) \\
 & = & \gamma_1\gamma_2\sigma(x).
\end{array}$$

□

The union of all possible complete sets of combined solutions according to the non-deterministic choice of identifications and linear orderings provides a complete set of unifiers.

Corollary 11.5 Unification in the combined theory $E_1 \cup E_2$ is finitary if unification with linear restriction is finitary in E_1 and E_2.

11.3.5 Rules for unification in the combined theory

The rules for unification in the union of arbitrary disjoint equational theories are given in Figure 11.2. There are mainly two steps called Purification and Combination that produce a symbolic solution in dag-solved form. A third step yields then a tree-solved form by applying as long as possible the Replacement rule. Applying these three steps sequentially obviously terminates and leads to a complete set of solutions.

The deterministic Purification step transforms a problem Γ into an equivalent conjonction of two pure equational problems $\Gamma_1 \wedge \Gamma_2$, each of them solvable in one component.

The non-deterministic Combination step is devoted to solve and combine solutions from each component. The parameters for this last step are:

- A conjunction of i-pure equational problems Γ_i $(i = 1, 2)$.
- A set of variables V_1 instantiated in E_1 and frozen in E_2.
- A set of variables V_2 instantiated in E_2 and frozen in E_1.
- A linear ordering $<$ on the disjoint union $V_1 \oplus V_2$ of V_1 and V_2 which is the occur-check ordering of the expected solution.

The conjonction of tree solved forms obtained at the Combination step leads to a unique tree solved form by applying repeatedly the Replacement rule.

The next result which is derived from [BS92], states that dag solved forms obtained after the Combination step represent a complete set of unifiers.

Theorem 11.2 *Let Γ_1 and Γ_2 be respectively the 1-pure and 2-pure equational problems obtained by purification from the original Γ in $E_1 \cup E_2$. The set of substitutions $(\sigma_1 \odot \sigma_2) \circ \xi$ with all possible*

- *identifications $\xi \in ID_{\mathcal{V}(\Gamma_1 \wedge \Gamma_2)}$,*

1. Purification

   Apply as long as possible the following deterministic rules

   - Variable Abstraction
   $$\frac{P \wedge s =^? t}{\exists x : P \wedge s =^? t[x]_\omega \wedge x =^? t_{|\omega}} \quad \text{if} \begin{cases} t(\omega) \in \mathcal{F}_i,\ t(\epsilon) \in \mathcal{F}_j,\ i \neq j \\ x \text{ is a new variable.} \end{cases}$$

   - Impure Equation
   $$\frac{P \wedge s =^? t}{\exists x : P \wedge x =^? s \wedge x =^? t} \quad \text{if} \begin{cases} s \in \mathcal{T}(\mathcal{F}_1, \mathcal{X})\backslash\mathcal{X},\ t \in \mathcal{T}(\mathcal{F}_2, \mathcal{X})\backslash\mathcal{X} \\ x \text{ is a new variable.} \end{cases}$$

2. Combination of solved forms

   Consider all possible

   - identifications $\xi \in ID_{\mathcal{V}(P_1 \wedge P_2)}$,
   - linear restrictions $<$ on $V_1 \oplus V_2 = \mathcal{V}(\xi(P_1) \wedge \xi(P_2))$

   and apply the following rule

   Solve
   $$\frac{(P_1 \wedge P_2)}{(\hat{\xi} \wedge \hat{\sigma}_1 \wedge \hat{\sigma}_2)} \quad \text{if } \sigma_i \in CSS^{<}_{E_i}(\xi(P_i), V_j)$$

3. Tree solved form

   Replacement
   $$\frac{P \wedge x =^? t}{\{x \mapsto t\}(P) \wedge x =^? t} \quad \text{if } x \in \mathcal{V}(P)$$

Figure 11.2: **The combination rules for unification in $E_1 \cup E_2$**

- *linear restrictions* $<$ *on* $V_1 \oplus V_2 = \mathcal{V}(\xi(\Gamma_1) \wedge \xi(\Gamma_2))$,
- *substitutions* $\sigma_i \in CSS^{<}_{E_i}(\xi(\Gamma_i), V_j)$

is a complete set of $(E_1 \cup E_2)$*-unifiers of* Γ.

Example 11.5 Let $\mathcal{F}_1 = \{*\}$ and $\mathcal{F}_2 = \{h\}$, $E_1 = \{x*(y*z) = (x*y)*z\}$ and $E_2 = \emptyset$. Solve the equational problem $(h(x) * y =^? y * h(z_1 * z_2))$ in $E_1 \cup E_2$.

11.4 Conclusion

A different approach for the problem of cycles has been proposed by Boudet [Bou90b, Bou90a]: assume again that $(x_1 =^? t_1[x_2])$ is solved in the first theory and $(x_2 =^? t_2[x_1])$ is solved in the second. Briefly speaking, the problem is solved a posteriori by an elimination algorithm that for instance tries to find a term u equivalent to an instance of t_2 in the first theory but which does not contain x_1 any more. This allows breaking the cycle and going on.

These results on the unification in the union of equational theories with disjoint signatures have been extended to signatures sharing some constants [Rin92, KR94a], then sharing some constructors [DKR93]. On the other hand, combination techniques have also been extended to the problem of combining two constraint languages and their solvers [KR92, Rin93]. In this context, it is important to allow shared constants in the two domains of interest.

Chapter 12

Syntactic theories

We will show in this chapter how unification procedures can be automatically deduced from the form of the axioms defining the theory, provided the theory has a particular property called syntacticness. This condition can be checked on an appropriate notion of critical pairs and a completion process deduced from this critical pair check. It allows computing a syntactic presentation of the theory from which the mutation operation of an equation can be deduced and thus the appropriate instance of the general unification rules will provide a unification procedure for the theory.

12.1 Syntacticness

12.1.1 Definitions and basic properties

Syntacticness is a property about the form of proofs in an equational theory. In order to define the kind of proofs we are interested in, let us first introduce our notations:

Definition 12.1 Let t, t' be two terms and E be a set of equational axioms.

1. $t \overset{\Lambda}{\longleftrightarrow}_E t'$ denotes a one step proof with one equational replacement at occurrence Λ.
2. $t \overset{\delta}{\longleftrightarrow}\overset{\Lambda}{_E} t'$ denotes a one step proof with either one or no application at occurrence Λ.
3. $t \overset{*}{\longleftrightarrow}\overset{\neq\Lambda}{_E} t'$ denotes a proof without application of an axiom at the top occurrence.
4. $t \overset{*}{\longleftrightarrow}\overset{\delta\Lambda}{_E} t'$ denotes a proof with at most one application of an axiom at the top occurrence, i.e $t \overset{*}{\longleftrightarrow}\overset{\neq\Lambda}{_E} \overset{\delta}{\longleftrightarrow}\overset{\Lambda}{_E} \overset{*}{\longleftrightarrow}\overset{\neq\Lambda}{_E} t'$.

If for example $E = \{a = b\}$ then: $a \overset{\Lambda}{\longleftrightarrow}_E b$ and $g(a) \overset{1}{\longleftrightarrow}\overset{\neq\Lambda}{_E} g(b)$.

The following notion of syntacticness has been introduced by C. Kirchner [Kir85a] and is so called because, as we will see, it allows to describe a unification procedure using the *syntactic* form of an appropriate presentation of the theory.

Definition 12.2 A set of axioms E is *syntactic* or *resolvent* when:

$$\overset{*}{\longleftrightarrow}_E \subseteq \overset{*}{\longleftrightarrow}\overset{\neq\Lambda}{_E} \overset{\delta}{\longleftrightarrow}\overset{\Lambda}{_E} \overset{*}{\longleftrightarrow}\overset{\neq\Lambda}{_E} .$$

An equational theory is *syntactic* when it admits a *finite* and syntactic presentation. A proof is *syntactic* when it has only one application of an axiom at the top occurrence.

Let us insist on the finiteness condition on the presentation in the previous definition of a syntactic *theory*. Without this condition, any equational equational theory would be syntactic since the whole theory itself (which is in general an infinite set of equational axioms) could be chosen as resolvent presentation. On the contrary, a resolvent presentation may be finite or not.

Example 12.1

1– The previous theory $E = \{a = b\}$ is clearly syntactic.

2– Another less immediate example is the associativity theory for which we will prove that the standard presentation $\mathsf{A}(+)$ is syntactic. For example, the following proof:

$$\begin{array}{rlcll} t = & ((a+(b+c))+d)+(e+f) & \overset{\Lambda}{\longleftrightarrow}_{\mathsf{A}} & (a+(b+c))+(d+(e+f)) & \overset{1}{\longleftrightarrow}_{\mathsf{A}} \\ & ((a+b)+c)+(d+(e+f)) & \overset{\Lambda}{\longleftrightarrow}_{\mathsf{A}} & (a+b)+(c+(d+(e+f))) & = t', \end{array}$$

with two steps at the top occurrence can be transformed into the following syntactic proof:

$$\begin{array}{rlcll} t = & ((a+(b+c))+d)+(e+f) & \overset{11}{\longleftrightarrow}_{\mathsf{A}} & (((a+b)+c)+d)+(e+f) & \overset{1}{\longleftrightarrow}_{\mathsf{A}} \\ & ((a+b)+(c+d))+(e+f) & \overset{\Lambda}{\longleftrightarrow}_{\mathsf{A}} & (a+b)+((c+d)+(e+f)) & \overset{2}{\longleftrightarrow}_{\mathsf{A}} \\ & (a+b)+(c+(d+(e+f))) & = t'. & & \end{array}$$

3– As this will be proven formally later, another simple example of syntactic theory is the commutative theory $\mathsf{C}(+)$.

Before coming to the first properties of syntactic theories, let us introduce the vectorial notation that we will be using in this chapter in order to ease the expression of formulas. We are denoting $\vec{t}$ a tuple of terms $(t_1, \dots, t_n)$ and its length n is written $|\vec{t}|$. By extension and when clear from the context $\vec{t}$ may also represent the set $\{t_1, \dots, t_n\}$. A typical use of this syntactic facility is to note the term $f(t_1, \dots, t_n)$ by $f\vec{t}$. Note that the natural n is in general dependent of the context and unspecified explicitly; usually we do not really care about its precise value. This notation is extended to equations in the following way: $\vec{v} =^? \vec{u}$ with $|\vec{u}| = |\vec{v}| = n$, denotes the system $v_1 =^? u_1 \wedge \dots \wedge v_n =^? u_n$.

We can now state the first basic property of syntactic theories which is that equality is decomposable:

Lemma 12.1 Let E be a set of equational axioms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$. The theory $\mathcal{TH}(E)$ is syntactic if and only if the following equivalence is satisfied for each E-equality:

$$f(\vec{t}) =_E g(\vec{u}) \Leftrightarrow \begin{cases} \bullet \begin{cases} f = g \text{ and} \\ \quad \forall i \in [1..\text{arity}(f)],\ t_i =_E u_i \end{cases} \\ \text{or} \\ \bullet \begin{cases} \exists l = r \in E,\ \exists \sigma \text{ such that} \\ \quad f = \sigma l(\Lambda) \\ \quad \text{and} \\ \quad \sigma r(\Lambda) = g \\ \quad \text{and} \\ \quad \forall i \in [1..\text{arity}(f)],\ t_i =_E \sigma l_{|i} \\ \quad \text{and} \\ \quad \forall i \in [1..\text{arity}(g)],\ \sigma r_{|i} =_E u_i \end{cases} \end{cases}$$

with $f, g \in \mathcal{F} \cup \mathcal{X}$.

Proof: Follows directly from the definition of syntacticity. □

Example 12.2 This last result implies in particular that almost-free theories (c.f. Section 2.4.8) with equational axioms of the form $f(\vec{u}) = f(\vec{v})$, are syntactic.

Syntacticness can also be characterized precisely by the form of the E-equality proof of two terms: E is resolvent iff for any two equal terms t and t' there exists a proof of this equational theorem with at most one application of an axiom at the top occurrence. Formally:

Proposition 12.1 A presentation E in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is resolvent if and only if:

$$\overset{\Lambda}{\longleftrightarrow}_E \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_E \overset{\Lambda}{\longleftrightarrow}_E \subseteq \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_E \overset{\delta}{\longleftrightarrow}{}^{\Lambda}_E \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_E .$$

Proof: If E is syntactic, then the conclusion is clear by the definition of syntacticness.

Conversely, assuming that a presentation satisfy the previous condition, one can eliminate by induction all the couples of consecutive $\overset{\Lambda}{\longleftrightarrow}_E$ steps, which results in a syntactic proof. □

Collapse axioms play a particular role with respect to syntacticness.

Lemma 12.2 Let E be a consistent presentation in $\mathcal{T}(\mathcal{F},\mathcal{X})$. If $l = x$ with $x \in \mathcal{X}$ is a collapse axiom applied at the top position in a proof, i.e. if $t \longleftrightarrow^{\Lambda}_{l=x} t' \longleftrightarrow_{g=d} u$ then there exists a syntactic proof where this collapse axiom is applied at the end:

$$t \overset{n}{\longleftrightarrow}^{\neq\Lambda}_{g=d} t\text{''} \longleftrightarrow^{\Lambda}_{l=x} u,$$

where n is the number of disjunct positions of the variable x in l.

Proof: Clear application of the definitions. $\square$

It is also useful to precise how non-syntactic theories behave with respect to the proof form. In order to state the next result we need to introduce the following definition:

Definition 12.3 For a given presentation E in $\mathcal{T}(\mathcal{F},\mathcal{X})$, a proof $t' \overset{n}{\longleftrightarrow}^{\omega_i}_{E} u'$ is a *subproof* of the proof $t \overset{*}{\longleftrightarrow}_E u$ if:

$$t \overset{*}{\longleftrightarrow}_E s[t']_p \overset{n}{\longleftrightarrow}^{p.\omega_i}_{E} s[u']_p \overset{*}{\longleftrightarrow}_E u$$

is a proof which can be obtained from $t \overset{*}{\longleftrightarrow}_E u$ by commuting axiom applications at disjunct positions.

Lemma 12.3 Let E be a presentation in $\mathcal{T}(\mathcal{F},\mathcal{X})$. If E is non-syntactic then there exist two terms t, u and a proof:

$$t \longleftrightarrow^{\Lambda}_{l=f(\vec{r})} t' \overset{*}{\longleftrightarrow}^{\neq\Lambda}_{E} u' \longleftrightarrow^{\Lambda}_{f(\vec{g})=d} u,$$

such that $t =_E u$ has no syntactic proof and any subproof of $t' =_E u'$ is syntactic.

Proof: Since E is not syntactic there exists a proof

$$t \longleftrightarrow^{\Lambda}_{l=l'} t' \overset{n}{\longleftrightarrow}^{\neq\Lambda}_{E} u' \longleftrightarrow^{\Lambda}_{d'=d} u, \tag{12.1}$$

that has no syntactic proof.
Because of lemma 12.2, l' and d' are not variables and thus the proof 12.1 is of the form:

$$t \longleftrightarrow^{\Lambda}_{l=f(\vec{r})} t' \overset{n}{\longleftrightarrow}^{\neq\Lambda}_{E} u' \longleftrightarrow^{\Lambda}_{f(\vec{g})=d} u. \tag{12.2}$$

Let us now prove that there exists a proof like 12.2 such that any of the subproofs of $t' =_E u'$ is syntactic. Thus let us consider such a proof of minimal length and assume by contradiction that there exists a non syntactic subproof of $t' =_E u'$. If $n = 0$ or $n = 1$ then this is clearly impossible. If $n > 1$, then there exists a subproof of $t' =_E u'$ of the form

$$v \longleftrightarrow^{\Lambda}_{E} v' \overset{k}{\longleftrightarrow}^{\neq\Lambda}_{E} r' \longleftrightarrow^{\Lambda}_{E} r$$

such that $k < n-3$. But then we have found a smaller proof than the one considered, which contradicts the hypothesis of minimality on the length of the proof considered. $\square$

12.1.2 Undecidability results

In his thesis F. Klay [Kla92], has shown that all the general properties concerning syntactic theories are undecidable. The following results summarize these negative properties:

Theorem 12.1 *[Kla92]*
For a presentation E in $\mathcal{T}(\mathcal{F},\mathcal{X})$ and its associated equational theory $\mathcal{TH}(E)$ it is undecidable if:

1. *E is resolvent,*
2. *$\mathcal{TH}(E)$ is syntactic.*

Moreover even when a theory is syntactic, collapse free and linear, the word problem is not decidable in general.

12.2 Unification in syntactic theories

We are mainly interested in syntactic theories since they allow a form of mutation directly deduced from the form of the axioms. This mutation can be considered as a more general form of the decomposition we have introduced above for the syntactic theory, or conversely we can now see the decomposition rule as the mutation operation for the empty theory.

Given a syntactic presentation, in order to express the mutation rules, we need to introduce a new unification symbol that allows to precise the positions where the equality steps are allowed during the solving process.

Definition 12.4 For a set of equational axioms E and two terms t and t', the solutions of the equation $t =^{\neq\Lambda ?} t'$ are as follow:

$$Sol_E(t =^{\neq\Lambda ?} t') = \{\sigma \mid \text{there exists a proof } \sigma(t) \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_{E} \sigma(t')\}.$$

The rules presented in Figure 12.1 give a version of the most general mutation rule for unification in syntactic theories. We give them without reference to the system P and disjunction D in which the redexes appear.

Example 12.3 If we consider for E the single axiom of commutativity of the symbol $+$, then by application of **Mut2** to equation $e = (u + v =^? u' + v')$ we get:

$$\left[\begin{array}{ll} \left\{\begin{array}{l} u =^? u' \\ v =^? v' \end{array}\right. & \\ \left\{\begin{array}{l} x =^? v \\ y =^? u \\ x =^? u' \\ y =^? v' \end{array}\right. & \text{(decomposition w.r.t. } y + x = x + y) \\ \left\{\begin{array}{l} x =^? u \\ y =^? v \\ x =^? v' \\ y =^? u' \end{array}\right. & \text{(decomposition w.r.t. } x + y = y + x) \end{array}\right.$$

These mutations rules preserves the completeness provided that the set of axioms E is resolvent as stated by the following result:

Theorem 12.2 *If E is a resolvent set of axioms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, then the rules in* **SyntacticMutation** *preserve the set of E-unifiers.*

Proof: This follows almost directly from the definition of syntacticness. A complete proof can be found in [Kla92]. □

What is moreover interesting is that in fact the previous set of rules allows to characterize syntactic theories:

Proposition 12.2 For a given set of axioms E in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, if the rules in **SyntacticMutation** preserve the set of E-unifiers of any unification problem then the presentation E is resolvent.

Proof: By contradiction: Assuming that the rules in **SyntacticMutation** do not preserve the set of E-unifiers allows to build an equational theorem with no syntactic proof in E, and this contradicts the hypothesis. □

Since the previous set of transformation rules gives us a mutation for syntactic theories it is natural to complete the mutation rules with merging and replacement rules in order to get a unification procedure. But the difficulty at this stage is to prove that the resulting set of rules is terminating, i.e. that it will allow to find by normalization a solved form representing a substitution smaller that any given E-unifier. Indeed this is still an open problem which intuitive difficulty relies on the fact that the replacement and merging rules do not preserve the form of the proof. For example there is a priori no direct connection between the syntactic proofs of $\sigma(x) =_E \sigma(t)$ and $\sigma(x) =_E \sigma(t')$ and a syntactic proof of $\sigma(t) =_E \sigma(t')$. Solving this problem will help in proving improvements of the Gallier-Snyder general unification procedure that we are describing in Section 14.1.1.

Nevertheless, we have at least a unification procedure for syntactic theories and we will show later useful examples of application of it. We are now describing in the following section tools to prove, if possible in an automatic way, that a theory is syntactic.

Dec1 $f(\vec{t}) =^{\neq\Lambda?} f(\vec{u})$

$\mapstochar\twoheadrightarrow$

$\vec{t} =^? \vec{u}$

Dec2 $f(\vec{t}) =^{\neq\Lambda?} x$

$\mapstochar\twoheadrightarrow$

$\exists\vec{v},\ \vec{t} =^? \vec{v} \wedge f(\vec{v}) =^{\neq\Lambda?} x$

if $x \in \mathcal{V}ar(f(\vec{t}))$

Mut1 $f(\vec{t}) =^? f(\vec{u})$

$\mapstochar\twoheadrightarrow$

$$\begin{array}{lr}
\vec{t} =^? \vec{u} & \vee \\
\bigvee_{f(\vec{l})=f(\vec{r})\in E} \exists\mathcal{V}ar(\vec{l},\vec{r}),\ (\vec{t} =^? \vec{l} \wedge \vec{r} =^? \vec{u}) & \vee \\
\bigvee_{f(\vec{l})=y\in E} \exists\mathcal{V}ar(\vec{l}),\ (\vec{t} =^? \vec{l} \wedge y =^{\neq\Lambda?} f(\vec{u})) & \vee \\
\bigvee_{y=f(\vec{r})\in E} \exists\mathcal{V}ar(\vec{r}),\ (f(\vec{t}) =^{\neq\Lambda?} y \wedge \vec{r} =^? \vec{u}) &
\end{array}$$

Mut2 $f(\vec{t}) =^? g(\vec{u})$

$\mapstochar\twoheadrightarrow$

$$\begin{array}{lr}
\bigvee_{f(\vec{l})=g(\vec{r})\in E} \exists\mathcal{V}ar(\vec{l},\vec{r}),\ (\vec{t} =^? \vec{l} \wedge \vec{r} =^? \vec{u}) & \vee \\
\bigvee_{f(\vec{l})=y\in E} \exists\mathcal{V}ar(\vec{l}),\ (\vec{t} =^? \vec{l} \wedge y =^{\neq\Lambda?} g(\vec{u})) & \vee \\
\bigvee_{y=g(\vec{r})\in E} \exists\mathcal{V}ar(\vec{r}),\ (f(\vec{t}) =^{\neq\Lambda?} y \wedge \vec{r} =^? \vec{u}) &
\end{array}$$

if $f \neq g$

Cycle $f(\vec{t}) =^? x$

$\mapstochar\twoheadrightarrow$

$$\begin{array}{lr}
\exists\vec{v},\ (\vec{t} =^? \vec{v} \wedge f(\vec{v}) =^{\neq\Lambda?} x) & \vee \\
\bigvee_{f(\vec{l})=g(\vec{r})\in E} \exists\mathcal{V}ar(\vec{l},\vec{r}),\ (\vec{t} =^? \vec{l} \wedge g(\vec{r}) =^{\neq\Lambda?} x) & \vee \\
\bigvee_{f(\vec{l})=y\in E} \exists\mathcal{V}ar(\vec{l}),\ (\vec{t} =^? \vec{l} \wedge x =^{\neq\Lambda?} y) & \vee \\
\bigvee_{y=g(\vec{r})\in E} \exists\mathcal{V}ar(\vec{r}),\ (f(\vec{t}) =^{\neq\Lambda?} y \wedge g(\vec{r}) =^{\neq\Lambda?} x) &
\end{array}$$

if $x \in \mathcal{V}ar(f(\vec{t}))$

Figure 12.1: **SyntacticMutation**: Rules for syntactic mutation

12.3 General Equations

In this section, we investigate the relationship between unifiability of certain kind of equations of the form $f(v_1, \ldots, v_n) =^? g(v'_1, \ldots, v'_m)$ that are called general, and the syntacticness of an equational theory $\mathcal{TH}(E)$. The main theorem, states that a theory is syntactic if and only if the general equations have *finite* complete set of E-solutions. This result is constructive in the sense that from the E-solutions of the general equations, a resolvent presentation is computed. This is applied to several theories both on the positive and negative sides, in particular in order to show that distributivity is not syntactic. On the contrary we will see that the theory of associativity and commutativity is syntactic, which allows to infer a new matching algorithm where there is no need to solve directly linear Diophantine equations [Hul79].

12.3.1 Definition

General equations are the most general kind of equation that one can built on a given signature, this justifies their name.

Definition 12.5 Let E be a set of equational axioms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $V = \{v_1, \ldots, v_n, v'_1, \ldots, v'_m\}$ a set of (of course distinct) variables, n and m the arities of the symbols f and g in $\mathcal{F}$. A *general unification equation* is of the form:

$$f(v_1, \ldots, v_n) =^? g(v'_1, \ldots, v'_m)$$

denoted $GenU(f, g, V)$. For $V = \{v, v_1, \ldots, v_n\}$, a *general matching equation* is of the form:

$$f(v_1, \ldots, v_n) \ll^? v,$$

It is denoted $GenM(f, V)$.

When it is clear from the context, we speak of *general equation*. For W a set of variables containing V, we denote $CU_E^W(f, g, V)$ a complete set of E-unifiers, away from W, of the general equation equation $f(v_1, \ldots, v_n) =^? g(v'_1, \ldots, v'_m)$. Similarly $CM_E^W(f, V)$ denotes a complete set of E-matches, away from W, of the general equation $f(v_1, \ldots, v_n) =^? v$.

Example 12.4 If we consider the set of axioms $\mathsf{C}(+)$ then $CU^W_{\mathsf{C}(+)}(+, +, V)$ is the minimal complete set of unifiers of the general equation $v_1 + v_2 =^? v_3 + v_4$ and it is equal to:

$$CU^W_{\mathsf{C}(+)}(+, +, V) = \left\{ \begin{array}{l} \left\{ \begin{array}{l} v_1 \mapsto x_1 \\ v_2 \mapsto x_2 \\ v_3 \mapsto x_1 \\ v_4 \mapsto x_2 \end{array} \right. \\ \\ \left\{ \begin{array}{l} v_1 \mapsto x_1 \\ v_2 \mapsto x_2 \\ v_3 \mapsto x_2 \\ v_4 \mapsto x_1 \end{array} \right. \end{array} \right.$$

The solutions of general equations allow to describe a mutation transformation that hopefully may simplify the system. This is stated in the next result:

Lemma 12.4 Let E be a set of equational axioms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $e = (f(\vec{t}) =^? g(\vec{u}))$ a unification problem and $f(v_1, \ldots, v_n) =^? g(v'_1, \ldots, v'_m)$ the corresponding general equation. If W is a set of variables such that $W \subseteq \mathcal{V}ar(e)$ then the following transformation preserve the set of solutions:

$$\textbf{Mut} \quad f(\vec{t}) =^? g(\vec{u}) \quad \mapsto\!\!\!\!\rightarrow \quad \bigvee_{\sigma \in CU_E^W(f,g,V)} \exists (\vec{v}, \vec{v'}, \mathcal{R}an(\sigma)),\ \vec{t} =^? \sigma(\vec{v}) \wedge \sigma(\vec{v'}) =^? \vec{u}.$$

Proof: Variable abstraction preserve the set of solutions thus:

$$f(\vec{t}) =^? g(\vec{u}) \Leftrightarrow_E \exists (\vec{v}, \vec{v'}),\ f(\vec{v}) =^? g(\vec{v'}) \wedge \vec{v} =^? \vec{t} \wedge \vec{v'} =^? \vec{u}.$$

Then, $f(\vec{v}) =^? g(\vec{v'})$ can be replaced by its solved form and we get;

$$\begin{array}{ll} f(\vec{t}) =^? g(\vec{u}) & \\ \Leftrightarrow_E & \\ \exists (\vec{v}, \vec{v'}), & (\bigvee_{\sigma \in CU_E^W(f,g,V)} \exists \mathcal{R}an(\sigma), \vec{v} =^? \sigma(\vec{v}) \wedge \vec{v'} =^? \sigma(\vec{v'})) \wedge \\ & \vec{v} =^? \vec{t} \wedge \vec{v'} =^? \vec{u}. \end{array}$$

Equivalence is preserved because of the hypothesis made on W and the variables in consideration that avoid any conflict. Then the last problem is normalized in its disjunctive normal form and merged if necessary, and we get the result. □

Exercice 44 — Apply the previous result to the case of commutativity using Example 12.4. What additional transformation of the mutation rule is needed to obtain **ComMutate** given in the set of rules **CommutativeUnification**?
Answer:

12.3.2 Unifiers of general equations

In this section we are describing the E-solutions of general equations when the presentation E is resolvent. The dual operation, consisting to deduce from the solutions of general equations a syntactic presentation, will be described in the next section.

Definition 12.6 Let E be a set of axioms in $\mathcal{T}(\mathcal{F},\mathcal{X})$, $V = \{v_1, v'_1, v_2, v'_2, \ldots\}$ and $W = \{w_1, w_2, \ldots\}$ two countable sets of distinct new variables. For all f, g in $\mathcal{F}$ we define the set of substitutions

$$\Sigma^{\mathcal{U}}(f,g,E,V,W) = \Sigma^{\mathcal{U}}_c(f,g,E,V,W) \cup \Sigma^{\mathcal{U}}_{nc}(f,g,E,V,W)$$

as follows:
• If $f \neq g$ then:

$$\begin{array}{lcl}\Sigma^{\mathcal{U}}_{nc}(f,g,E,V,W) & = & \{\{\vec{v} \mapsto \vec{l}, \vec{v}' \mapsto \vec{r}\} | f(\vec{l}) = g(\vec{r}) \in E \cup E^{-1}\} \\ \Sigma^{\mathcal{U}}_{c}(f,g,E,V,W) & = & \{\{\vec{v} \mapsto (\{x \mapsto g(\vec{w})\}(\vec{l})), \vec{v}' \mapsto \vec{w}\} | f(\vec{l}) = x \in E \cup E^{-1}\} \\ & & \cup \\ & & \{\{\vec{v}' \mapsto (\{x \mapsto f(\vec{w})\}(\vec{r})), \vec{v} \mapsto \vec{w}\} | x = g(\vec{r}) \in E \cup E^{-1}\}\end{array}$$

• If $f = g$ then:

$$\begin{array}{lcl}\Sigma^{\mathcal{U}}_{nc}(f,f,E,V,W) & = & \{\{\vec{v} \mapsto \vec{l}, \vec{v}' \mapsto \vec{r}\} | f(\vec{l}) = f(\vec{r}) \in E \cup E^{-1}\} \\ & & \cup \\ & & \{\vec{v} \mapsto \vec{w}, \vec{v}' \mapsto \vec{w}\} \\ \Sigma^{\mathcal{U}}_{c}(f,f,E,V,W) & = & \{\{\vec{v} \mapsto (\{x \mapsto f(\vec{w})\}(\vec{l})), \vec{v}' \mapsto \vec{w}\} | f(\vec{l}) = x \in E \cup E^{-1}\} \\ & & \cup \\ & & \{\{\vec{v}' \mapsto (\{x \mapsto f(\vec{w})\}(\vec{r})), \vec{v} \mapsto \vec{w}\} | x = f(\vec{r}) \in E \cup E^{-1}\}\end{array}$$

Example 12.5 A mettre

The following property is an important consequence of the definitions above:

Lemma 12.5 [Kla92]
If the presentation E in $\mathcal{T}(\mathcal{F},\mathcal{X})$ is resolvent then for all $f, g \in \mathcal{F}$, the set $\Sigma^{\mathcal{U}}(f,g,E,V,W)$ is a complete set of E-unifiers of the general equation $GenU(f,g,V)$.

Proof: Let $e = GenU(f,g,V)$. Any substitution $\sigma \in \Sigma^{\mathcal{U}}(f,g,E,V,W)$ is an E-unifier of e since $\sigma(e)$ is either an axiom or an instance of a collapse axiom.
In order to prove completeness, let α be any E-unifier of e. Since E is resolvent, there exists a syntactic proof of $\alpha(e)$:

$$\alpha(f(\vec{v})) \overset{\neq\Lambda}{\underset{E}{\overset{*}{\longleftrightarrow}}} f(\vec{t}) \overset{\Lambda}{\underset{E}{\overset{\delta}{\longleftrightarrow}}} g(\vec{u}) \overset{\neq\Lambda}{\underset{E}{\overset{*}{\longleftrightarrow}}} \alpha(g(\vec{v}')). \qquad (12.3)$$

• If $\delta = 0$ then $f = g$ and $\alpha(\vec{v}) =_E \alpha(\vec{v}')$. In this case the substitution

$$\sigma = \{\vec{v} \mapsto \vec{w}, \vec{v}' \mapsto \vec{w}\} \in \Sigma^{\mathcal{U}}_{nc}(f,f,E,V,W)$$

satisfy for $\rho = \{\vec{w} \mapsto \alpha(\vec{v})\}$:

$$\alpha(\vec{v}') =_E \alpha(\vec{v}) = \rho\sigma(\vec{v}) = \rho\sigma(\vec{v}'),$$

and thus $\sigma \leq_E^{\mathcal{V}ar(e)} \alpha$.
• If $\delta = 1$ then let $l = r$ be the equational axiom applied at occurrence Λ using the substitution ϕ is the proof 12.3.

- If $l = r$ is a non collapse axiom of the form $f(\vec{l}) = g(\vec{r})$ then $\alpha(\vec{v}) =_E \phi(\vec{l})$ and $\phi(\vec{r}) =_E \alpha(\vec{v}')$. With the substitution:
$$\sigma = \{\vec{v} \mapsto \vec{l}, \vec{v}' \mapsto \vec{r}\} \in \Sigma^{\mathcal{U}}_{nc}(f, g, E, V, W)$$
we can write:
$$\alpha(\vec{v}) =_E \phi(\vec{l}) = \phi\sigma(\vec{v})$$
and,
$$\alpha(\vec{v}') =_E \phi(\vec{r}) = \phi\sigma(\vec{v}'),$$
and thus $\sigma \leq_E^{\mathcal{V}ar(e)} \alpha$.
- If $l = r$ is a collapse axiom of the form $f(\vec{l}) = x$, then let us take the substitution:
$$\phi' = \phi_{|\mathcal{V}ar(l)\setminus\{x\}} \cup \{\vec{w} \mapsto \vec{u}\}$$
where the $\vec{w}$ are new distinct variables. This substitution is such that $\phi =^{\mathcal{V}ar(\vec{l})} \phi'.\{x \mapsto g(\vec{w})\}$ and thus $\alpha(\vec{v}) =_E \phi'.\{x \mapsto g(\vec{w})\}(\vec{l})$ and $\phi'(\vec{w}) = \vec{u} =_E \alpha(\vec{v}')$. Now, considering the substitution:
$$\sigma = \{\vec{v} \mapsto (\{x \mapsto g(\vec{w})\}(\vec{l})), \vec{v}' \mapsto \vec{w}\} \in \Sigma^{\mathcal{U}}_{c}(f, g, E, V, W),$$
we get:
$$\alpha(\vec{v}) =_E \phi(\vec{l}) = \phi'.\{x \mapsto g(\vec{w})\}(\vec{l}) = \phi'\sigma(\vec{v}),$$
and
$$\alpha(\vec{v}') =_E \phi'(\vec{w}) = \phi'\sigma(\vec{v}'),$$
which proves that $\sigma \leq_E^{\mathcal{V}ar(e)} \alpha$.
Finally if the collapse axiom is applied in the other direction the proof is similar.

□

In a similar way we are introducing the solutions of general matching equations:

Definition 12.7 Let E be a set of axioms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and $V = \{v, v_1, v_2, \ldots\}$ a countable set of distinct new variables. For all f in $\mathcal{F}$ we define the set of substitutions:
$$\Sigma^{\mathcal{M}}(f, E, V) = \{\{\vec{v} \mapsto (\{x \mapsto v\}(\vec{l}))\} \mid f(\vec{l}) = x \in E \cup E^{-1}\}.$$

As for general unification equations we get a completeness result. The proof technique is quite similar to those of the previous lemma and we do not give it here.

Lemma 12.6 [Kla92]
Let E be a resolvent set of axioms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, then for all f in $\mathcal{F}$, $\Sigma^{\mathcal{M}}(f, E, V)$ is a complete set of E-matches of the general match equation $GenM(f, V) = (f(v_1, \ldots, v_n) \ll^? v)$.

From the two previous results we can now deduce:

Corollary 12.1 Let $\mathcal{TH}(E)$ be a syntactic theory. Then all the general equations have a finite complete set of E-unifiers.

Proof: This follows from the fact that the theory being syntactic, there exists a finite and resolvent presentation of it. By application of the previous lemmas, we thus get a finite number of elements in the complete set of E-unifiers. □

12.3.3 General equations and syntacticness

We have seen in the previous section how to compute the E-unifiers of general equations (either unification or matching equations) from a resolvent presentation. We are of course interested in the dual search of a resolvent presentation starting from a complete set of E-unifiers of general equations. This is the purpose of this section which main results come from [KK90, Kla92].

The following set of axioms is defined as the instances of the general equations by the elements of one of its complete set of E-unifiers.

Definition 12.8 If E is a set of equational axioms in $\mathcal{T}(\mathcal{F},\mathcal{X})$, we define a *general presentation* $\mathcal{R}(E)$ as the following set of axioms:

$$\mathcal{R}(E) = E \cup \bigcup_{f,g\in\mathcal{F}} \bigcup_{\sigma\in CU_E^W(f,g,V)} \sigma(f(\vec{v})) = \sigma(g(\vec{v}')) \bigcup_{f\in\mathcal{F}} \bigcup_{\sigma\in CM_E^W(f,V)} \sigma(f(\vec{v})) = v.$$

From this definition follows immediately the next very useful property:

Proposition 12.3 The equational theory generated by E and any of its general presentation $\mathcal{R}(E)$ are the same: $\mathcal{TH}(E) = \mathcal{TH}(\mathcal{R}(E))$.

Proof: By definition of $CU_E^W(f,g,V)$ and $CM_E^W(f,V)$, all the elements in $\mathcal{R}(E)$ that are not in E are instance of a general equation by one of its E-unifier, thus they are equal modulo E. □

The fundamental property of general presentations is that they are resolvent.

Proposition 12.4 For E any consistent equational presentation in $\mathcal{T}(\mathcal{F},\mathcal{X})$, $\mathcal{R}(E)$ is resolvent.

Proof: Let us consider each of the possible cases for an equality in E:

- If it is of the form $f(\vec{t}) =_E g(\vec{u})$ then the substitution $\alpha = \{\vec{v} \mapsto \vec{t}, \vec{v}' \mapsto \vec{u}\}$ is a E-unifier of the general equation $f(\vec{v}) =^? g(\vec{v}')$. Thus there exists $\sigma \in CU_E^W(f,g,V)$, with $V = \{v_1, v_1', \ldots, v_n, v_n'\}$ such that $\sigma \leq_E^V \alpha$, which means that there exists ρ such that $\rho\sigma =_E^V \alpha$. By definition of $\mathcal{R}(E)$ there exists a proof:

$$f(\vec{t}) = \alpha f(\vec{v}) \overset{*}{\longleftrightarrow}{}_E^{\neq\Lambda} \rho\sigma f(\vec{v}) \longleftrightarrow_{\mathcal{R}(E)}^{\Lambda} \rho\sigma g(\vec{v}') \overset{*}{\longleftrightarrow}{}_E^{\neq\Lambda} \alpha g(\vec{v}') = g(\vec{u}),$$

  which is resolvent by definition.
- If it is of the form $f(\vec{t}) =_E x$, then the proof works in a similar manner using the appropriate general matching equation.
- The only remaining case is $x =_E x$ since E is assumed to be consistent, and this case is trivial.

□

As a clear consequence we get:

Corollary 12.2 Let E be a presentation in $\mathcal{T}(\mathcal{F},\mathcal{X})$. If all the general equations have a finite complete set of E-unifiers then $\mathcal{TH}(E)$ is a syntactic theory.

We can be more precise in the sense that only general unification equations have to be considered:

Proposition 12.5 Let E be a presentation in $\mathcal{T}(\mathcal{F},\mathcal{X})$, if unification is finitary for E then $\mathcal{TH}(E)$ is a syntactic theory.

Proof: What we shall prove is that if unification is finitary in E then all the general matching equations are finitary (i.e. have a finite complete set of E-solutions). Let $f(v_1,\ldots,v_n) \ll^? v$ be a general matching equation. Since E-unification is assumed to be finitary there exists a finite complete set Σ of E-unifiers of the equation $f(v_1,\ldots,v_n) =^? v$. Let

$$\Sigma' = \{\{x \mapsto v\}\sigma_{|\vec{v}} \mid \sigma \in \Sigma \text{ and } \sigma(v) =_E x \text{ and } x \in \mathcal{X}\}.$$

Since Σ is finite, Σ' is necessary finite too and it is clearly a set of E-matches of the general matching equation. Moreover it is a complete set of matches since any match is a unifier when the variables of the two members of the equation are distinct. □

Finally let us state the main result:

Theorem 12.3 *Let E be a presentation in $\mathcal{T}(\mathcal{F},\mathcal{X})$, then the two following properties are equivalent:*

1 *$\mathcal{TH}(E)$ is a syntactic theory,*

2 *All the general equations have a finite complete set of E-solutions.*

If the theory is collapse free, this second condition specializes to:

2' *All the general unification equations have a finite complete set of E-unifiers.*

Proof: The only difficulty consists to remark that when the theory is collapse free, then no general matching equation can have E-solutions. □

12.3.4 Applications

As applications of the previous results, let us give two main examples. The first one concerns associative-commutative theories which are syntactic and the second one concerns (left and right) distributivity which is an example of non syntactic theory.

AC theories are syntactic

Considering the case of an operator symbol $+$ satisfying $\mathsf{AC}(+)$, we know from Section 13.1 that AC-unification is finitary. Moreover the (minimal) complete set of AC-unifier of the only general (unification) equation of interest $v_1 + v_2 =^? v'1 + v'2$ is given on page 167. It is thus very easy to compute the following syntactic presentation $\mathcal{R}(AC(+))$ of this theory:

$$\begin{cases} w_1 + w_2 & = \ w_1 + w_2 \\ w_1 + w_2 & = \ w_2 + w_1 \\ w_1 + (w_2 + w_3) & = \ (w_1 + w_3) + w_2 \\ (w_1 + w_2) + w_3 & = \ w_1 + (w_2 + w_3) \\ (w_1 + w_2) + w_3 & = \ (w_1 + w_3) + w_2 \\ w_1 + (w_2 + w_3) & = \ w_3 + (w_1 + w_2) \\ (w_1 + w_3) + (w_2 + w_4) & = \ (w_1 + w_4) + (w_2 + w_3) \end{cases}$$

We can remark that this set is redundant (in particular the first axiom is useless). This presentation of AC theories has been given first in [KK90] and can be used to give an interesting version of the AC-matching algorithm. Its application to AC-unification is still an open problem since the rough mutation transformation generated in this case produces a non terminating procedure.

Distributivity is not syntactic

Left and right distributivity are unitary collapse free theories. They are thus syntactic. Curiously, when put together these two axioms

$$D = \begin{cases} x * (y + z) & = & (x * y) + (x * z) \\ (y + z) * x & = & (y * x) + (z * x) \end{cases}$$

generate a non syntactic theory. Let us show why by proving that the general equation $v_1 * v_2 =^? v_3 * v_4$ is not finitary D-unifying since there is an infinite set Σ which is included in a minimal complete set of D-solutions of this equation. If $\mathbf{P}$ is the set of prime numbers (in $\mathbf{N}$) we have:

$$\Sigma = \bigcup_{i \in P} \begin{cases} v_1 \mapsto \sum_{j=1}^{i} u_1 \\ v_2 \mapsto \ u_2 \\ v_3 \mapsto \ u_1 \\ v_4 \mapsto \sum_{j=1}^{i} u_2 \end{cases}$$

$$where \quad \sum_{j=1}^{n} x = \underbrace{((\cdots((x + x) + x)\cdots + x) + x)}_{n\ times}$$

The substitutions of this set are obviously $\leq_D$-uncomparable and are D-solutions of the general equation $v_1 * v_2 =^? v_3 * v_4$. We now prove that Σ is included in a minimal complete set of D-unifiers of the general equation or in other words that for all $\sigma \in \Sigma$ it does not exist a D-solution ρ such that $\rho <_D \sigma$. By contradiction suppose that such a substitution exists and note the following points:

- Since no axiom can be applied to the terms of the codomain of σ, the terms of the codomain of ρ must have the same structure:

$$\rho = \begin{cases} v_1 \mapsto \sum_{j=1}^{n} x_j \\ v_2 \mapsto y \\ v_3 \mapsto x \\ v_4 \mapsto \sum_{j=1}^{m} y_j \end{cases}$$

- Since ρ is a D-solution of $v_1 * v_2 =^? v_3 * v_4$ we must have:

$$(\sum_{j=1}^{n} x_j) * y =_D x * (\sum_{j=1}^{m} y_j)$$

or equivalently:

$$\sum_{j=1}^{n}(x_j * y) = \sum_{j=1}^{m}(x * y_j)$$

This means that $n = m$ and $\forall j \in [1..n],\ x_j = x$ and $\forall j \in [1..m],\ y_j = y$.

Thus ρ has the form:

$$\rho = \begin{cases} v_1 \mapsto \sum_{j=1}^{n} x \\ v_2 \mapsto y \\ v_3 \mapsto x \\ v_4 \mapsto \sum_{j=1}^{n} y \end{cases}$$

and it is easy to see that there is $\sigma' \in \Sigma$ such that $\sigma' \leq_{\mathsf{D}} \rho$. This leads to a contradiction since the elements of Σ are uncomparable.

So we have an example where a combination of non-disjoint syntactic theories is a non-syntactic theory. T. Nipkow [Nip90a] has shown that if the signatures of the theories are *disjoint* then the combination of syntactic theories is a syntactic theory.

Exercice 45 — Let E be the the following set of equational axioms:

$$A = \{fg(x) = f(x)\}.$$

Prove that the generated theory is non-syntactic because the general equation $f(v_1) =^? f(v_2)$ is not finitary E-unifying.

Hint: Consider the following set of A-solutions of this equation:

$$\sigma_0 = \begin{cases} v_1 & \mapsto & x \\ v_2 & \mapsto & x \end{cases}, \quad \sigma_1 = \begin{cases} v_1 \mapsto x \\ v_2 \mapsto g(x) \end{cases},$$
$$\sigma_2 = \begin{cases} v_1 \mapsto x \\ v_2 \mapsto g(g(x)) \end{cases}, \quad \cdots$$
$$\sigma_n = \begin{cases} v_1 \mapsto x \\ v_2 \mapsto g^n(x) \end{cases}, \quad \cdots.$$

Answer:

12.4 Λ-confluence

Because of the undecidable nature of syntacticness, it is natural to search for sufficient conditions making this notion decidable. The first one has been given in the case of collapse free theories by [Kir85a, Kir86] and is based on the notion of Λ-confluence. We detail here this notion in the general case after the work of F. Klay [Kla92].

12.4.1 Definitions

The first sufficient condition we give is simply a condition on the occurrences of applications of the axioms in the equality proof of two terms. From that we deduce the second sufficient condition which consists in a localization of the previous check using an appropriate notion of critical pairs.

Definition 12.9 A set of equational axioms E in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is said Λ-*confluent* if the following two conditions are satisfied for any terms t_0, t_1, t_2:

1. If

$$t_1 \longleftrightarrow_E^{\Lambda} t_0 \longleftrightarrow_E^{\Lambda} t_2,$$

then there exists terms w and w' such that:

$$t_1 \overset{*}{\longleftrightarrow}{}_E^{\neq\Lambda}\ w \overset{\delta}{\longleftrightarrow}{}_E^{\Lambda}\ w' \overset{*}{\longleftrightarrow}{}_E^{\neq\Lambda}\ t_2.$$

2. If
$$t_1 \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_{E}\; t_0 \longleftrightarrow^{\Lambda}_{E} t_2,$$
then there exists t_3 such that
$$t_1 \overset{\delta}{\longleftrightarrow}{}^{\Lambda}_{E}\; t_3 \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_{E}\; t_2.$$

As a simple application of Lemma 12.2, one can see that if a presentation contains only collapse axioms then it is Λ-confluent.
This definition has an immediate consequence:

Lemma 12.7 If E is a set of axioms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ which is Λ-confluent then E is resolvent.

Proof: By induction on the number of axioms applications at occurrence Λ in one proof of $t =_E t'$. □

But even more interesting is the fact that Λ-confluence induces a particular proof form for any equational theorem. This particular proof has the property that the occurrence of application of an axiom increases in the term as the proof is going on:

Proposition 12.6 Let E be a set of axioms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$. E is Λ-confluent iff for all E-equality $t =_E t'$, there exists a proof:
$$t = t_0 \longleftrightarrow^{p_1}_{A} t_1 \longleftrightarrow^{p_2}_{A} t_2 \dots t_{n-1} \longleftrightarrow^{p_n}_{A} t_n = t' \tag{12.4}$$
such that for all $i, j \in [1..n], i < j \Rightarrow p_i < p_j$ or $p_i \bowtie p_j$.

Proof: If an equational theorem has a proof like 12.4 then obviously the theory is Λ-confluent.
Conversely, let us prove first that if a theory is Λ-confluent, any equational theorem $t = t'$ has a proof of the form:
$$t \overset{\delta}{\longleftrightarrow}{}^{\Lambda}_{E}\; \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_{E}\; t'.$$
For this let us define for any proof
$$t \overset{n_1}{\longleftrightarrow}{}^{\neq\Lambda}_{E}\; r \longleftrightarrow^{\Lambda}_{E} s \overset{*}{\longleftrightarrow}_{E} t' \tag{12.5}$$
of $t = t'$, n_2 the length of the prefix proof without Λ application of axiom, and n_1 the number of application of an axiom at occurrence Λ. Let us made an induction on (n_1, n_2) compared with the lexicographic order on couples of naturals.

- For $(0, 0)$ and $(1, 0)$ the result is clear.
- Otherwise,
  - If $n_2 > 0$, then by definition of Λ-confluence there exists a proof deduced from 12.5 which is of the form:
  $$t \overset{\delta}{\longleftrightarrow}{}^{\Lambda}_{E}\; s' \longleftrightarrow^{\neq\Lambda}_{E} s \overset{*}{\longleftrightarrow}_{E} t'$$
  which complexity is either $(n_1 - 1, n_2')$ or $(n_1, 0)$ for which we can apply the induction hypothesis.
  - If $n_2 = 0$ and $n_1 > 0$, then we can assume that the proof is of the form:
  $$t \longleftrightarrow^{\Lambda}_{E} s \longleftrightarrow^{\Lambda}_{E} s' \overset{*}{\longleftrightarrow}_{E} t'$$
  in which case by definition this can be transformed in a proof:
  $$t \overset{\delta}{\longleftrightarrow}{}^{\Lambda}_{E}\; s' \overset{*}{\longleftrightarrow}_{E} t'$$
  which complexity is $(n_1 - 1, n'2)$.

This proves that any equational theorem $t = t'$ has a proof of the form: $t \overset{\delta}{\longleftrightarrow}{}^{\Lambda}_{E}\; \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_{E}\; t'$. The result is then obtained by a simple structural induction on t'. □

This last result is the fundamental tool in order to prove:

Proposition 12.7 [Kla92]
If E is a Λ-confluent and regular presentation in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ then matching is decidable modulo E.

12.4.2 Localization of Λ-confluence

We localize now the test of Λ-confluence.

Definition 12.10 A critical pair

$$\sigma(g[l]_p) \longleftrightarrow^{p}_{l=r} \sigma(g) \longleftrightarrow^{\Lambda}_{g=d} \sigma(d)$$

for a set E of equational axioms in $\mathcal{T}(\mathcal{F},\mathcal{X})$ is said Λ-*confluent* if there exists a proof

$$\sigma(g[l]_p) \overset{\delta}{\longleftrightarrow}{}^{\Lambda}_{E} u \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_{E} \sigma(d).$$

Proposition 12.8 [Kir85a, DK91]
Let E be a linear presentation in $\mathcal{T}(\mathcal{F},\mathcal{X})$. E is a Λ-confluent set of axioms if and only if any critical pair is Λ-confluent.

Proof: Λ-confluence immediately implies Λ-confluence of critical pairs.
Conversely,

- If $t \longleftrightarrow^{\Lambda}_{E} \longleftrightarrow^{\Lambda}_{E} t'$ then the conclusion is clear by application of the definition of Λ-confluence of critical pairs.
- If $t \overset{n}{\longleftrightarrow}{}^{\neq\Lambda}_{E} \longleftrightarrow^{\Lambda}_{E} t'$ then let us made the proof by induction on n.
  - If $n = 0$ this is clear.
  - If $n > 0$, then the proof is of the form:

$$t \overset{n-1}{\longleftrightarrow}{}^{\neq\Lambda}_{E} u[\mu(l)]_p \longleftrightarrow^{\neq\Lambda}_{l=r} u \longleftrightarrow^{\Lambda}_{g=d} t'.$$

Let p be the position of application of $l = r$ in u.
If $p \notin \mathcal{D}om(g)$ then the two last steps of the proof commute since g is assumed to be linear and the induction hypothesis can be applied on the resulting proof.
If $p \in \mathcal{D}om(g)$ then, since the critical pair is Λ-confluent, the proof can be replaced by the proof:

$$t \overset{n-1}{\longleftrightarrow}{}^{\neq\Lambda}_{E} \overset{\delta}{\longleftrightarrow}{}^{\Lambda}_{E} \longleftrightarrow^{\neq\Lambda}_{E} t',$$

on which the induction hypothesis allows to conclude.

□

Furthermore the Λ-confluence of a linear presentation E in $\mathcal{T}(\mathcal{F},\mathcal{X})$ is decidable. This comes from the fact that if E is a Λ-confluent presentation then following Proposition 12.6 every critical pair (u,v) has a proof of the form:

$$u = t_0 \longleftrightarrow^{p_1}_{A} t_1 \longleftrightarrow^{p_2}_{A} t_2 \dots t_{n-1} \longleftrightarrow^{p_n}_{A} t_n = v$$

such that for all $i,j \in [1..n], i < j \Rightarrow p_i < p_j$ or $p_i \bowtie p_j$. The existence of such a proof is decidable since the terms u and v are finite.

So we get a sufficient decidable condition for a set of axioms to be resolvent:

Corollary 12.3 Any set of axioms E in $\mathcal{T}(\mathcal{F},\mathcal{X})$ such that every critical pair between two elements of E is Λ-confluent is resolvent.

This allows to test automatically if a set of axioms is resolvent. As a consequence, this gives a way to compute automatically the mutation process attached to the theory generated by E. For this kind of theories we are able to build automatically an E-unification procedure which termination should be further proven.

Example 12.6 One should notice that we can not overcome the linearity restriction in the previous results as shown by the following example found by Didier Rémy. Let $E = \{a = f(x,x), b = c\}$, there is no critical pairs and the proof $f(c,b) \longleftrightarrow^{\neq\Lambda}_{E} f(b,b) \longleftrightarrow^{\Lambda}_{E} a$ can not be transformed in a proof of the form $f(c,b) \overset{\delta}{\longleftrightarrow}{}^{\Lambda}_{E} \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_{E} a$.

```
UNIF-COMPLETION = proc ($E$ : set of axioms) returns (set of axioms)
    $B$ : set of axioms := $\emptyset$
    While $E \neq \emptyset$ do
        choose an axiom $(l = r)$ in $E$
        Compute the critical pairs of $l = r$ on itself and with the elements
        of $B$
        Add any critical pair non $\wedge$-confluent under $E \cup B$ into $E$
        Add $l = r$ into $B$
    enddo
    return($B$)
end UNIF-COMPLETION
```

Figure 12.2: Completion into a resolvent set of axioms

NewCP	$Ax \cup \{g = d\}, Pc, B$	$\mapsto\!\!\!\!\rightarrow$	$Ax, Pc \cup PC, B \cup (g = d)$ where PC is the set of $\wedge$-critical pairs obtained by superposing $g = d$ with the elements of $(Ax \cup B)$.
ConfluentCP	$Ax, Pc \cup \{u, v\}, B$	$\mapsto\!\!\!\!\rightarrow$	Ax, Pc, B if (u, v) is $\wedge -$ confluent for $Ax \cup B$
TrivialCP	$Ax, Pc \cup \{u, u\}, B$	$\mapsto\!\!\!\!\rightarrow$	Ax, Pc, B
NonConfluentCP	$Ax, Pc \cup \{u, v\}, B$	$\mapsto\!\!\!\!\rightarrow$	$(Ax \cup \{u = v\}), Pc, B$ if (u, v) is not $\wedge-$confluent for $(Ax \cup B)$

Figure 12.3: **UnificationCompletion**: Rules for computing unification procedures.

12.4.3 A unification completion procedure

As usual [KB70, Hue81, JK84] and as described in Chapter 16, we deduce from the previous critical pair check, a completion procedure which returns, whenever it terminates, a resolvent set of axioms.

Starting from a set of axioms E, the procedure UNIF-COMPLETION described in Figure 12.2 computes the E-critical pairs and checks if they are $\wedge$-confluent. If not, it adds to E the axiom allowing to make the critical pair $\wedge$-confluent.

If UNIF-COMPLETION terminates, it returns a finite and resolvent set of axioms. Otherwise, provided a fairness hypothesis on the choice of the axioms into E, the infinite set of axioms generated is resolvent. UNIF-COMPLETION is an instance of a general completion scheme described by the transformation rules set **UnificationCompletion** in Figure 12.3. The completeness and correctness of these transformations is made in [DK91] using the orderings for equational proofs as described in Chapter 15. A given strategy in the application of the rules of **UnificationCompletion**, together with data representations, fix a completion procedure.

The transformation rules of the syntactic completion procedures are transforming 3-tuples (Ax, Pc, B) where Ax is initially the set of axioms to be completed, Pc the set of critical pairs for which we are testing the $\wedge-$confluence and B the resolvent set of axioms deduced from Ax. We give them here is order to show an interesting application of the completion technique to unification problems. Further details are given in [DK91].

Example 12.7 Let us consider the case (easy but useful) of a set of commutativity axioms:

$$C = \{x +_i y = y +_i x \, | i \in I\}$$

This problem has been first studied by J.Siekmann [Sie79] for I reduced to one element. C is resolvent since clearly all critical pairs are $\wedge$-confluent. Any equation $e = (u +_i v =^? u' +_i v')$ can thus be transformed into the C-equivalent disjunction of systems:

$$\mathrm{Mut}(e) = \left[\begin{array}{l} \left\{\begin{array}{l} u =^? u' \\ v =^? v' \end{array}\right. \\ \left\{\begin{array}{l} u =^? v' \\ v =^? u' \end{array}\right. \end{array}\right.$$

The mutation rule can be computed from this last decomposition scheme and unification algorithms described as we have done in Section 10.5.4.

12.5 Extended presentations

In this section we are presenting another approach to prove syntacticness based on the idea of T. Nipkow [Nip90a] to use an extended signature to check syntacticness.

Definition 12.11 Let E be a presentation in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and $l = f(\vec{r}), f(\vec{g}) = d$ be two axioms in E. We are denoting

$$l = f(\vec{r}) \downarrow_E^{\Lambda} f(\vec{g}) = d \Leftrightarrow (\forall \sigma,\ \sigma(\vec{r}) =_E \sigma(\vec{g}) \Rightarrow \sigma(l) \overset{\delta\Lambda}{\xleftrightarrow{*}_E} \sigma(d)).$$

This allows to give another characterization of syntactic theories:

Lemma 12.8 For a presentation E in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, E is syntactic iff

$$\forall l = f(\vec{r}),\ f(\vec{g}) = d \in E \cup E^{-1}, l = f(\vec{r}) \downarrow_E^{\Lambda} f(\vec{g}) = d.$$

Proof: It is an easy consequence of the last definition and of syntacticness. □

The goal of the next definition is to formalize the use of variables as constants.

Definition 12.12 For any term algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})$, let $\underline{\mathcal{X}}$ be a set of new constants (i.e. function symbols of arity 0 and such that $\mathcal{F} \cap \underline{\mathcal{X}} = \emptyset$) such that there exists a bijection $\zeta : \mathcal{X} \to \underline{\mathcal{X}}$. By extension, this induces an isomorphism $\zeta : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{T}(\mathcal{F} \cup \underline{\mathcal{X}})$ and for any term t, we denote $\zeta(t)$ by $\underline{t}$.

Proposition 12.9 Let E be a presentation in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and $l = f(\vec{r}), f(\vec{g}) = d$ be two axioms in E. Let $\mathcal{E} = E \cup E'$ where $E' = \{r_1 = g_1, \ldots, r_n = g_n\}$. If there exists a proof:

$$l \overset{\delta_0\Lambda}{\xleftrightarrow{*}_E} t_1 \xleftrightarrow{p_1}_{E'} u_1 \ldots u_{k-1} \overset{\delta_{k-1}\Lambda}{\xleftrightarrow{*}_E} t_k \xleftrightarrow{p_k}_{E'} u_k \overset{\delta_k\Lambda}{\xleftrightarrow{*}_E} d \tag{12.6}$$

where $\Sigma_{i=0}^{i=k} \delta_k \leq 1$ and $\forall i \in [1..k], p_i \neq \Lambda$, then

$$l = f(\vec{r}) \downarrow_E^{\Lambda} f(\vec{g}) = d.$$

Proof: Let σ be a substitution such that $\sigma(\vec{r}) =_E \sigma(\vec{g})$, whe shall prove that $\sigma(l) \overset{\delta\Lambda}{\xleftrightarrow{*}_E} \sigma(d)$.
With respect to the proof 12.6 the proof goes by induction on k and shows that there exists a proof of $\sigma(l) =_E \sigma(d)$ having exactly $\Sigma_{i=0}^{i=k} \delta_k$ Λ-applications of axioms.

- If $k = 0$, then $l \overset{\delta_0\Lambda}{\xleftrightarrow{*}_E} d \Rightarrow \sigma(l) \overset{\delta_0\Lambda}{\xleftrightarrow{*}_E} \sigma(d)$.
- If $k > 0$, then by induction hypothesis:

$$\sigma(l) \overset{\delta\Lambda}{\xleftrightarrow{*}_E} \sigma(t_k) \text{ and } \sigma(u_k) \overset{\delta_k\Lambda}{\xleftrightarrow{*}_E} \sigma(d)$$

with $\delta' = \Sigma_{i=0}^{i=k-1} \delta_k$ and thus $\delta' + \delta_k \leq 1$. Moreover since $t_k \xleftrightarrow{p_k}_{E'} u_k$ there exists i such that $t_{k|p_k} = \mu(r_i)$ and $u_{k|p_k} = \mu(g_i)$. By hypothesis we have in particular for i; $\sigma(r_i) =_E \sigma(g_i)$. This allows to write:

$$\sigma(t_{k|p_k}) = \sigma(\mu(r_i)) =_\sigma (\mu(g_i)) = \sigma(u_{k|p_k})$$

which proves that:

$$\sigma(l) \overset{\delta\Lambda}{\xleftrightarrow{*}_E} \sigma(t_k) \overset{\neq\Lambda}{\xleftrightarrow{*}_E} \sigma(u_k) \overset{\delta_k\Lambda}{\xleftrightarrow{*}_E} \sigma(d).$$

□

12.6 Applications

12.6.1 Transitivity

12.6.2 Shallow theories

12.6.3 AC matching

12.6.4 Acyclic theories

12.6.5 Touffues theories

Chapter 13

Restricted semantic unification

13.1 Associative-Commutative unification

13.1.1 Introduction

Associative-commutative unification (in short AC-unification) is solving equations when a finite set of functions symbols $(+_i)_{i\in I}$ are associative and commutative, that is satisfy $AC(+_i)$:

$$\begin{array}{c}(x +_i y) +_i z = x +_i (y +_i z)\\ x +_i y = y +_i x\end{array}$$

This unification problem has been the more intensively studied after the empty theory for at least two reasons. The first one is that the associative and commutative axioms are associated in many algebraic structures of interest for theorem proving or algebraic specifications. The second and more technical one is that, in term rewriting applications, one cannot dissociate associativity from commutativity because the rewriting relation associated to (left or right) associativity modulo commutativity is not terminating. Indeed, the rewriting relation $\longrightarrow^{R/C}$ is not terminating as the reader can see considering the rewrite rule $(x + y) + z \to x + (y + z)$ and the equational axiom $C(+) = \{x + y = y + x\}$. We have then the following derivation:

$$(x + y) + z \to x + (y + z) =_C (z + y) + x \to z + (y + x) =_C (x + y) + z.$$

Thus one should work modulo associativity and commutativity both as equational axioms.

But solving associative-commutative equations is a difficult problem unlike what happens for commutativity only. Difficulties are localized firstly in the discovery of complete AC-unification procedures, secondly in the proof of their termination, thirdly in managing the algorithmic complexity of the problem. The first AC-unification algorithms have been discovered independently by Livesey and J.Siekmann [LS76] and M.Stickel [Sti76, Sti81]. They are mainly differing by the way generalization is handled: Stickel generalizes using variables and thus transforms an AC-equation into an homogeneous linear diophantine equation, while Livesey and Siekmann consider certain variables as constants, so that an AC-equation is transformed into an inhomogeneous linear diophantine equation. Termination of Stickel's algorithm in presence of free symbols has been proved almost ten years after his discovery by F.Fages [Fag84] whose complexity measure has been extended to prove the termination of Robinson-like combination of unification algorithms, as we have seen previously. Fages's complexity measure has also been used to prove the termination of an improvement of the Livesey-Siekmann AC-unification algorithm in [HS85]. AC-unification has been proved to be NP-complete by D.Kapur and Narandran [KN86] and big efforts have been spent in order to handle efficiently reasonable AC-problems [Hul80c, CL87].

In this context, we will show in this section that the rule based framework that has been developed before, can be used to study associative-commutative unification. As a consequence of the previous study on the combination of unification algorithms, we are only studying the problem of AC-unification in presence of one AC-symbol.

13.1.2 Preparation and simplification of the problem

In what follows, $*$ in an associative-commutative symbol and $\mathcal{F} = \{*\}$. In other words, we consider only terms in $\mathcal{T} = \mathcal{T}(\{*\}, \mathcal{X})$. A system of multiequations built on $\mathcal{T}$ is called an AC$_{\mathcal{T}}$*-system* or shortly an *AC*-system.

Definition 13.1 The flattened form of the term $t \in \mathcal{T}$ is the pair $\{*, \{x_1, \ldots, x_n\}\}$ composed of the symbol $*$ and the multiset of variables occurring in t. It is also denoted $*(x_1, \ldots, x_n)$ or $x_1 * \ldots * x_n$.

Example 13.1 The flattened form of $(x * y) * (x * z)$ is $*(x, x, y, z)$.

Since $*$ is AC, it is left and right-regular, and thus this property allows to simplify the equations:

Lemma 13.1 For all variables $x, x_1, \ldots, x_n, x'_1, \ldots, x'_p$ we have

$$x * x_1 * \ldots * x_n =^? x * x'_1 * \ldots * x'_p \Leftrightarrow_{AC} x_1 * \ldots * x_n =^? x'_1 * \ldots * x'_p$$

Note that this result is valid for any repeated term, a remark that an implemantation should use eagerly.

Example 13.2 $f(a, b) + x + y + x =^? x + z + f(a, b)$ is simplified into $x + y =^? z$.

The first step for AC-full-decomposition is thus flattening and simplification.

13.1.3 Solving strategies for AC problems

Parallel versus sequential solving

Let S be the system:

$$S = \begin{cases} x * y & =^? \; u * v \\ x * y' & =^? \; u * v' \end{cases}$$

If the two equations are concurrently (and thus independently) solved, one gets a disjunction of systems containing in particular the system:

$$\begin{cases} x & =^? \; x_1 * x_2 \; =^? \; x'_1 * x'_2 \\ u & =^? \; x_1 * x_3 \; =^? \; x'_1 * x'_3 \\ y & =^? \; x_3 * x_4 \\ v & =^? \; x_2 * x_4 \\ y' & =^? \; x'_3 * x'_4 \\ v' & =^? \; x'_2 * x'_4 \end{cases}$$

But the two first multiequations are the same as the equations of S, up to a renaming of the variables.

Of course one can sequentialy solve each equation and instanciate with the solutions the other equations in the system. But as we will see later, this is extremely expensive and moreover it leads to quite redondant complete set of AC-unifiers.

The key idea, in order to get rid of this phenomena, is to solve directly the whole system instead of one equation after another. In fact this approach is very natural in our framework.

Necessity of replacement

Provided an algorithm for solving systems of diophantine equations is chosen, one may think that the AC-mutation process will terminate. But this is not true since fully decomposed equation (i.e. of the form $x =^? t$) may interfer with solving the remaining of the AC$_{\mathcal{T}}$-system, as in the following example.

Example 13.3 Let:

$$S = \begin{cases} x * y & =^? \; z * t \\ x & =^? \; z * u \end{cases}$$

By replacing the first equation by one of the systems obtained by solving it, one gets after merging:

$$S' = \begin{cases} x & =^? \; z * u \; =^? \; x_1 * x_2 \\ z & =^? \; x_1 * x_3 \\ y & =^? \; x_3 * x_4 \\ t & =^? \; x_2 * x_4 \end{cases}$$

and one recognizes in the two first equations of S', a renaming of the equations of S.

So it is not sufficient to solve systems of non-fully decomposed equations independently of the fully decomposed ones in the system. One needs either to solve the system as a whole, thus including all the equations of the system, or to replace the already known constraints on variables into the non-fully decomposed equations. In the case of the previous example, we get then:

Example 13.4 (Continued)

$$S'' = \begin{cases} z * u * y =^? z * t \\ x =^? z * u \end{cases}$$

which can be simplified and trivially solved.

These two previous remarks about the termination of the AC-solving process show in particular that one can not parallelize naively the AC-unification algorithm and that either one must solve maximal systems of equations as a whole, or that each equation has to be solved one by one with an eager application of replacement.

13.1.4 Associative-commutative unification

We are now showing the relationship between AC-systems and systems of diophantine homogenous linear solution, also called in the following *dio-systems*.

Reduction to diophantine systems

Let S be the following simplified and flattened $\mathrm{AC}_{\mathcal{T}}$-system:

$$S = (u_1 * \ldots * u_m =^? v_1 * \ldots * v_p)_{k \in [1..q]}$$

where the u_i, v_j are (possibly similar) variables from $\mathcal{X}$.

Since $*$ is associative-commutative, S can also be written:

$$S = (a^k_1 u_1 * \ldots * a^k_n u_m =^? b^k_1 v_1 * \ldots * b^k_p v_p)_{k \in [1..q]}$$

where $a^j_i u$ stands for $\underbrace{u * \ldots * u}_{a^j_i}$ and for all $k \in [1..q]$, the coefficients $(a^k_i)_{i \in [1..m]}$ and $(b^k_j)_{j \in [1..p]}$ are naturals.
By definition, for any term t, $0t$ is the empty term Λ. We suppose furthermore that this empty term is by convention an identity for $*$: $\forall t \in \mathcal{T}, t * \Lambda = t$.

Example 13.5 The following is a system of two equations with four variables:

$$S^0 = \begin{cases} 2x * 3y =^? 2z * 2u \\ u * 5y =^? 3z * x \end{cases}$$

If $X = (x_1, \ldots, x_n)^t$ are the distinct variables of the system S, it is more convenient to write S under the form:

$$AX = \Lambda,$$

where A is an integer $q \times n$ matrix.

Example 13.6 (13.5 continued) The previous system S^0 is written:

$$\begin{pmatrix} 2 & 3 & -2 & -2 \\ -1 & 5 & -3 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ u \end{pmatrix}$$

The solutions of the system S are necessarily of the form:

$$x_i = \prod_{r \in R} \delta^i_r z_r, \text{ with } \delta^i_r \in \mathbf{N}, \sum_{r \in R} \delta^i_r \neq 0 \text{ and } \forall r \in R, z_r \in \mathcal{X} - \mathcal{V}ar(S) \tag{13.1}$$

for some finite set of indexes R. Denoting $Z = (z_1, \ldots, z_{|R|})^t$ and Δ the matrix $(\delta^i_r)^{1 \le i \le n}_{r \in R}$, the solutions satisfy the relation:

$$X = \Delta Z.$$

An AC-solution has to satisfy:

$$\begin{array}{l}\forall k \in [1..q] \quad \prod_i a_i^k (\prod_{r\in R} \delta_r^i z_r) = \Lambda \\ \Leftrightarrow \\ \forall k \in [1..q] \quad \prod_r (\sum_i a_i^k \delta_r^i) z_r = \Lambda\end{array}$$

The coefficients of each z_r have thus to be identically nul:

$$\forall k \in [1..q], \forall r \in R \quad \sum_i a_i^k \delta_r^i = 0$$

which states that the δ_r^i are solutions of the diophantine linear homogeneous system:

$$A\mathbf{X} = 0 \tag{13.2}$$

which is denoted S_d and where $\mathbf{X} = (\mathbf{x}_i)_{i\in[1..n]}$ are the unknows *ranging over* $\mathbf{N}$.

Since Clifford and Preston [CP61] have shown that the set of solutions of such a system is a finitely generated monoid, we are in fact interested in the set of minimal solutions of such a system for the order induced on the cartesien product by the natural order on naturals:

$$\mathbf{X} \leq \mathbf{Y} \Leftrightarrow \forall i, \;\; \mathbf{x}_i \leq \mathbf{y}_i.$$

Back to terms

Supposing the system of diophantine equations S_d solved (this point is reached in the next section) one have then to compute from its solutions, the initial system on terms S complete set of solutions.

Let $U = \{u_l | l \in L\}$, the finite set of minimal solutions of the diophantine system S_d. Each solution is then of the form:

$$\sum_{l\in L} \lambda_l u_l \text{ with } \lambda_l \in \mathbf{N}$$

so that, by definition of the δ_r:

$$\forall r, \; \forall i, \; \exists \lambda_l^r, \; \delta_r^i = \sum_{l\in L} \lambda_l^r u_l^i$$

where the λ_l^r can be zero. Combining 13.1 and the last equality we get:

$$\forall i, \;\; x_i = \prod_{r\in R} \sum_{l\in L} \lambda_l^r u_l^i z_r = \sum_{l\in L} u_l^i (\prod_{r\in R} \lambda_l^r z_r).$$

For all $l \in L$, let ζ_l be a new variable, it is more general than the term:

$$\prod_{r\in R} \lambda_l^r z_r$$

and thus the substitution:

$$\forall i, \;\; x_i \mapsto \prod_{l\in L} \zeta_l u_l^i$$

would be a solution of the initial problem S, more general than any other AC-solution. But one should take care of the fact that the λ_l^r may be zero, which means that ζ_l is then the empty term. And there is $2^{|L|}$ possible choice for the ζ_l to be, or not be, the null term. For all these possibilities one have only to keep those such that:

$$\forall i, \;\; x_i \neq \Lambda,$$

i.e. the variable x_i should be instanciated with a not the empty term. This is the major difference with the case of associative, commutative and identity (AC1) unification.

Finaly we get the following result:

Theorem 13.1 *The set of substitutions:*

$$\Sigma = \{\sigma | \sigma(x_i) = \prod_l u_l^i \zeta_l \text{ with } \zeta_l = \Lambda \text{ or } z_l \text{ and } \forall i \in [1..n], \prod_l u_l^i \zeta_l \neq \Lambda\},$$

is a complete set of AC-unifiers of S.

Proof: By construction of Σ, all its elements are AC-solutions of S. We have thus to show the completeness of Σ. Let α be any AC-solution of S. Its components should satisfy 13.1 and by definition of the λ_l^r, there exists $\sigma \in \Sigma$ such that $\sigma \leq^{\mathcal{V}ar(S)} \alpha$. □

An example

Let us solve the equation $x * y =^? u * v$ where $*$ is an AC symbol and x, y, u, v are variables.

The diophantine equation to be solved is $X + Y =^? U + V$ which have as minimal complete set of solutions:

$$\{(1,0,1,0), (1,0,0,1), (0,1,1,0), (0,1,0,1)\}.$$

Let us represent the situation in the following matrix:

x	y	u	v	
1	0	1	0	ζ_1
1	0	0	1	ζ_2
0	1	1	0	ζ_3
0	1	0	1	ζ_4

where a (term) variable ζ_i is associated to each solution of the diophantine equation.

Now for clarity we represent all the possibilities for the ζ_i to be the empty term or not. The empty rows are corresponding to the case where one of the variable $x(= \zeta_1 * \zeta_2), y(= \zeta_3 * \zeta_4), u(= \zeta_1 * \zeta_3), v(= \zeta_2 * \zeta_4)$ has for value the empty term.

ζ_1	0								1							
ζ_2	0				1				0				1			
ζ_3	0		1		0		1		0		1		0		1	
ζ_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x							ζ_2	ζ_2		ζ_1		ζ_1		$\zeta_1 * \zeta_2$	$\zeta_1 * \zeta_2$	$\zeta_1 * \zeta_2$
y							ζ_3	$\zeta_3 * \zeta_4$		ζ_4		$\zeta_3 * \zeta_4$		ζ_4	ζ_3	$\zeta_3 * \zeta_4$
u							ζ_3	ζ_3		ζ_1		$\zeta_1 * \zeta_3$		ζ_1	$\zeta_1 * \zeta_3$	$\zeta_1 * \zeta_3$
v							ζ_2	$\zeta_2 * \zeta_4$		ζ_4		ζ_4		$\zeta_2 * \zeta_4$	ζ_2	$\zeta_2 * \zeta_4$

Which prove that the complet set of AC-unifier (minimal in this case but, unfortunately this is not true in general) of the equation $x * y =^? u * v$ consists of the following seven substitutions:

$$\left\{\begin{array}{l} x\mapsto\zeta_2 \\ y\mapsto\zeta_3 \\ u\mapsto\zeta_3 \\ v\mapsto\zeta_2 \end{array}\right. \quad \left\{\begin{array}{l} x\mapsto\zeta_2 \\ y\mapsto\zeta_3 * \zeta_4 \\ u\mapsto\zeta_3 \\ v\mapsto\zeta_2 * \zeta_4 \end{array}\right. \quad \left\{\begin{array}{l} x\mapsto\zeta_1 * \zeta_2 \\ y\mapsto\zeta_3 * \zeta_4 \\ u\mapsto\zeta_1 * \zeta_3 \\ v\mapsto\zeta_2 * \zeta_4 \end{array}\right. \quad \left\{\begin{array}{l} x\mapsto\zeta_1 \\ y\mapsto\zeta_4 \\ u\mapsto\zeta_1 \\ v\mapsto\zeta_4 \end{array}\right. \quad \left\{\begin{array}{l} x\mapsto\zeta_1 \\ y\mapsto\zeta_3 * \zeta_4 \\ u\mapsto\zeta_1 * \zeta_3 \\ v\mapsto\zeta_4 \end{array}\right.$$

$$\left\{\begin{array}{l} x\mapsto\zeta_1 * \zeta_2 \\ y\mapsto\zeta_4 \\ u\mapsto\zeta_1 \\ v\mapsto\zeta_2 * \zeta_4 \end{array}\right. \quad \left\{\begin{array}{l} x\mapsto\zeta_1 * \zeta_2 \\ y\mapsto\zeta_3 \\ u\mapsto\zeta_1 * \zeta_3 \\ v\mapsto\zeta_2 \end{array}\right.$$

One can see on this simple example that solving linear diophantine system of equations is only one part of the problem. The second important one is to combine properly (i.e. without introducing empty term) these solutions. Since there is no way to avoid checking that a column is not full of zero by interaction of the value of λ_l and the component of a dio-solution, this combination process is exponential in the number of minimal solution of the diophantine equation associated to the problem.

13.1.5 Solving systems of Diophantine equations

Localization of the minimal solutions

When the system is reduced to one equation ($k = 1$), then the minimal solutions $X_1, \ldots, X_m$, $Y_1, \ldots, Y_p$ of the equation:

$$\sum_{1\le i\le m} a_i X_i =^? \sum_{1\le j\le p} b_j Y_j,$$

are bounded as follow:

$$\sum_{i=1,\ldots,m} X_i \le \max_{j=1,\ldots,p} b_j \text{ and } \sum_{j=1,\ldots,p} Y_i \le \max_{i=1,\ldots,m} a_i$$

as proved by J.-L. Lambert [Lam87b], improving the bound:

$$\max_{i=1,\ldots,m} X_i \le \max_{j=1,\ldots,p} b_j \text{ and } \max_{j=1,\ldots,p} Y_i \le \max_{i=1,\ldots,m} a_i,$$

given by G. Huet [Hue78].

For a system of diophantine equations $S = (A\mathbf{X} =^? 0)$, let $l = \max_{1\leq i\leq n, 1\leq j\leq k} |a_{ij}|$. J.-F. Romeuf has proved [Rom88] that the minimal solutions $(Z_i)_{1\leq i\leq n}$ of the system S satisfy:

$$\max_{1\leq i\leq n} Z_i \leq (2l)^{2^n-1}$$

This is satisfactory from a theoretical point of view, but this bound is quite large:

Example 13.7 In the system S^0 above, $l = 5$ and $n = 2$ so that the bound is $(2\times 5)^{2^2-1} = 1000$, which is quite large for such a small system.

We do not address here the problem of solving linear homogeneous diophantine equations with constrains as in [Hul80c] or linear inhomogeneous equations as in [Büt85]. A way to nicely implement AC-unification (as well as AC-matching) is described in [Eke93].

Solving systems of homogeneous and linear diophantine equations

The problem is now to solve *systems* of multiequations built on terms of $\mathcal{T}$ without any equation of the form $x =^? t$ where x is a variable. This problem can be reduced to two steps:

1. the solving of linear homogeneous diophantine equations, followed by

2. the correct combination of the diophantine equations solutions in order to get a complete set of AC-solutions.

This reduction has been detailed in [Hul80c, Büt85] for example. The second subproblem is specifically addressed in [CL87].

The major difficulty that occurs in solving linear diophantine equations is to bound the search space. There are two possibilities: the first one is to extend Huet's algorithm for the solving of diophantine equations to handle systems of multiequations and this can be done in several ways:

1. Find a generalization to non-trivial systems of Huet's [Hue78] or Lambert's [Lam87b] bounds of the minimal solutions, but this seems non-trivial.

2. Solve each diophantine equation sequentially using Huet's or Lambert's algorithm and replace the result in the other ones. But it is close to the previous (Stickel-like) approach since, up to data representation and combination of diophantine solutions, it is similar to solve diophantine equations and AC-equations in $\mathcal{T}$.

3. Solve the system like in real or rational vector space using the Gauss elimination method. This approach has the drawback to be quite sensible to the way the Gauss eliminations are performed. We are not developping the point here, but only give an example:
   Assume that we want to solve the system:

$$\left\{\begin{array}{lll} 2x+3y & =^? & 2z \\ x+5y & =^? & 3z \end{array}\right.$$

   It is equivalent to:

$$\left\{\begin{array}{lll} 6z-10y+3y & =^? & 2z \\ x & =^? & 3z-5y \end{array}\right. \Leftrightarrow \left\{\begin{array}{lll} y & =^? & 4k \\ z & =^? & 7k \\ x & =^? & 3z-5y \end{array}\right.$$

   for all k in $\mathbf{N}$.

The second possibility is to give algorithms for solving directly systems of linear Diophantine equations. One has been given by [CD91] and generalizes to systems the algorithm of A. Fortenbacher [For83, Lan87] improved in [CF90]. Another one has been given by [Dom91b] using geometrical arguments, and [Pot91] has given an algorithm based on Grobner basis.

13.1.6 Conclusion

This approach to associative-commutative unification has the great advantage to (1) benefit of the accumulation of constraints consisting of multiple non-fully decomposed equations, so that the search space involved by the solving of the diophantine equations is dramatically cut down and (2) to factorize multiple combination steps needed to built the AC-solutions from the diophantine solutions. Let us show the point on the following example where the algorithm we have presented and Stickel's one are used comparatively:

Example 13.8 Suppose that $*$ is associative commutative, that f is a free symbol and let us solve the equation $e = (f(x * y, x * b) =^? f(u * v, u * a))$

- by decomposition and generalization we get:

  $$S = \{x * y =^? u * v, x * x_1 =^? u * x_2, x_1 =^? b, x_2 =^? a\}$$

  and after solving the associated system of two diophantine equations we get directly, (roughly speaking in one step) as complete set of AC-solution:

  $$\begin{aligned}
  \mu_1 &= \{x \mapsto a, u \mapsto b, v \mapsto a, y \mapsto b\}\\
  \mu_2 &= \{x \mapsto a, u \mapsto b, v \mapsto (z_1 * a), y \mapsto (z_1 * b)\}\\
  \mu_3 &= \{u \mapsto (b * z_3), x \mapsto (a * z_3), v \mapsto a, y \mapsto b\}\\
  \mu_4 &= \{u \mapsto (b * a), x \mapsto (a * a), v \mapsto (z_1 * a), y \mapsto (z_1 * b)\}
  \end{aligned}$$

- With Stickel's algorithm, the equation $x * y =^? u * v$ is first solved into

  $$\begin{aligned}
  \sigma_1 &= \{v \mapsto x, u \mapsto y\}\\
  \sigma_2 &= \{u \mapsto (y * z_3), x \mapsto (v * z_3)\}\\
  \sigma_3 &= \{u \mapsto x, v \mapsto y\}\\
  \sigma_4 &= \{v \mapsto (y * z_2), x \mapsto (z_2 * u)\}\\
  \sigma_5 &= \{u \mapsto (z_2 * x), y \mapsto (v * z_2)\}\\
  \sigma_6 &= \{v \mapsto (z_1 * x), y \mapsto (z_1 * u)\}\\
  \sigma_7 &= \{u \mapsto (z_2 * z_4), v \mapsto (z_1 * z_3), x \mapsto (z_3 * z_4), y \mapsto (z_1 * z_2)\}
  \end{aligned}$$

  The terms of the equation $x * b =^? u * a$ have then to be successively instantiated by each of these substitutions and the equations obtained, that are generally more complex than the original one, have to be solved. For example the equation $\sigma_7(x * x_1) =^? \sigma_7(u * x_2)$, that has a complete set of AC-solutions of seven elements, has to be solved.

In this particular case, the first approach requires to solve one $\text{AC}_{\mathcal{T}}$-system of two AC-equations and thus to apply one solving step for a system of two linear diophantine equations and *one* combination step, while Stickel's algorithm requires to solve sequentially eight AC-equations and thus eight linear diophantine equations and eight combination problems.

13.1.7 Improvements

The basic algorithm that we have described can be improved mainly with a more careful generalization than the one introduced. As we said before, the free symbols may be considered as part of a given theory –for example associativity-commutativity–. In this case the diophantine equations to be solved are inhomogeneous and it is of great interest, since in this case, the search space largely cut down and the strategy guided and thus more efficient. Let us take the geometric interpretation of a diophantine equation on three variables. If it is homogeneous, it is of the form $ax + by + cz = 0$ which is the equation of an hyperplan containing the origin point $(0, 0, 0)$: in that case, one must bound the search space to find a basis of the set of solution, that is a basis of the set of points in the hyperplan that have natural coordinates: no side of the hyperplan is prohibited. But suppose now that the equation is inhomogeneous i.e. of the form $ax + by + cz = d$. In that case on can try all the possible paths from the origin to a point of the hyperplan whose coordinate are naturals, in such a way that all the points of the path are in the same half plan as the origin. Furthermore, in the case where generalization is restricted as much as possible, the search space is also cut down, since the variables issued from generalization have not to be considered and merged after with the initial constraint [HS85, Her87].

13.2 Boolean unification

13.2.1 Introduction

We consider in this section the problem of solving equations in specific algebras with finite domains, rather than on the term algebra as it is the case for standard unification. But the algebras considered here have in common with the term algebra an important property: unification is unitary, which means that any equation has a most general unifier, up to substitution equivalence. This unifier is a substitution in a quotient term algebra and schematizes all the solutions of the given equation. One of the main difficulty to build this unifier will be to minimize the set of variables used in the image of the substitution.

Unification in these algebras open rich domains to resolution-based theorem prover or programming languages: design of digital cicuits, combinatorics, multi-valued logic and propositional logic, mathematics over finite fields,....

In the first part of this chapter, we focus our attention on boolean rings. Methods for finding unifiers for boolean equations are known from a long time, going back to Boole himself in 1847 [Boo47] or to Löwenhein in 1908 [Löw08]. An algorithm for computing the most general unifier of a boolean equation, based on Boole's method, is described in [BS87]. A survey on Boolean unification can be found in [MN89]. Implementations of efficient Boolean unification algorithms are described in [RT90, Rau90]. After describing the algebraic structure of boolean rings, we consider unification in boolean rings and more specifically in the two-elements boolean ring, usually called boolean unification.

In the second part of the chapter, we consider boolean algebras and their generalization called primal algebras. Examples are finite fields, in particular modular arithmetic, Post algebras, matrix rings over finite fields. The unification problem in the class of primal algebras and in their varieties is extensively studied in [Nip90b]. Unification in finite algebras is detailed in [BES$^+$90, Rin90].

13.2.2 Boolean rings

The class of boolean rings is defined by the next set of axioms BR built on the set of function symbols $\mathcal{F}_{\mathcal{BR}} = \{0, 1, \cdot, \oplus\}$.

$$BR = \left\{ \begin{array}{rcl} x \oplus 0 & = & x \\ x \oplus x & = & 0 \\ x \cdot 0 & = & 0 \\ x \cdot 1 & = & x \\ x \cdot x & = & x \\ x \cdot (y \oplus z) & = & (x \cdot y) \oplus (x \cdot z) \\ x \cdot y & = & y \cdot x \\ (x \cdot y) \cdot z & = & x \cdot (y \cdot z) \\ x \oplus y & = & y \oplus x \\ (x \oplus y) \oplus z & = & x \oplus (y \oplus z) \end{array} \right.$$

Definition 13.2 A *boolean ring* $\mathcal{B} = (B, \mathcal{F}_{\mathcal{BR}})$ is defined by a non-empty carrier B and a set of operators $\mathcal{F}_{\mathcal{BR}}$ and is a model of BR.

The set of axioms BR has the property to be equivalent to a class rewrite system given by the following rewrite rules, as proved by Hsiang and Dershowitz [HD83].

$$\begin{array}{rcl} x \oplus 0 & \to & x \\ x \oplus x & \to & 0 \\ x \cdot 0 & \to & 0 \\ x \cdot 1 & \to & x \\ x \cdot x & \to & x \\ x \cdot (y \oplus z) & \to & (x \cdot y) \oplus (x \cdot z) \end{array}$$

and the associativity and commutativity axioms of $\cdot$ and $\oplus$:

$$x \cdot y \quad = \quad y \cdot x$$

$$\begin{aligned}(x \cdot y) \cdot z &= x \cdot (y \cdot z)\\ x \oplus y &= y \oplus x\\ (x \oplus y) \oplus z &= x \oplus (y \oplus z)\end{aligned}$$

This class rewrite system is convergent modulo AC and can be used to decide equality in boolean rings.

13.2.3 Unification in boolean rings

The unification problem in boolean rings, and more precisely in the initial boolean ring $\mathcal{T}(\mathcal{F_{BR}})/BR$, is often called boolean unification and has attracted considerable interest for its applications: it is of practical relevance for manipulating hardware descriptions and solving formulas of propositional calculus; its implementation in constraint logic programming languages allowed the handling of Boolean constraints (CHIP, Prolog III) or sets constraints (CAL).

Boolean terms are built on the set of function symbols $\mathcal{F_{BR}} = \{0, 1, \cdot, \oplus\}$ and a set of variables $\mathcal{X}$. Let us denote by $t(\underline{x})$ any boolean term whose variables $x_1, \ldots, x_n$ are linearly ordered and $\underline{x} = (x_1, \ldots, x_n)$. So $t(\underline{x})$ is a function of the variables $x_1, \ldots, x_n$. Given a boolean ring $\mathcal{B} = (B, \mathcal{F_{BR}})$ of carrier B, each boolean term $t(x_1, \ldots, x_n)$ is interpreted as a function $t_{\mathcal{B}} : B^n \to B$, where B^n denotes the n times cartesian product of the domain B. From now on, we focus on the case where $B = \{0, 1\}$.

Definition 13.3 A boolean equation is of the form $t_1 =^? t_2$ where t_1 and t_2 are boolean terms. Let $\mathcal{V} = \mathcal{V}ar(t_1) \cup \mathcal{V}ar(t_2)$ be the set of variables in the equation. A *solution* of $t_1 =^? t_2$ is an assignment α of $\mathcal{V}$ into $\mathcal{T}(\mathcal{F}_{BR})$ (extended to an homomorphism from $\mathcal{T}(\mathcal{F}_{BR}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}_{BR})/BR$) such that $\mathcal{B} \models \alpha(t_1) = \alpha(t_2)$. A *unifier* of $t_1 =^? t_2$ is a substitution σ of $\mathcal{T}(\mathcal{F}_{BR}, \mathcal{X})$ such that $\sigma(t_1) =_{BR} \sigma(t_2)$.

Example 13.9 Consider the equation $x \cdot y \oplus z =^? 0$. $\mathcal{V} = \{x, y, z\}$. A solution is for instance $\alpha = (x \mapsto 0)(y \mapsto 1)(z \mapsto 0)$ and a unifier would be $\sigma = (z \mapsto x \cdot y)$.

However the techniques developed below also permit to solve equations in boolean rings finitely generated by a set of constants $\mathcal{C}$. In this case, equations $t_1 =^? t_2$ involves terms t_1 and t_2 of $\mathcal{T}(\mathcal{F_{BR}}, \mathcal{C} \cup \mathcal{X})$. A solution of $t_1 =^? t_2$ is an assignment α of $\mathcal{V}$ into $\mathcal{T}(\mathcal{F_{BR}}, \mathcal{C})$ (or $\mathcal{T}(\mathcal{F_{BR}} \cup \mathcal{C})$) such that $\alpha(t_1) = \alpha(t_2)$ holds in the boolean ring generated by $\mathcal{C}$. A unifier of $t_1 =^? t_2$ is a substitution σ of $\mathcal{T}(\mathcal{F_{BR}}, \mathcal{C} \cup \mathcal{X})$ such that $\sigma(t_1) =_{BR} \sigma(t_2)$.

The first step of equation solving in a boolean ring is first to transform the equation $t_1 =^? t_2$ to a matching problem $t =^? 0$, thanks to the properties of these structures:

Lemma 13.2 The two equations $t_1 =^? t_2$ and $t_1 \oplus t_2 =^? 0$ are equivalent.

Proof: Solutions of $t_1 =^? t_2$ are solutions of $t_1 \oplus t_2 =^? t_2 \oplus t_2$, which are solutions of $t_1 \oplus t_2 =^? 0$, thanks to the axiom $x \oplus x = 0$. Solutions of $t_1 \oplus t_2 =^? 0$ are then solutions of $t_1 \oplus t_2 \oplus t_1 =^? t_1$, which are solutions of $t_1 =^? t_2$, again thanks to the axiom $x \oplus x = 0$. □

So we are left to solve equations with a right-hand side equal to 0, so actually to solve a matching problem.

Unification in boolean ring is unitary, as proved below. But since equivalence classes modulo BR are infinite, there exist in general an infinite number of most general unifiers which are equivalent. The following criterion may be applied to select one of them.

Definition 13.4 A unifier σ of $t =^? 0$ is *reproductive* if for any $\underline{b}$ *solution* of $t =^? 0$, i.e. $t(\underline{b}) = 0$, σ satisfies

$$(\sigma(x_1)(\underline{b}), \ldots, \sigma(x_n)(\underline{b})) = \underline{b}.$$

This property is enough to prove that a unifier is a most general unifier.

Proposition 13.1 A reproductive unifier is a most general unifier.

Proof: Assuming that ϕ is a unifier and σ a reproductive unifier, we prove that $\sigma \leq_{BR}^{\mathcal{V}(t)} \phi$.

Let $\mathcal{V}Ran(\phi) = \{y_1, \ldots, y_m\}$ be the variables occurring in the image of ϕ. For any m-tuple $\underline{b'}$, $\phi(\underline{b'})$ is a solution of $(t =^? 0)$. Since σ is reproductive, we get

$$\forall \underline{b'},\ \forall i = 1, \ldots, n,\ \sigma(x_i)(\phi(\underline{b'}) = (\phi(\underline{b'}))_i$$

So $\forall x_i \in \mathcal{V}(t),\ \phi(\sigma(x_i)) =_{BR} \phi(x_i)$ and as a consequence $\sigma \leq_{BR}^{\mathcal{V}(t)} \phi$. □

In this framework there are two methods for computing a most general unifier for a boolean equation $t(\underline{x}) =^? 0$: the successive variable elimination method originally defined by Boole [Boo47], and the use of a particular solution as proposed by Löwenheim [Löw08]. Both compute reproductive unifiers.

Boole's method

The successive variable elimination method, due to Boole, relies on the next ideas:

1. The equation $t(x_1, \ldots, x_n) =^? 0$ is equivalent to another equation where the first variable is isolated, i.e. of the form $a \cdot x_1 \oplus b$, where $a = t(0, x_2, \ldots, x_n) \oplus t(1, x_2, \ldots, x_n)$ and $b = t(0, x_2, \ldots, x_n)$.

2. The equation $t(x_1, \ldots, x_n) =^? 0$ has a solution iff $t(0, x_2, \ldots, x_n) \cdot t(1, x_2, \ldots, x_n) =^? 0$ has a solution.

These two facts lead to the explicit construction of the most general unifier when the equation is satisfiable.

Theorem 13.2 *[Boo47] Boolean unification is unitary: Any satisfiable equation $t(x_1, \ldots, x_n) =^? 0$ has a most general unifier.*

The substitution σ defined by

$$\sigma = \{x_1 \mapsto (t(0, x_2, \ldots, x_n) \oplus t(1, x_2, \ldots, x_n) \oplus 1) \cdot x_1 \oplus t(0, x_2, \ldots, x_n)\}\sigma',$$

where σ' is a most general unifier of

$$t(0, x_2, \ldots, x_n) \cdot t(1, x_2, \ldots, x_n) =^? 0$$

is a most general unifier of $t =^? 0$.

Proof: The result is proved by induction on the number n of variables in t.

If $n = 0$, then t is either 1 and the equation has no solution, or t is 0 and the identity is a unifier which is reproductive.

Let us assume now that any equation with $n - 1$ variables has a reproductive unifier.

Let $a = t(0, x_2, \ldots, x_n) \oplus t(1, x_2, \ldots, x_n)$ and $b = t(0, x_2, \ldots, x_n)$. We can check that $\forall x_1, \ldots, x_n,\ t(x_1, \ldots, x_n) =_{BR} a \cdot x_1 \oplus b$: either $x_1 = 0$ and $t(0, x_2, \ldots, x_n) =_{BR} b$, or $x_1 = 1$ and $t(1, x_2, \ldots, x_n) =_{BR} t(0, x_2, \ldots, x_n) \oplus t(1, x_2, \ldots, x_n) \oplus t(0, x_2, \ldots, x_n)$.
Moreover σ can be written as $\sigma = \{x_1 \mapsto (a \oplus 1) \cdot x_1 \oplus b\}\sigma'$, or again $\sigma = \{x_1 \mapsto (\sigma'(a) \oplus 1) \cdot \sigma'(x_1) \oplus \sigma'(b)\} \cup \sigma'$, since $\mathcal{D}om(\sigma') = \{x_2, \ldots, x_n\} = \mathcal{V}ar(a) = \mathcal{V}ar(b)$. Note also that σ' being defined on $\{x_2, \ldots, x_n\}$, $\sigma(a) = \sigma'(a)$ and $\sigma(b) = \sigma'(b)$.
Moreover let us prove that $(\sigma'(a) \cdot \sigma'(b)) \oplus \sigma'(b) =_{BR} 0$:
since $(a \cdot b) \oplus b = ((t(0, x_2, \ldots, x_n) \oplus t(1, x_2, \ldots, x_n)) \cdot t(0, x_2, \ldots, x_n)) \oplus t(0, x_2, \ldots, x_n)$
$=_{BR} (t(0, x_2, \ldots, x_n) \cdot t(0, x_2, \ldots, x_n)) \oplus (t(1, x_2, \ldots, x_n) \cdot t(0, x_2, \ldots, x_n)) \oplus t(0, x_2, \ldots, x_n)$
$=_{BR} (t(1, x_2, \ldots, x_n) \cdot t(0, x_2, \ldots, x_n))$, and by definition of σ', we get the result.

With these notations, let us prove that σ is a solution of $t(x_1, \ldots, x_n) =^? 0$ or equivalently of $a \cdot x_1 \oplus b =^? 0$.
$(\sigma(a) \cdot \sigma(x_1)) \oplus \sigma(b) = (\sigma(a) \cdot ((\sigma'(a) \oplus 1) \cdot \sigma'(x_1) \oplus \sigma'(b))) \oplus \sigma(b)$
$=_{BR} (\sigma'(a) \cdot ((\sigma'(a) \oplus 1) \cdot \sigma'(x_1) \oplus \sigma'(b))) \oplus \sigma'(b)$
$=_{BR} (\sigma'(a) \cdot ((\sigma'(a) \oplus 1) \cdot \sigma'(x_1))) \oplus (\sigma'(a) \cdot \sigma'(b)) \oplus \sigma'(b)$,
$=_{BR} (\sigma'(a) \cdot ((\sigma'(a) \oplus 1) \cdot \sigma'(x_1))) =_{BR} ((\sigma'(a) \cdot \sigma'(a)) \oplus \sigma'(a)) \cdot \sigma'(x_1) =_{BR} 0$.

The second step is to prove that σ is reproductive. Let $\underline{b}$ be a solution of $t =^? 0$.
Either $\underline{b} = (0, b_2, \ldots, b_n)$ and $\sigma(x_1)(\underline{b}) = t(0, b_2, \ldots, b_n) =_{BR} 0$,
or $\underline{b} = (1, b_2, \ldots, b_n)$ and $\sigma(x_1)(\underline{b}) = t(0, b_2, \ldots, b_n) \oplus t(0, b_2, \ldots, b_n) \oplus 1 =_{BR} 1$.
Since we have assumed σ' reproductive, and $\underline{b}$ being a solution of $b =^? 0$

$$(\sigma(x_1)(\underline{b}), \sigma'(x_2)(\underline{b}), \ldots, \sigma'(x_n)(\underline{b})) = (\sigma(x_1)(\underline{b}), b_2, \ldots, b_n) = (b_1, \ldots, b_n).$$

□

Example 13.10 Consider the term $t = (x_1 \oplus x_2) \cdot x_1$. Then $t(0, x_2) = 0$ and $t(1, x_2) = (1 \oplus x_2)$. The most general unifier is given by

$$\sigma = \{x_1 \mapsto (0 \oplus 1 \oplus x_2 \oplus 1) \cdot x_1\}\sigma'$$

where σ' is the most general unifier of $0 \cdot (1 \oplus x_2) =^? 0$. Since $0 \cdot (1 \oplus x_2) =_{BR} 0$, the identity substitution can be chosen for σ'. After simplification, $\sigma = \{x_1 \mapsto x_2 \cdot x_1\}$. Indeed one can verify that

$$\sigma(t) = ((x_2 \cdot x_1) \oplus x_2) \cdot (x_2 \cdot x_1) =_{BR} (x_2 \cdot x_1) \oplus (x_2 \cdot x_1) =_{BR} 0.$$

Example 13.11 Let us prove that the equation $x \oplus y \oplus x \cdot y \oplus c =^? 0$ is satisfiable in the free boolean ring generated by c. Let $t(x, y) = x \oplus y \oplus x \cdot y \oplus c$. Then $t(0, y) = y \oplus c$ and $t(1, y) = 1 \oplus c$. Let consider now $u(y) = (y \oplus c) \cdot (1 \oplus c) = y \oplus c \cdot y$. $u(0)$ being 0, the equation $u(0) \cdot u(1) =^? 0$ is satisfiable, so the initial equation is satisfiable too.

Let us now compute the most general unifier using the previous method. The most general unifier for $u(y) =^? 0$ is $\sigma_1(y) = ((u(0) \oplus u(1) \oplus 1) \cdot y \oplus u(0)) = (c \cdot y)$.

The most general unifier for $t(x, y) =^? 0$ is given by:
$\sigma(y) = \sigma_1(y) = c \cdot y)$
$\sigma(x) = (t(0, \sigma_1(y)) \oplus t(1, \sigma_1(y)) \oplus 1) \cdot x \oplus t(0, \sigma_1(y))$
$= (t(0, c \cdot y) \oplus t(1, c \cdot y) \oplus 1) \cdot x \oplus (t(0, c \cdot y))$
$=_{BR} ((c \cdot y \oplus c) \oplus (1 \oplus (c \cdot y) \oplus (c \cdot y)) \oplus c \oplus 1) \cdot x_1 \oplus (c \cdot y \oplus c)$
$=_{BR} ((c \cdot y) \cdot x \oplus (c \cdot y) \oplus c$.

The previous theorem directly provides an algorithm, described by Büttner and Simonis [BS87]. The algorithm transforms the equation to solve into an equivalent problem in which a selected variable has been eliminated. This transformation is iterated until a trivial problem is obtained.

Notation: Let us denote $t_{x=0}$ the result of applying to the term t the substitution $(x \mapsto 0)$ and respectively by $t_{x=1}$ the result of applying to the term t the substitution $(x \mapsto 1)$.

```
$D := t$
$\sigma :=$ Identity
If $D =_{BR} 1$ then $t =^? 0$ unsatisfiable
else while $D \neq_{BR} 0$ do
          choose a new $x \in \mathcal{V}ar(D)$
          $\sigma := (x \mapsto D_{x=0} \oplus x \cdot (D_{x=0} \oplus D_{x=1} \oplus 1)) \circ \sigma$
          $D := D_{x=0} \cdot D_{x=1}$
          enddo
endif
return $\sigma$
```

Example 13.12 Let us solve the equation

$$x_1 \oplus x_2 \oplus (x_1 \cdot x_2) =^? 0$$

Eliminating first x_1 gives $\sigma := (x_1 \mapsto x_2 \oplus x_1 \cdot x_2)$ since
$D_{x_1=0} \oplus x_1 \cdot (D_{x_1=0} \oplus D_{x_1=1} \oplus 1) = x_2 \oplus x_1 \cdot (x_2 \oplus 1 \oplus x_2 \oplus x_2 \oplus 1) = x_2 \oplus x_1 \cdot x_2$.
The new equation to solve is now $x_2 =^? 0$ since: $D_{x_1=0} \cdot D_{x_1=1} = x_2 \cdot (1 \oplus x_2 \oplus x_2) = x_2$.
Eliminating then x_2 gives $\sigma := (x_2 \mapsto 0)(x_1 \mapsto x_2 \oplus x_1 \cdot x_2)$ and $D_{x_1=0} \cdot D_{x_1=1} = 0$.
The returned substitution is thus $\sigma := (x_2 \mapsto 0)(x_1 \mapsto 0)$.

In this algorithm, the choice of the variable that is eliminated is not determined. The following example shows that different most general unifiers can be obtained according to different choices for eliminating variables.

Example 13.13 Consider the equation $x_1 \cdot x_2 \oplus x_3 =^? 0$. Eliminating first x_3 leads to $\sigma := (x_3 \mapsto x_1 \cdot x_2)$ since $D_{x_3=0} \oplus x_3 \cdot ((D_{x_3=0} \oplus D_{x_3=1} \oplus 1) = x_1 \cdot x_2 \oplus x_3 \cdot (x_1 \cdot x_2 \oplus x_1 \cdot x_2 \oplus 1 \oplus 1) = x_1 \cdot x_2$.

Then $D := 0$ and the returned substitution is $\sigma_1 = (x_3 \mapsto x_1 \cdot x_2)$.

Choosing instead to eliminate first x_1 leads to $\sigma := (x_1 \mapsto x_3 \oplus (x_1 \cdot x_2) \oplus x_1)$ and $D_{x_1=0} \cdot D_{x_1=1} = x_2 \cdot x_3 \oplus x_3$.

Eliminating then x_2 leads to $\sigma := (x_2 \mapsto x_3 \oplus (x_2 \cdot x_3) \oplus x_2)(x_1 \mapsto x_3 \oplus (x_1 \cdot x_2) \oplus x_1)$. Then $D := x_3 \cdot (x_3 \oplus x_3) =_{BR} 0$ and the returned substitution is $\sigma_2 = (x_1 \mapsto x_3 \oplus x_1 \cdot (x_3 \oplus (x_2 \cdot x_3) \oplus x_2) \oplus x_1)(x_2 \mapsto x_3 \oplus (x_2 \cdot x_3) \oplus x_2)$.

Let us denote by u_1 the boolean term $x_3 \oplus x_1 \cdot (x_3 \oplus (x_2 \cdot x_3) \oplus x_2) \oplus x_1$ and by u_2 the boolean term $x_3 \oplus (x_2 \cdot x_3) \oplus x_2$. Defining now $\alpha = (x_1 \mapsto u_1)(x_2 \mapsto u_2)$, it is easy to verify that $\alpha\sigma_1 = \sigma_2[\{x_1, x_2, x_3\}]$, since $u_1 \cdot u_2 =_{BR} x_3$.

Conversely defining $\beta = (x_3 \mapsto x_1 \cdot x_2)(x_1 \mapsto x_1)(x_2 \mapsto x_2)$, it is easy to verify that $\beta\sigma_2 = \sigma_1[\{x_1, x_2, x_3\}]$.
This proves that the two computed unifiers are equivalent.

Lövenheim's method

The Löwenheim's algorithm consists of finding a particular solution of the equation and substituting it in a general formula.

Theorem 13.3 *Let $\underline{b}$ be a particular solution of the equation $t(\underline{x}) =^? 0$. Then the vector of functions*

$$\sigma(\underline{x}) = \underline{x} \oplus t(\underline{x}) \cdot (\underline{x} \oplus \underline{b})$$

is a most general solution of the equation.

In the formulation of the previous theorem, note that the definition of $\oplus$ and $\cdot$ has been extended to operate on vectors of functions. To illustrate how this method works, let us consider an example from [MN89].

Example 13.14 Let $c{\cdot}x{\oplus}d{\cdot}y{\oplus}c =^? 0$ be the equation to solve. A particular solution is $(x \mapsto 1)(y \mapsto 0)$. So the most general solution computed with Löwenheim's formula is: $\sigma(\underline{x}) = (x,y){\oplus}c{\cdot}x{\oplus}d{\cdot}y{\oplus}c{\cdot}((x,y){\oplus}(1,0)) = (x{\oplus}((c{\cdot}x){\oplus}(d{\cdot}y){\oplus}c){\cdot}(x{\oplus}1), y{\oplus}((c{\cdot}x){\oplus}(d{\cdot}y){\oplus}c){\cdot}y) = (x{\oplus}(d{\cdot}x{\cdot}y){\oplus}(c{\cdot}x){\oplus}(d{\cdot}y){\oplus}c, y{\oplus}(c{\cdot}x{\cdot}y){\oplus}(d{\cdot}y){\oplus}(c{\cdot}y))$.

The problem left aside is to find a particular solution for a given boolean equation, which is known to be an NP-complete problem. More development on this point and further references can be found in [MN89].

A quick comparison of Löwenheim and Boole's methods immediately reveals that the first one provides more complex substitutions than the first. This is partly due to the fact that the expression of the result contains always as many variables as in the initial equation, which is not the case for Boole's method. Solving the equation $x \oplus y =^? 0$ illustrates this remark:

Example 13.15 For the equation $x{\oplus}y =^? 0$, Boole's method yields the most general unifier $\sigma(x,y) = (y,y)$ if x is eliminated first or $\sigma(x,y) = (x,x)$ if y is eliminated first. With Löwenheim's formula, choosing the particular solution $(x,y) = (0,0)$ gives $\sigma_0(x,y) = (x{\cdot}y, x{\cdot}y)$ and choosing the particular solution $(x,y) = (1,1)$ gives $\sigma_1(x,y) = (x \cdot y \oplus x \oplus y, x \cdot y \oplus x \oplus y)$. Note that $\sigma_1\sigma_0 = \sigma_1[\{x,y\}]$ and $\sigma_0\sigma_1 = \sigma_0[\{x,y\}]$, which proves that these two unifiers are equivalent.

13.2.4 Boolean algebras

Several presentations for boolean algebras laws can be given, but all of them include axioms for associativity and commutativity of $+$ and $\cdot$, distributivity, absorption and complementation laws, and for unit and zero elements. For instance, boolean algebras are models of the following set of axioms BA built on the set of function symbols $\mathcal{F}_{\mathcal{BA}} = \{0, 1, \cdot, +, \bar{}\}$:

$$\begin{array}{rcl}
x + 0 & = & x \\
x \cdot 1 & = & x \\
x \cdot (y + z) & = & (x \cdot y) + (x \cdot z) \\
x + (y \cdot z) & = & (x + y) \cdot (x + z) \\
(x + y) \cdot y & = & y \\
(x \cdot y) + y & = & y \\
x + \bar{x} & = & 1 \\
x \cdot \bar{x} & = & 0 \\
x \cdot y & = & y \cdot x \\
(x \cdot y) \cdot z & = & x \cdot (y \cdot z) \\
x + y & = & y + x \\
(x + y) + z & = & x + (y + z)
\end{array}$$

Definition 13.5 A boolean algebra $\mathcal{B}$ is defined by a non-empty carrier B and a set of operators $\mathcal{F}_{\mathcal{BA}}$ and is a model of BA.

The two first axioms of BA can be replaced by the following ones:

$$\begin{array}{rcl}
x + 1 & = & 1 \\
x \cdot 0 & = & 0
\end{array}$$

Some other properties also hold in boolean algebras:

$$\begin{array}{rcl}
x + x & = & x \\
x \cdot x & = & x \\
\overline{x + y} & = & \bar{x} \cdot \bar{y} \\
\overline{x \cdot y} & = & \bar{x} + \bar{y} \\
\bar{\bar{x}} & = & x
\end{array}$$

But in order to get a decision procedure for equality of boolean formulas through rewriting, the structure of boolean ring is needed. This can be obtained by defining a new operator $\oplus$ related to the previous ones by the following axiom:

$$x \oplus y = (x + y) \cdot \overline{(x \cdot y)}$$

or equivalently

$$x \oplus y = (\bar{x} \cdot y) + (x \cdot \bar{y})$$

So a first possible method to solve equations in boolean algebra is to enrich the signature with the new symbol $\oplus$, to use the previous unification method in boolean rings and to translate back the solution by eliminating $\oplus$. Another method is described below in the more general framework of primal algebras.

13.2.5 Primal algebras

We now generalize the class of boolean algebras to primal algebras, and propose another unification method that further allows doing unification in finite algebras. An algebra is finite when its carrier and its set of functions are both finite. It can be given a richer structure of primal algebra, in which every finitary function on the carrier can be composed from the basic operations. The 2-elements Boolean algebra is the simplest example of primal algebra, since every truth-function can be expressed in terms of the basic connectives, for instance $\cdot$ (and) and $^-$ (not).

Let us first make clear the relationship between finite algebras, primal algebras and Boolean algebras. The main result is Theorem 13.5 that states an isomorphism between a primal algebra and an adequate term algebra.

Let $\mathcal{F}$ be a set of function symbols, $\mathcal{X}$ a set of variables, and $\mathcal{A}$ an $\mathcal{F}$-algebra, whose carrier is denoted by A. $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the free $\mathcal{F}$-algebra over $\mathcal{X}$.

An assignment α is a mapping from $\mathcal{X}$ to A; it uniquely extends to an homomorphism α from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ to $\mathcal{A}$. The set of all assignments is denoted by $ASS_A^{\mathcal{X}}$ or ASS_A, when $\mathcal{X}$ is clear from the context.

A term t built on a set of function symbols $\mathcal{F}$ and m variable symbols in an ordered set of variables $\mathcal{X}$, defines a function $t^{\mathcal{A}} : A^m \to A$ as follows:

$$\forall (a_1, \ldots, a_m) \in A^m,\ t^{\mathcal{A}}(a_1, \ldots, a_m) = \alpha(t),$$

where α is an assignment such that $\forall i \in [1 \ldots m],\ \alpha(x_i) = a_i$ (also denoted by $(x_i \mapsto a_i)$). Conversely,

Definition 13.6 [Nip90b] An $\mathcal{F}$-algebra $\mathcal{A}$ is *primal* if any finitary function on its carrier A with an arity greater than 0 is equal to $t^{\mathcal{A}}$ for some t in $T(\mathcal{F}, \mathcal{X})$.

Given a primal $\mathcal{F}$-algebra $\mathcal{A}$ such that $\mathcal{F}$ is a finite set of finitary function symbols, its carrier A is necessarily finite. In the sequel, only finite primal algebras are considered. To any finite algebra, we can associate a primal algebra with the same carrier and an extended set of function symbols.

Definition 13.7 Given the $\mathcal{F}$-algebra $\mathcal{A}$ with the carrier $A = \{0, \ldots, n-1\}$, the *enriched finite algebra* $\overline{\mathcal{A}}$ is defined by the carrier A, and the set of function symbols

$$\overline{\mathcal{F}} = \mathcal{F} \cup \{\perp, [1], \ldots, [n-2], \top, C_0, \ldots, C_{n-1}, +, \cdot\}$$

interpreted as follows:

$\perp_A = 0$
$\top_A = n - 1$
$\forall i \in A \setminus \{0, n-1\},\ [i]_A = i$
$\forall i \in A, \forall x \in A,\ C_{iA}(x) = \text{ if } x = i \text{ then } n-1 \text{ else } 0$
$\forall (x, y) \in A^2,\ x +_A y = \max(x, y)$
$\forall (x, y) \in A^2,\ x \cdot_A y = \min(x, y).$

Example 13.16 The algebra defined by the carrier $A = \{0, 1\}$ together with the set of additional operators of Definition 13.7 is the 2-elements Boolean algebra. C_0 corresponds to $^-$ (not) and C_1 is the identity.

The algebra $\overline{\mathcal{A}}$ is primal [Nip90b], since any function $f : A^m \to A$ is equal to the functional interpretation of the term

$$\sum_{(a_1, \ldots, a_m) \in A^m} C_{a_1}(x_1) \cdots C_{a_m}(x_m) \cdot [f(a_1, \ldots, a_m)] \qquad (POST)$$

where $[f(a_1,\dots,a_m)]$ denotes the operator corresponding to the value taken by the function f on $(a_1,\dots,a_m)$. Intuitively, this term represents the truth table of the function f. It may be obtained by decomposing first
$f(x_1,\dots,x_m) = C_0(x_1)\cdot f(0,x_2,\dots,x_m) + C_1(x_1)\cdot f(1,x_2,\dots,x_m) + \dots C_{n-1}(x_1)\cdot f(n-1,x_2,\dots,x_m)$ and iterating further the decomposition in each term of the sum.

We now exhibit a finite set AF of equational axioms such that each term $t \in T(\overline{\mathcal{F}},\mathcal{X})$, is equal modulo AF to a specific canonical form, which is the $POST$ decomposition of $t^{\overline{\mathcal{A}}}$.

Definition 13.8 Let AF be the *finite set of axioms* on $\mathcal{T}(\overline{\mathcal{F}},\mathcal{X})$:

$$\begin{array}{rclcrcl}
x+(y+z) & = & (x+y)+z & & x+\bot & = & x\\
x+y & = & y+x & & x+\top & = & \top\\
x\cdot(y\cdot z) & = & (x\cdot y)\cdot z & & x\cdot\bot & = & \bot\\
x\cdot y & = & y\cdot x & & x\cdot\top & = & x\\
x\cdot(y+z) & = & x\cdot y + x\cdot z & & x+x & = & x\\
x+(y\cdot z) & = & (x+y)\cdot(x+z) & & x\cdot x & = & x
\end{array}$$

$$\begin{array}{rcl}
\forall f\in\overline{\mathcal{F}}_p, \forall i\in A,\ C_i(f(x_1,\dots,x_p)) & = & \displaystyle\sum_{f_A(i_1,\dots,i_p)=i} C_{i_1}(x_1)\cdots C_{i_p}(x_p)\\
\forall i\in A,\ C_i([i]) & = & \top\\
\forall (i,j)\in A^2, i\neq j,\ C_i(x)\cdot C_j(x) & = & \bot\\
\displaystyle\sum_{i=0}^{n-1} C_i(x) & = & \top\\
\displaystyle\sum_{i=0}^{n-1} C_i(x)\cdot[i] & = & x
\end{array}$$

Example 13.17 If we consider the carrier size $n=2$ and $\mathcal{F}=\emptyset$, the set of axioms given above generates the Boolean theory.

In order to simplify notation, the product $\prod_{x\in V} C_{\alpha(x)}(x)$ will be denoted by $\prod \alpha(V)$ and called atom, which is a usual terminology in Boolean algebras.

Theorem 13.4 *[KR94a] Any term t in $T(\overline{\mathcal{F}},\mathcal{X})$ is equal modulo AF to its* canonical form*:*

$$\sum_{\{\alpha:\mathcal{V}(t)\to A\}} \prod_{x\in\mathcal{V}(t)} C_{\alpha(x)}(x)\cdot[\alpha(t)].$$

Example 13.18 With $A=\{0,1\}$, the term $t = x+y\cdot z$ is decomposed into the following canonical form:
$t =_{AF} C_0(x)\cdot C_0(y)\cdot C_0(z)\cdot 0 + C_0(x)\cdot C_0(y)\cdot C_1(z)\cdot 0 + C_0(x)\cdot C_1(y)\cdot C_0(z)\cdot 0 + C_0(x)\cdot C_1(y)\cdot C_1(z)\cdot 1 + C_1(x)\cdot C_0(y)\cdot C_0(z)\cdot 1 + C_1(x)\cdot C_0(y)\cdot C_1(z)\cdot 1 + C_1(x)\cdot C_1(y)\cdot C_0(z)\cdot 1 + C_1(x)\cdot C_1(y)\cdot C_1(z)\cdot 1.$
$=_{AF} C_0(x)\cdot C_1(y)\cdot C_1(z) + C_1(x)\cdot C_0(y)\cdot C_0(z) + C_1(x)\cdot C_0(y)\cdot C_1(z) + C_1(x)\cdot C_1(y)\cdot C_0(z) + C_1(x)\cdot C_1(y)\cdot C_1(z).$

Another notation is often used in Boolean algebras with the convention that $C_0(v)=\overline{v}$ and $C_1(v)=v$ for any variable v. Then $t =_{AF} \overline{x}\cdot y\cdot z + x\cdot\overline{y}\cdot\overline{z} + x\cdot\overline{y}\cdot z + x\cdot y\cdot\overline{z} + x\cdot y\cdot z$.

As usual with Boolean algebras, the empty sum is by definition equal to $\bot$ and the empty product equal to $\top$. Then, if t is a ground term, we have

$$t =_{AF} \sum_{\{\alpha:\emptyset\to A\}} \top\cdot[\alpha(t)] =_{AF} [\alpha(t)],$$

where α denotes the unique homomorphism from $\mathcal{T}(\overline{\mathcal{F}})$ to $\overline{\mathcal{A}}$ since $\mathcal{T}(\overline{\mathcal{F}})$ is the initial algebra of the class of $\overline{\mathcal{F}}$-algebras.

The canonical form of t must be compared to the previous $POST$ decomposition where f corresponds to $t^{\overline{\mathcal{A}}}$ and $f(a_1,\dots,a_m)$ to $\alpha(t)$.

Theorem 13.4 leads to the next result, useful in the context of constraint solving in primal algebras, since it justifies to work at the level of terms instead of functions and values. It explains in particular why unification in the 2-elements Boolean algebra performs unification in the class of Boolean algebras. A similar property holds between the enriched finite algebra $\overline{\mathcal{A}}$ and the class of models satisfying axioms in AF.

Theorem 13.5 *[KR94a] The $\overline{\mathcal{F}}$-algebras $\overline{\mathcal{A}}$ and $\mathcal{T}(\overline{\mathcal{F}}, \mathcal{X})/=_{AF}$ have the same equational theorems: for any universally quantified equality $(t = t')$, $\overline{\mathcal{A}} \models (t = t')$ iff $t =_{AF} t'$. Moreover, $\overline{\mathcal{A}}$ and $\mathcal{T}(\overline{\mathcal{F}})/=_{AF}$ are isomorphic.*

Corollary 13.1 The presentation $(\overline{\mathcal{F}}, AF)$ is ω-complete, i.e. the algebras $\mathcal{T}(\overline{\mathcal{F}}, \mathcal{X})/=_{AF}$ and $\mathcal{T}(\overline{\mathcal{F}})/=_{AF}$ have the same equational theorems: for any universally quantified equality $(t = t')$, $\mathcal{T}(\overline{\mathcal{F}})/=_{AF} \models (t = t')$ iff $t =_{AF} t'$.

Theorems 13.4 and 13.5 can be obtained as consequences of more general results on the variety of primal algebras given in [Fos53].

Unification in primal algebras has been studied in [Nip90b] by generalizing algorithms for solving equations in finite Boolean rings and algebras. We give here a different proof technique derived from [BES$^+$90], where a method is proposed for computing a most general unifier in a primal algebra whose carrier is of cardinality n. We describe the method in the case of equation solving, but it works as well if other constraints expressed with predicates defined on the carrier of the algebra are considered.

In the context of a finite algebra, the set $Sol(e)$ of solutions of an equation e is usually easy to compute since the carrier A is finite: just consider all assignments of variables to their possible values and check for each of them whether the equation is satisfied. But we are rather interested in a more compact representation of the set of solutions, provided by a complete set of unifiers, or even better by a most general unifier. To analyze the problem, let us first characterize a most general unifier σ of e thanks to a surjective mapping between assignments from $ASS_A^{\mathcal{V}(\sigma(e))}$ to $Sol(e)$.

For a given e, a substitution σ defines a mapping $\sigma_e : ASS_A^{\mathcal{V}(\sigma(e))} \mapsto ASS_A^{\mathcal{V}(e)}$, which maps any $\alpha \in ASS_A^{\mathcal{V}(\sigma(e))}$ to the assignment defined by $\forall x \in \mathcal{V}(e),\ \sigma_e(\alpha)(x) = \alpha(\sigma(x))$. This relation extends by straightforward induction, to terms built on $\mathcal{V}(e)$.

Let $\mathcal{R}an(\sigma_e)$ denote the range of σ_e: $\mathcal{R}an(\sigma_e) = \{\sigma_e(\alpha) | \alpha \in ASS_A^{\mathcal{V}(\sigma(e))}\}$.

Example 13.19 In the 2-elements Boolean algebra, consider the equation $e = (x =^? x + y)$ and the substitution $\sigma = \{y \mapsto x\}$. σ_e maps the assignment $(x \mapsto 0)$ onto $(x \mapsto 0)(y \mapsto 0)$ and $(x \mapsto 1)$ onto $(x \mapsto 1)(y \mapsto 1)$.

The next result reduces the symbolic solving problem to a necessary and sufficient condition on σ_e.

Proposition 13.2 A substitution σ is a unifier of the equation e if and only if $\mathcal{R}an(\sigma_e) \subseteq Sol(e)_{|\mathcal{V}(e)}$. If there exists a substitution σ such that $\mathcal{R}an(\sigma_e) = Sol(e)_{|\mathcal{V}(e)}$, σ is a most general unifier of e.

Proof: Let us assume that e is $(t_1 =^? t_2)$. A substitution σ is a unifier of e if and only if $\forall \alpha, \alpha(\sigma(t_1)) = \alpha(\sigma(t_2))$
if and only if $\forall \alpha, \sigma_e(\alpha)(t_1) = \sigma_e(\alpha)(t_2)$ if and only if $\mathcal{R}an(\sigma_e) \subseteq Sol(e)_{|\mathcal{V}(e)}$. Indeed the equation e has no unifier if $Sol(e)$ is empty. Furthermore, there is a very simple way to be sure that two substitutions are comparable with $\leq_{AF}^{\mathcal{V}(e)}$. Let σ and σ' be two substitutions and e an equation. let us first prove that

$$\mathcal{R}an(\sigma'_e) \subseteq \mathcal{R}an(\sigma_e)$$

iff

$$\sigma \leq_{AF}^{\mathcal{V}(e)} \sigma'.$$

($\Rightarrow$) An assignment from $\mathcal{V}(\sigma'(e))$ to A is denoted by α'. By assumption, we have

$$\forall \alpha' \exists \alpha,\ \forall x \in \mathcal{V}(e), \alpha'(\sigma'(x)) = \sigma'_e(\alpha')(x) = \sigma_e(\alpha)(x) = \alpha(\sigma(x)).$$

So there exists

$$u : ASS_A^{\mathcal{V}(\sigma'(e))} \to ASS_A^{\mathcal{V}(\sigma(e))}$$

such that

$$\forall \alpha'\ \forall x \in \mathcal{V}(e),\ \alpha'(\sigma'(x)) = u(\alpha')(\sigma(x)).$$

Let μ be the substitution

$$\{x \mapsto \sum_{\{\alpha' : \mathcal{V}(\sigma'(e)) \to A\}} \prod \alpha'(\mathcal{V}(\sigma'(e))) \cdot [u(\alpha')(x)]\}_{x \in \mathcal{V}(\sigma(e))}.$$

According to the definition of additional operators in $\overline{\mathcal{F}}$, we have $\mu_{\sigma(e)} = u$. Then

$$\forall \alpha' \ \forall x \in \mathcal{V}(e), \ \alpha'(\mu(\sigma(x))) = \mu_{\sigma(e)(\alpha')}(\sigma(x)) = u(\alpha')(\sigma(x)) = \alpha'(\sigma'(x)).$$

So in the equational theory AF,

$$\forall x \in \mathcal{V}(e), \ \sigma'(x) =_{AF} \mu(\sigma(x)).$$

($\Leftarrow$) The notation is analogous to the if-part. Let us assume without loss of generality that $\mathcal{V}(\mu(\sigma(e))) = \mathcal{V}(\sigma'(e))$.

$$\begin{aligned} \forall \alpha' \ \forall x \in \mathcal{V}(e), \ \sigma'_e(\alpha')(x) &= \alpha'(\sigma'(x)) \\ &= \alpha'(\mu(\sigma(x))) \\ &= \mu_{\sigma(e)}(\alpha')(\sigma(x)) \\ &= \sigma_e(\mu_{\sigma(e)}(\alpha'))(x). \end{aligned}$$

That is $\mathcal{R}an(\sigma'_e) \subseteq \mathcal{R}an(\sigma_e)$.

We get the result as corollary: if there exists a substitution σ such that $\mathcal{R}an(\sigma_e) = Sol(e)_{|\mathcal{V}(e)}$, σ is a (unique) most general unifier of e. □

Now the problem is to prove the existence of such a substitution σ. This is done by giving explicitly the construction of a mapping σ_e from assignments of new variables Y (introduced to express all assignments $\alpha : \mathcal{V}(\sigma(e)) \mapsto A$) to assignments of variables $\mathcal{V}(e)$. The number of new variables in Y must be chosen as small as possible but satisfying the condition $n^{|Y|} \geq |Sol(e)_{|\mathcal{V}(e)}|$. Indeed since σ_e is a mapping, we necessarily have

$$|ASS^Y_A| \geq |\mathcal{R}an(\sigma_e)| = |Sol(e)_{|\mathcal{V}(e)}|.$$

Moreover $|ASS^Y_A|$ is equal to $|A|^{|Y|}$ where $|A| = n$. In the worst case, $|Y|$ is equal to $|\mathcal{V}(e)|$. Then any surjective mapping of ASS^Y_A onto $Sol(e)_{|\mathcal{V}(e)}$ can be used as the mapping σ_e.

Example 13.20 In the 2-elements Boolean algebra, consider the equation $e = (x + y \cdot z =^? x \cdot y \cdot z)$. An assignment (for instance $\beta = (x \mapsto 0)(y \mapsto 0)(z \mapsto 0)$) is next abusively denoted by its atom ($\overline{x} \cdot \overline{y} \cdot \overline{z}$ for β)). The reader can check that $Sol(e) = \{\overline{x} \cdot \overline{y} \cdot \overline{z}, \overline{x} \cdot \overline{y} \cdot z, \overline{x} \cdot y \cdot \overline{z}, x \cdot y \cdot z\}$.

Since $|Sol(e)| = 4$, the condition $2^{|Y|} \geq 4$ implies $|Y| = 2$ as the smallest possibility. So let us consider two new variables y_1 and y_2. Then the mapping σ_e can be chosen as follows:

$$\sigma_e(\overline{y_1} \cdot \overline{y_2}) = \overline{x} \cdot \overline{y} \cdot \overline{z} \quad \sigma_e(\overline{y_1} \cdot y_2) = \overline{x} \cdot \overline{y} \cdot z \quad \sigma_e(y_1 \cdot \overline{y_2}) = \overline{x} \cdot y \cdot \overline{z} \quad \sigma_e(y_1 \cdot y_2) = x \cdot y \cdot z$$

We are now able to explicit a most general unifier, thanks to the canonical form of $\sigma(x)$ in the theory AF, for each $x \in \mathcal{V}(e)$.

Theorem 13.6 *Let e be an equation, Y a finite set of variables disjoint of $\mathcal{V}(e)$ and σ_e a mapping from ASS^Y_A to $ASS^{\mathcal{V}(e)}_A$ such that $\mathcal{R}an(\sigma_e) = Sol(e)_{|\mathcal{V}(e)}$. The substitution*

$$\sigma = \{x \mapsto \sum_{\{\alpha : Y \to A\}} \prod \alpha(Y) \cdot [\sigma_e(\alpha)(x)]\}_{x \in \mathcal{V}(e)}$$

is a most general unifier of e.

Proof: According to Theorem 13.4, for any $x \in \mathcal{V}(e)$,

$$\sigma(x) =_{AF} \sum_{\{\alpha : Y \to A\}} \prod \alpha(Y) \cdot [\alpha(\sigma(x))]$$

and $\alpha(\sigma(x)) = \sigma_e(\alpha)(x)$ by construction. The property of σ to be a most general unifier results from Proposition 13.2. □

Example 13.21 (Example 13.20 continued: $e = (x + y \cdot z =^? x \cdot y \cdot z)$). $\sigma_e(\alpha)(x) = 1$ if α corresponds to the atom $y_1 \cdot y_2$, $\sigma_e(\alpha)(y) = 1$ if α is $y_1 \cdot \overline{y_2}$ or $y_1 \cdot y_2$, $\sigma_e(\alpha)(z) = 1$ if α is $\overline{y_1} \cdot y_2$ or $y_1 \cdot y_2$. After simplication, we get the most general unifier

$$(x \mapsto y_1 \cdot y_2)(y \mapsto y_1)(z \mapsto y_2).$$

In order to deal with the danger of exponential complexity, inherent to boolean algebra and similar structures, a computational representation of primal algebras using directed acyclic graphs (dags) can be used. Having fixed an ordering on its variables, a function t of m variables $x_1, \ldots, x_m$ is associated to a n-ary tree: If t is a constant $0, \ldots, n-1$, then the tree consists of just one node labelled $0, \ldots, n-1$, otherwise the tree for t is obtained by creating a new node whose sons are the trees associated respectively to $t(0, x_2, \ldots, x_m), \ldots, t(n-1, x_2, \ldots, x_m)$. To get a dag, duplicated subterms are eliminated. Such a tree contains all the information on the decomposition of the function t. Each path in the tree leading to a constant $i \in [0, \ldots, n-1]$ corresponds to an atom of the $POST$ decomposition (i.e. an assignment) for which t takes the value i.

Exercice 46 — Build the binary tree associated to the boolean fonction $f(x_1, x_2, x_3) = x_1 \cdot \overline{x_2} + x_3$ and the associated dag.

Answer:

To conclude this section, several examples of unification in primal algebras are given. The first one deals with modeling digital circuits and verifying their correctness. The following example comes from [DSvH87].

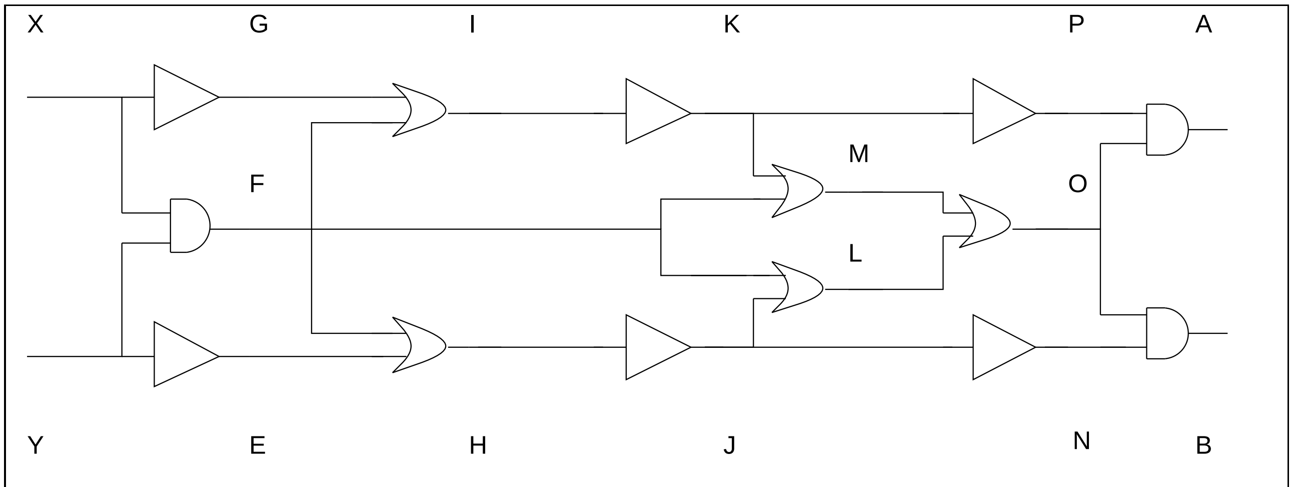


Figure 13.1: Le circuit CROSS

Example 13.22 The CROSS circuit on Figure 13.1 is only composed of AND, OR and NOT gates. It thus can be specified in Boolean algebra by the following set of equations:

$$\begin{cases} G =^? \overline{X} \\ E =^? \overline{Y} \\ F =^? X \cdot Y \\ I =^? G + F \\ H =^? E + F \\ K =^? \overline{I} \\ J =^? \overline{H} \\ M =^? K + F \\ L =^? J + F \\ O =^? L + M \\ P =^? \overline{K} \\ N =^? \overline{J} \\ A =^? P \cdot O \\ B =^? N \cdot O \end{cases}$$

The unification algorithm applied to this system of equations computes the most general unifier

$$\{X \mapsto Y_1, Y \mapsto Y_2, A \mapsto Y_2, B \mapsto Y_1\}.$$

This provides a proof of the fact that the circuit really exchanges its entries. Another possible use of this system is to express the outputs A, B (which are then the variables) with respect to entries X, Y (considered as constants in this case).

Another example is a crypto-arithmetic puzzle, where arithmetic modulo is involved.

Example 13.23 *The lion and the unicorn*:

"When Alice entered the forest of furgetfulness, she did not forget everything, only certain things. She often forgot her name, and the most likely to forget was the day of the week. Now, the lion and the unicorn were frequent visitors to this forest. These two are strange creatures. The lion lies Mondays, Tuesdays and Wednesdays and tells the truth on the other days of the week. The unicorn, on the other hand lies on Thursdays, Fridays and Saturdays, but tells the truth on the other days of the week. One day Alice met the lion and the unicorn resting under a tree. They made the following statements:

Lion: Yesterday was one of my lying days.

Unicorn: Yesterday was one of my lying days.

From these statements, Alice who was a bright girl, was able to deduce the day (d) of the week. What was it?"

In other words, the lion (resp. the unicorn) tells the truth today and lied yesterday, or lies today and told the truth yesterday.

To every day of the week is associated an integer in $[0, 6]$: 0 for monday,. . ., 6 for sunday. The operator $+_7$ is addition modulo 7. If $d \in [0, 6]$ corresponds to the day j, then $d +_7 6$ corresponds to the previous day. So the next problem has to be solved:

$$\begin{gathered}
(d = 0 \vee d = 1 \vee d = 2) \wedge (d +_7 6 = 3 \vee d +_7 6 = 4 \vee d +_7 6 = 5 \vee d +_7 6 = 6) \\
\vee\ (d +_7 6 = 0 \vee d +_7 6 = 1 \vee d +_7 6 = 2) \wedge (d = 3 \vee d = 4 \vee d = 5 \vee d = 6) \\
\bigwedge \\
(d = 3 \vee d = 4 \vee d = 5) \wedge (d +_7 6 = 0 \vee d +_7 6 = 1 \vee d +_7 6 = 2 \vee d +_7 6 = 6) \\
\vee\ (d +_7 6 = 3 \vee d +_7 6 = 4 \vee d +_7 6 = 5) \wedge (d = 0 \vee d = 1 \vee d = 2 \vee d = 6)
\end{gathered}$$

In order to translate the problem in a primal algebra, the following properties are used:

$$\begin{cases}
x = i & \Leftrightarrow \quad C_i(x) = \top \\
(t = \top) \vee (t' = \top) & \Leftrightarrow \quad t + t' = \top \\
(t = \top) \wedge (t' = \top) & \Leftrightarrow \quad t \cdot t' = \top
\end{cases}$$

So the previous problem is equivalent to:

$$\begin{cases}
\top & =^? \quad (C_0(d) + C_1(d) + C_2(d)) \cdot (C_3(d +_7 6) + C_4(d +_7 6) + C_5(d +_7 6) + C_6(d +_7 6)) \\
 & + \quad (C_0(d +_7 6) + C_1(d +_7 6) + C_2(d +_7 6)) \cdot (C_3(d) + C_4(d) + C_5(d) + C_6(d)) \\
\top & =^? \quad (C_3(d) + C_4(d) + C_5(d)) \cdot (C_0(d +_7 6) + C_1(d +_7 6) + C_2(d +_7 6) + C_6(d +_7 6)) \\
 & + \quad (C_3(d +_7 6) + C_4(d +_7 6) + C_5(d +_7 6)) \cdot (C_0(d) + C_1(d) + C_2(d) + C_6(d))
\end{cases}$$

The only solution is $d = 3$.

Example 13.24 *Inverse of a matrix in arithmetic*:

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \\ 2 & 1 & 1 \end{pmatrix}$$

$$M^{-1} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = ?$$

Carrier size : $N = 4$.

$$\begin{cases}
a +_4 2 \cdot_4 d +_4 3 \cdot_4 g & =^? \quad 1 \\
b +_4 2 \cdot_4 e +_4 3 \cdot_4 h & =^? \quad 0 \\
c +_4 2 \cdot_4 f +_4 3 \cdot_4 i & =^? \quad 0 \\
a +_4 d +_4 g & =^? \quad 0 \\
b +_4 e +_4 h & =^? \quad 1 \\
c +_4 f +_4 i & =^? \quad 0 \\
2 \cdot_4 a +_4 d +_4 g & =^? \quad 0 \\
2 \cdot_4 b +_4 e +_4 h & =^? \quad 0 \\
2 \cdot_4 c +_4 f +_4 i & =^? \quad 1
\end{cases}$$

The returned solution is

$$a = 0,\ b = 3,\ c = 1,\ d = 3,\ e = 1,\ f = 2,\ g = 1,\ h = 1,\ i = 1.$$

Chapter 14

Procedures for semantic unification

14.1 A Semidecision procedure

14.1.1 General E-unification

As we have seen, equational unification is undecidable since unification of ground terms boils down to the word problem. It is indeed semi-decidable by interleaving production of substitutions with generation of equational proofs. Gallier and Snyder gave a complete set of rules (**GS-Unify**) for computing a complete set of unifiers to a unification problem P in an arbitrary theory E [GS87, GS89, Sny88].

Delete	$P \wedge s =^?_E s$	$\mapstochar\twoheadrightarrow$	P
Decompose	$P \wedge f(\vec{s}) =^?_E f(\vec{t})$	$\mapstochar\twoheadrightarrow$	$P \wedge s_1 =^?_E t_1 \wedge \ldots \wedge s_n =^?_E t_n$
Coalesce	$P \wedge x =^?_E y$	$\mapstochar\twoheadrightarrow$	$\{x \mapsto y\}P \wedge x =^?_E y$
			if $x, y \in \mathcal{V}ar(P)$ and $x \neq y$
Eliminate	$P \wedge x =^?_E s$	$\mapstochar\twoheadrightarrow$	$\{x \mapsto s\}P \wedge x =^?_E s$
			if $x \notin \mathcal{V}ar(s)$, $s \notin \mathcal{X}$ and $x \in \mathcal{V}ar(P)$
LazyParamodulate	$P \wedge s =^?_E t$	$\mapstochar\twoheadrightarrow$	$P \wedge s\|_p =^?_E l \wedge s[r]_p =^?_E t$
			if $s\|_p \notin \mathcal{X}$ and $s\|_p(\Lambda) = l(\Lambda)$
			where $l = r \in E$

GS-Unify: Gallier and Snyder's rules for E-unification

A major tool used for E-unification is paramodulation, that is, the use of some equation in the set E. Each time such an equation is used in the coming sets of rules, the assumption is tacitly made that the variables of the equation are renamed to avoid possible captures.

Gallier and Snyder prove that for any E-unifier γ of a problem P, there exists a sequence of rules (where **Decomposition** is always applied immediately after **LazyParamodulate**) whose result is a tree solved form yielding an idempotent unifier $\sigma \leq \gamma$. In this sense, the set of rules is complete. This result is improved in [DJ90b] where a restrictive version of **LazyParamodulate** is proved to be complete. General E-unification transformations have also been given in [Höl89].

Some improvements can be made on the above set for some sets E of equations: a failing path in the tree of inferences is either infinite or else its leaf is labeled by a set of equations not in solved form. Some rules may sometimes be added to the previous set in order to obtain failing leaves: The set of *conflicting* symbols $\mathcal{C}$ is the largest subset of $\mathcal{F} \times \mathcal{F}$ such that $(f, g) \in \mathcal{C}$ if and only if the equation $f(\vec{s}) = g(\vec{t})$ has no unifier.

Conflict is, of course, sound and complete for conflicting symbols and hence can be added to the above set in order to get failure nodes. In a similar way we can introduce the set of *decomposable* symbols, for which **Decompose** is sound and complete. In this case the rules **Mutate** and **Splice** should not apply. Hence a symbol f is decomposable if there are no collapse axioms, and f is not the top function symbol of a left- or right- hand side of an equation. Decomposable symbols were introduced in [Kir84a], and conflicting ones in [Mza85, Mza86].

14.2 Narrowing

Narrowing is a relation on terms which generalize rewriting in the way that it uses unification instead of matching in order to fight a rule whose application results in the complete instanciation and one step reduction on the narrowed term. This relation has been first introducted by M. Fay in order to perform unification in equational theories presented by a confluent and terminating term rewriting system. It is one of the main application of narrowing that we are presenting in details in this section. Another application of narrowing is its use as an operational semantics of logic and functional programming languages like BABEL [MNRA92], EQLOG [GM86] and SLOG [Fri85b] among many others.

Narrowing a term is finding the minimal instanciation of it such that one rewrite step becomes applicable, and to apply it. This consists of replacing a non-variable subterm which unifies with a left-hand side of a rewrite rule by the right-hand side, and instantiating the result with the computed most general unifier. If this process is applied to an equation seen as a term with top symbol $=^?$, and is iterated until finding an equation whose both terms are syntactically unifiable, then the composition of the most general unifier with all the substitutions computed during the narrowing sequence yields a unifier of the initial equation in the equational theory. The narrowing process that builds all the possible narrowing derivations starting from the equation to be solved, is a general unification method that yields complete sets of unifiers, provided that the theory can be presented by a terminating and confluent rewrite system [Fay79, Hul80a, JKK83, MH92]. Furthermore, this method is incremental in that it allows building, from a unification algorithm in a theory A, a unification procedure for a theory $R \cup A$, provided the class rewrite system defined by R and A is Church-Rosser and terminating modulo A [JKK83].

However, the drawback of such a general method is that it very often diverges and several attempts have been made to restrict the size of the narrowing derivation tree [Hul80a, RKKL85, Rét87, DS88, You89, NRS89, KB91, Chr92]. A very successful attempt to solve this problem has been made by J.-M. Hullot in generalizing narrowing into basic narrowing. It has the main advantage to separate the solving of the unification constraint from the narrowing process itself. It is in fact a particular case of deduction with constraints, and is the first such process to have been designed [Hul80a].

In this section we present the two relations of narrowing and basic (or constraint) narrowing and their application to the unification problem in equational theories presented by terminating and confluent term rewrite system. We also shortly present the use of narrowing as operational semantics of logic and functional languages.

14.2.1 Narrowing relations

Definition 14.1 (Narrowing)

A term t is *narrowed* into t', at the non variable position $p \in \mathcal{D}om(t)$, using the rewrite rule $l \to r$ and the substitution σ, when σ is a most general unifier of $t|_p$ and l and $t' = \sigma(t[r]_p)$. This is denoted $t \rightsquigarrow_{[p,l\to r,\sigma]} t'$ and it is always assumed that there is no variable conflict between the rule and the term, i.e. that $\mathcal{V}ar(l,r) \cap \mathcal{V}ar(t) = \emptyset$.

For a given term rewriting system R, this generates a binary relation on terms called *narrowing* relation and denoted $\rightsquigarrow^R$.

Note that narrowing is a natural extension of rewriting since unification is used instead of matching. As a consequence the rewriting relation is always included in the narrowing one: $\longrightarrow^R \subseteq \rightsquigarrow^R$ since, for terms with disjoint sets of variables, a match is always a unifier.

Example 14.1 If we consider the rule $f(f(x)) \to x$ then the term $f(y)$ narrows at position Λ:

$$f(y) \rightsquigarrow_{[\Lambda, f(f(x))\to x, \{(x\mapsto z),(y\mapsto f(z))\}]} z.$$

On this example, we can notice that narrowing may introduce new variables, due to the unification step. Now, if we narrow the term $g(y, f(y))$, we get the following derivation:

$$\begin{array}{lll} g(y, f(y)) & \rightsquigarrow_{[2, f(f(x))\to x, \{(x\mapsto z),(y\mapsto f(z))\}]} & g(f(z), z) \\ & \rightsquigarrow_{[1, f(f(x))\to x, \{(x\mapsto z'),(z\mapsto f(z'))\}]} & g(z', f(z')) \\ & \ldots & \end{array}$$

which shows that even if the term rewriting system is obviously terminating, the narrowing derivation may not be so.

Exercice 47 — Use the system BasicArithmetic on page 118 to narrow the terms $succe(succe(0)) + prede(0)$, $succe(succe(x)) + prede(0)$, $succe(succe(x)) + prede(y)$.

The following notion of basic narrowing, due to J.-M. Hullot [Hul80a, Hul80c], is a restriction of the narrowing relation.

Definition 14.2 [Basic Narrowing]

Let t be a term and $\mathcal{O}$ be a set of non-variable occurences of t: $\mathcal{O} \subseteq \mathcal{G}rd(t)$. Using the rule $l \to r$ of the system R at occurence p, the narrowing step $t \rightsquigarrow_{[p,l\to r,\sigma]} t'$ is *basic* with respect to $\mathcal{O}$ if $p \in \mathcal{O}$ and the resulting set of positions associated with t' is:

$$\mathcal{O}' = \mathcal{O} \setminus \{q \mid q \geq p\} \cup \{p.q \mid q \in \mathcal{G}rd(r)\}.$$

This is denoted: $(t, \mathcal{O}) \overset{b}{\rightsquigarrow}{}^{R}_{[p,l\to r,\sigma]} (t', \mathcal{O}')$.

A *basic narrowing* derivation starting from t_0 is a sequence of basic narrowing steps, where the initial set of positions is the set of all non-variable positions of t_0:

$$(t_0, \mathcal{G}rd(t_0)) \overset{b}{\rightsquigarrow}{}^{R} (t_1, \mathcal{O}_1) \overset{b}{\rightsquigarrow}{}^{R} (t_2, \mathcal{O}_2) \overset{b}{\rightsquigarrow}{}^{R} \dots$$

In such a derivation, $\mathcal{O}_n$ is called the set of *basic positions* of t_n.

One of the reason for introducing this new narrowing relation is that it terminates more often than standard narrowing. For example coming back to the previous example, we get:

$$(g(y, f(y)), \{\Lambda, 2\}) \overset{b}{\rightsquigarrow}_{[2, f(f(x))\to x, \{(x\mapsto z),(y\mapsto f(z))\}]} (g(f(z), z), \{\Lambda\}),$$

and this last term is no more narrowable with respect to the set of positions $\{\Lambda\}$.

Exercice 48 — Use the system BasicArithmetic on page 118 to narrow basically the terms $succe(succe(0))+prede(0)$, $succe(succe(x)) + prede(0)$, $succe(succe(x)) + prede(y)$.

Following [Hul80a], we will see later that basic narrowing is complete for equation solving modulo equational theories presented by a confluent and terminating term rewriting system.

Let us come now to another notion of narrowing which generalizes the previous narrowing relations and allows us to give a more convenient definition of basic narrowing. We first need to introduce constrained terms which are a special case of constrained formulas (see Section 7.6.2).

Definition 14.3 A *constraint term* $(\exists W, t \parallel c)$ is a couple made of a term t and a system of constraints c together with a set of existentially quantified variables W. It schematizes the set of all instances of t by a solution of c with no assumption on W, i.e.

$$\{\exists W, \sigma(t) \mid \sigma \in \mathcal{S}ol(\exists W, c)\}.$$

The set of free variables of a constraint term $(\exists W, t \parallel c)$ is the union of the set of free variables of t and the free variables set of c minus W: $\mathcal{V}ar((\exists W, t \parallel c)) = \mathcal{V}ar(t) \cup \mathcal{V}ar(c) \setminus W$. When W is the empty set, $(\exists W, t \parallel c)$ is simply denoted $(\exists, t \parallel c)$.

We consider in this chapter only constraint consisting of system of syntactic equations. This can be extended to more general constraint languages and domains as proposed for example in [KK89, KKR90, Cha94] and explained in Section 7.6.

Example 14.2 The formula $\tau = (f(x, f(a, x)) \parallel \exists z, f(x, z) =^? f(g(a, y), f(a, y)) \wedge y =^? g(u, b))$ is a constraint term. It schematizes the terms:

$$\begin{array}{l} f(g(a, g(u, b)), f(a, g(a, g(u, b)))), \\ f(g(a, g(a, b)), f(a, g(a, g(a, b)))), \\ f(g(a, g(b, b)), f(a, g(a, g(b, b)))), \\ \dots \end{array}$$

and $\mathcal{V}ar(\tau) = \{x, y, u\}$.

Constraint terms allow giving a very simple and concise definition of a relation similar to basic narrowing and called constraint narrowing:

Definition 14.4 (Constraint narrowing)

A constraint term $(\exists W, t[u]_p \parallel c)$ *c-narrows* (narrows with constraints) into the constraint term $(\exists W \cup \mathcal{V}ar(l), t[r]_p \parallel c \wedge u =^?_{\emptyset} l)$ at the non-variable position $p \in \mathcal{G}rd(t)$, using the rewrite rule $l \to r$ of the rewrite system R, if the system $c \wedge u =^?_{\emptyset} l$ is satisfiable and provided that the variables of the rule and the constraint terms are disjoint: $\mathcal{V}ar(l, r) \cap \mathcal{V}ar((t \parallel c)) = \emptyset$. This is denoted:

$$(\exists W, t[u]_p \parallel c) \overset{c}{\rightsquigarrow}{}^{R}_{[p,l\to r]} (\exists W \cup \mathcal{V}ar(l), t[r]_p \parallel c \wedge u =^?_{\emptyset} l).$$

Example 14.3 If we consider as previously the rule $f(f(x)) \to x$, then the term $(f(y) \parallel \mathbf{T})$ c-narrows at position Λ:

$$(\exists, f(y) \parallel \mathbf{T}) \overset{c}{\rightsquigarrow}_{[\Lambda, f(f(x))\to x]} (\exists\{x\}, x \parallel f(y) =^?_{\emptyset} f(f(x))),$$

and similarly:

$$(\exists, g(y, f(y)) \parallel \mathbf{T}) \overset{c}{\rightsquigarrow}_{[2, f(f(x))\to x, \{(x\mapsto z),(y\mapsto f(z))\}]} (\exists\{x\}, g(y, x) \parallel f(y) =^? f(f(x))).$$

As one can infer from the previous definitions and examples, there is a close relationship between basic narrowing derivations and constraint narrowing ones. In fact, it follows directly from the definition of the set of basic positions that the terms in a constraint narrowing derivation are a representation of the set of basic positions:

Lemma 14.1 Let R be a term rewriting system and for all i, let t_i, s_i be terms such that furthermore $s_0 = t_0$. If:

$$(t_0, \mathcal{G}rd(t_0)) \overset{b}{\rightsquigarrow}^{R}_{[p_1,l_1\to r_1]} (t_1, \mathcal{O}_1) \overset{b}{\rightsquigarrow}^{R}_{[p_2,l_2\to r_2]} (t_2, \mathcal{O}_2) \overset{b}{\rightsquigarrow}^{R}_{[p_3,l_3\to r_3]} \cdots,$$

and:

$$(\exists W_0, s_0 \parallel \mathbf{T}) \overset{c}{\rightsquigarrow}^{R}_{[p_1,l_1\to r_1]} (\exists W_1, s_1 \parallel c_1) \overset{c}{\rightsquigarrow}^{R}_{[p_2,l_2\to r_2]} (\exists W_2, s_2 \parallel c_2) \overset{c}{\rightsquigarrow}^{R}_{[p_3,l_3\to r_3]} \cdots,$$

then $s_{i|\mathcal{O}_i} = t_{i|\mathcal{O}_i}$, i.e. the terms s_i and t_i are equal up to all positions in $\mathcal{O}_i$, i.e. $\forall p \in \mathcal{O}_i, s_i(p) = t_i(p)$.

Proof: By a straightforward application of the definitions of basic and constraint narrowing. $\Box$

14.2.2 Narrowing versus rewriting

The relation between rewriting and narrowing can be made more precise than just the trivial relationship $\longrightarrow^R \subseteq \rightsquigarrow^R$.

Lemma 14.2 For any term t and term rewriting system R, if $t \rightsquigarrow^{R}_{[m,g\to d,\sigma]} t'$ then $\sigma(t) \longrightarrow^{R}_{[m,g\to d]} t'$.

This can be pictured as follows:

$$\begin{tikzcd}
 & \sigma(t) \arrow[dr, "{[m,g\to d]}"] & \\
t \arrow[ur, mapsto, "\sigma"] \arrow[rr, Mapsto, "{[m,g\to d,\sigma]}"] & & \sigma(t[d]_m)
\end{tikzcd}$$

The dual of this property, i.e. the rewriting to narrowing correspondance schema is more subtle and has been exhibited first by J.-M. Hullot [Hul80a, Hul80c].

Proposition 14.1 Let t_0 be a term and ρ be a R-normalized substitution such that $\rho(t_0) \longrightarrow^{R}_{[m,g\to d]} t'_1$. Then there exist substitutions σ et μ such that:

1. $t_0 \rightsquigarrow^{R}_{[m,g\to d,\sigma]} t_1$,
2. $\mu(t_1) = t'_1$,
3. $\rho =^{\mathcal{V}ar(t_0)} \mu\sigma$,
4. μ is R-normalized.

$$\begin{tikzcd}
\rho(t_0) \arrow[r, "g\to d"] & t'_1 \\
t_0 \arrow[u, mapsto, "\rho"] \arrow[r, Mapsto, "{[m,g\to d,\sigma]}"] & t_1 \arrow[u, mapsto, "\mu"']
\end{tikzcd}$$

Proof: Let us first prove that t_0 is R-narrowable. Since $\rho(t_0)$ is R-reducible at position m using rule $g \to d$ (which is always supposed such that $\mathcal{V}ar(g) \cap \mathcal{V}ar(\rho(t_0)) = \emptyset$), there exists a substitution γ such that:

$$\gamma g = \rho(t_0)|_m \quad \text{with} \quad \begin{array}{l} \mathcal{V}ar(g) \cap \mathcal{D}om(\rho) = \emptyset, \\ \mathcal{V}ar(g) \cap \mathcal{V}ar(\rho(t_0)) = \emptyset \text{ and} \\ \mathcal{D}om(\gamma) \subseteq \mathcal{V}ar(g). \end{array}$$

By the previous (non restrictive) hypothesis, $\rho(g) = g$, $\mathcal{D}om(\gamma) \cap \mathcal{V}ar(\rho(t_0)) = \emptyset$, and we have:

$$\gamma\rho g = \gamma\rho(t_0)|_m,$$

which shows that $\gamma\rho$ is an unifier of g and $t_0|_m$. Since γ and ρ have been chosen with disjoint domains (which is always possible), we have also $\gamma\rho = \gamma + \rho$.

Since we need to preserve the set of variables under consideration, let us call them $W = \mathcal{V}ar(g) \cup \mathcal{V}ar(t_0)$. There exists a most general unifier σ of g and $t_0|_m$ outside W, and a substitution δ such that:

$$\delta.\sigma =^{\mathcal{V}ar(g) \cup \mathcal{V}ar(t_0|_m)} \gamma + \rho, \tag{14.1}$$

which proves that t_0 is R-narrowable using the rule $g \to d$ at position m.
In order to extend the relation (14.1) to the set of variables $\mathcal{V}ar(g) \cup \mathcal{V}ar(t_0)$, as needed later on, let us define:

$$\delta(x) = \rho(x) \text{ if } x \in \mathcal{X} - (\mathcal{V}ar(g) \cup \mathcal{V}ar(t_0|_m)).$$

This is consistent with (14.1) since σ has been chosen outside W and thus $\mathcal{R}an(\sigma) \cap W = \emptyset$. With this extension of δ, we now get:

$$\delta.\sigma =^{\mathcal{X}} \gamma + \rho.$$

As a restriction of it, and thanks to the definition of γ:

$$\delta.\sigma =^{\mathcal{V}ar(\rho(t_0))} \rho. \tag{14.2}$$

We can now prove the last statements of the theorem.
By defining μ to be δ, the relation (14.2) proves statement (3). Let us now check statement (2) i.e. $\mu(t_1) = t'_1$.

$$\begin{array}{rcll} \mu(t_1) & = & \mu.(\sigma(t_0)[m \leftarrow d]) & \text{by definition of } t_1, \\ & = & (\mu.\sigma)t_0[m \leftarrow \mu\sigma(d)] & \\ & = & \rho(t_0)[m \leftarrow \rho(d)] & \text{because of (14.2)}, \\ & = & t'_1 & \text{by definition of } t'_1. \end{array}$$

□

This result can be easily extended by induction on the number of steps to any rewriting derivation:

Corollary 14.1 Let t_0 be a term and ρ be a R-normalized substitution such that:

$$\rho(t_0) \longrightarrow^{R}_{[m_1, g_1 \to d_1]} \cdots \longrightarrow^{R}_{[m_n, g_n \to d_n]} t_n.$$

Then there exist substitutions $\sigma_i (i = 1, \ldots, n)$ et μ such that:

1. $t_0 \rightsquigarrow^{R}_{[m_1, g_1 \to d_1, \sigma_1]} t_1 \ldots \rightsquigarrow^{R}_{[m_n, g_n \to d_n, \sigma_n]} t_n$,
2. $\mu(t_n) = t'_n$,
3. $\rho =^{\mathcal{V}ar(t_0)} \mu\sigma_n \ldots \sigma_1$.

14.2.3 Narrowing for unification

We now use narrowing as a process to compute complete set of R-unifiers. This process is always correct and we study in which cases it is complete.

In order to simplify the description of the process, we introduce, for any rewrite system R on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, a new rule $x =^?_R x \to \mathbf{T}$ in which the symbol $=^?_R$ is considered as a new symbol of the signature (i.e. $=^?_R \;\notin \mathcal{F}$). Such a rule is applicable on an equation $s =^?_R t$ if and only if s and t are identical. A narrowing step using this rule on an equation $s =^?_R t$ is possible iff s and t are syntacticaly unifiable. This leads to an easy characterisation of R-unifiers:

Lemma 14.3 Let σ be a substitution, s and t be terms and R be any confluent rewrite system. Let $\bar{R} = R \cup \{x =^?_R x \to \mathbf{T}\}$. σ is a R-unifier of s and t if and only if $\sigma(s =^?_R t) \overset{*}{\longrightarrow}^{\bar{R}} \mathbf{T}$.

Proof: If σ is a R-unifier of s and t then $\sigma(s) =_R \sigma(t)$ and since R is confluent:

$$\exists u \in \mathcal{T}(\mathcal{F}, \mathcal{X}),\ \sigma(s) \overset{*}{\longrightarrow}^{R} u \;{}_{R}\!\overset{*}{\longleftarrow} \sigma(t)$$

and thus:

$$\sigma(s =^?_R t) = (\sigma(s) =^?_R \sigma(t)) \overset{*}{\longrightarrow}^{R} (u =^?_R u) \longrightarrow^{\bar{R}}_{x =^?_R x \to \mathbf{T}} \mathbf{T}.$$

Conversely, by definition of $\bar{R}$, the derivation should be of the following form:

$$\sigma(s =^?_R t) = (\sigma(s) =^?_R \sigma(t)) \overset{*}{\longrightarrow}^{\bar{R}} (u =^?_R v) \longrightarrow^{\bar{R}}_{x =^?_R x \to \mathbf{T}} \mathbf{T},$$

which means that u and v are syntactically equal terms and thus $\sigma(s) =_R u = v =_R \sigma(t)$ which shows that σ is a R-unifier of u and v. Notice that in this case the confluence hypothesis is useless. □

We can now prove the correctness of narrowing with respect to the computation of R-unifiers. Note that no hypothesis on the rewrite system R is required:

Theorem 14.1 *(Correctness)*
Let R be a term rewriting system and $\bar{R} = R \cup \{x =^?_R x \to \mathbf{T}\}$. For any equation $s =^?_R t$, if:

$$s =^?_R t \rightsquigarrow^{\bar{R}}_{[\sigma_1]} s_1 =^?_R t_1 \rightsquigarrow^{\bar{R}}_{[\sigma_2]} \dots \rightsquigarrow^{\bar{R}}_{[\sigma_n]} \mathbf{T},$$

then $\sigma_n \dots \sigma_1$ is a R-unifier of $s =^?_R t$.

Proof: By Lemma 14.2,

$$s =^?_R t \rightsquigarrow^{\bar{R}}_{[\sigma_1]} s_1 =^?_R t_1 \rightsquigarrow^{\bar{R}}_{[\sigma_2]} \dots \rightsquigarrow^{\bar{R}}_{[\sigma_n]} \mathbf{T}$$

implies that $\sigma_n \dots \sigma_1(s =^?_R t) \overset{+}{\longrightarrow}{}^{\bar{R}} \mathbf{T}$. By Lemma 14.3 (as said, the confluence hypothesis is not necessary for this direction), $\sigma_n \dots \sigma_1$ is a R-unifier of s and t. □

The completeness result is of course more complicated to prove and needs (at least in a first stage) to assume the term rewrite system R both confluent and terminating:

Theorem 14.2 *(Completeness)*
Let R be a terminating and confluent term rewriting system and $\bar{R} = R \cup \{x =^? x \to \mathbf{T}\}$. If the substitution σ is a R-unifier of the terms s and t, then there exists a narrowing derivation:

$$s =^?_R t \rightsquigarrow^{\bar{R}}_{[\sigma_1]} \dots \rightsquigarrow^{\bar{R}}_{[\sigma_n]} \mathbf{T}$$

such that:

$$\sigma_n \dots \sigma_1 \leq^{\mathcal{V}ar(s,t)}_R \sigma.$$

Proof: see [MH92]. It works in the same way as the completeness proof of constrained narrowing given later in this chapter. □

In this last result, the subscript R can be dropped out in $\sigma_n \dots \sigma_1 \leq^{\mathcal{V}ar(s,t)}_R \sigma$ when considering only normalized substitutions. This allows then to conclude:

Corollary 14.2 Narrowing is complete for confluent term rewriting systems with respect to normalizable substitutions.

Example 14.4 Consider the rewrite system $R = \{x + 0 \to x, x + s(y) = s(x + y)\}$. Then let us narrow the equation $x + y =^?_R s(0)$:

$$x + y =^?_R s(0) \rightsquigarrow^{R}_{\Lambda, x'+0 \to x', \{x' \mapsto x, y \mapsto 0\}} x =^?_R s(0) \rightsquigarrow^{\bar{R}}_{\Lambda, x =^?_R x \to \mathbf{T}, \{x \mapsto s(0)\}} \mathbf{T},$$

and thus $\{x \mapsto s(0), y \mapsto 0\}$ is proved to be a R-unifiers.

Note that the rewrite rule $x + s(y) = s(x + y)$ can always narrow the term that it has just narrowed and thus the narrowing process will not end on this quite simple example, but it discovers of course the two solutions of the equation.

Exercice 49 — Use the system BasicArithmetic on page 118 to solve the equation $x * x =^? x + x$.

14.2.4 Constraint narrowing for unification

Similarly to the previous section, we now consider the problem of generating complete sets of R-unifiers using the relation of constraint narrowing.

Since we consider constraint equation systems, let us first defined what a solution of such an entity is. Note that we give the definition in the particular case where the constraint consists in a system of equations in the empty theory. For a more general definition, see [KR93].

Definition 14.5 Let R be a term rewriting system on $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A *constrained system* is a constraint term $(\exists W, P \parallel c)$ where $P = \bigwedge_{i=1,\dots,n} s_i =^?_R t_i$ is a system of R-equations, i.e. a term in $\mathcal{T}(\mathcal{F} \cup \{\wedge, =^?_R\}, \mathcal{X})$. In general c can be any kind of constraint but, in the restricted case that we are dealing with here, it is a system of equations in the empty theory: $c = \bigwedge_{i=1,\dots,n} s_i =^?_\emptyset t_i$.

A *R-unifier* of a constrained system $(\exists W, P \parallel c)$ is a substitution σ which is $\emptyset$-unifier of $(\exists W, c)$ and R-unifier of $(\exists W, P)$.

For example $(x =^?_R y \parallel \mathbf{F})$ has no R-unifier. The R-solutions of $(s =^?_R t \parallel \mathbf{T})$ are the same as the R-unifiers of $s =^?_R t$.

Proposition 14.2 (Correctness)
Let R be a term rewriting system. If:

$$G = (\exists W, s =^?_R t \parallel c) \overset{c}{\rightsquigarrow}^R_{[1.p,l\to r]} G' = (\exists W \cup \mathcal{V}ar(l), s[r]_p =^?_R t \parallel c \wedge s|_p =^?_\emptyset l),$$

then:

$$\mathcal{U}_R(G') \subseteq \mathcal{U}_R(G).$$

Proof: Let σ be a R-unifier of G'. Then by definition, σ is an $\emptyset$-unifier of $(\exists W \cup \mathcal{V}ar(l), c \wedge s|_p =^?_\emptyset l)$ and thus of $(\exists W, c)$.
Now, the fact that σ is a R-unifier of $(\exists W, s =^?_R t)$ comes from:

$$\sigma(t) =_R \sigma(s[r]_p) = \sigma(s)[\sigma(r)]_p =_R \sigma(s)[\sigma(l)]_p = \sigma(s[l]_p) = \sigma(s).$$

□

The completeness of the narrowing process with respect to R-unification is derived from the next results that are quite similar to those of Section 14.2.3. The main and quite important difference relies on the way the proofs are conducted. Results and proofs are then generalisable to more complicated constraint systems like the one induced by associative-commutative equality. This is not at all the case of the other proof methods, that several attempts to generalize have shown to be quite technical.

Proposition 14.3 Let R be a terminating term rewriting system, $(\exists W, t \parallel c)$ be a constraint term and ρ be a R-normalized substitution such that $\rho(t)$ is R-reducible and ρ is $\emptyset$-unifier of c. Then there exist:

- a rule $l \to r \in R$ and an occurence $m \in \mathcal{G}rd(t)$,
- a substitution μ,

such that:

1. $(\exists W, t \parallel c) \overset{c}{\rightsquigarrow}^R_{[m,l\to r]} (\exists W \cup \mathcal{V}ar(l), t[r]_m \parallel c \wedge l =^?_\emptyset t|_m)$,
2. $\mu(t[r]_m) = t'$,
3. μ is R-normalized,
4. μ is an $\emptyset$-unifier of $(\exists W \cup \mathcal{V}ar(l), c \wedge l =^?_\emptyset t|_m)$,
5. $\rho =^{\mathcal{V}ar(t)\cup\mathcal{V}ar(c)} \mu$.

Proof: Since $\rho(t)$ is reducible, let us take one of its innermost redex determined by a rule $l \to r$ is R and an occurence $m \in \mathcal{G}rd(t)$. So we have $\rho(t) \longrightarrow^R_{[m,l\to r]} t'$, and there exists a substitution γ such that $(\rho(t))_{|m} = \gamma(l)$ and $t' = \rho(t)[\gamma(r)]_m$. Since ρ is normalized, we necessarily have $(\rho(t))_{|m} = \rho(t_{|m})$. And since $\rho(t_{|m}) = \gamma(l)$, $t_{|m}$ and l are $\emptyset$-unifiable. Notice that γ is necessarily R-normalized since m is an innermost position of a redex in $\rho(t)$.

Let us define the substitution μ on $\mathcal{V}ar(t) \cup \mathcal{V}ar(c) \cup \mathcal{V}ar(l)$ by:

- $\mu =^{\mathcal{V}ar(t)\cup\mathcal{V}ar(c)} \rho$ and,
- $\mu =^{\mathcal{V}ar(l)} \gamma$.

Since $\mu =^{\mathcal{V}ar(t)\cup\mathcal{V}ar(c)} \rho$, μ is $\emptyset$-unifier of c. By definition, μ is also $\emptyset$-unifier of $t_{|m} =^?_\emptyset l$ and thus this shows that μ is an $\emptyset$-unifier of $c \wedge l =^?_\emptyset t|_m$.

Now we simply have: $\mu(t[r]_m) = \mu(t)[\mu(r)]_m = \rho(t)[\gamma(r)]_m = t'$.

Last but not least, μ is R-normalized since ρ and γ are so. □

This result extends easily, by induction based on the rewrite relation, to a derivation, yielding the following result:

Corollary 14.3 Let R be a terminating term rewriting system, $(\exists W_0, t_0 \parallel c_0)$ be a constrained term and ρ be a R-normalized substitution such that $\rho(t_0)$ is R-reducible and ρ is a $\emptyset$-unifier of c. Then, there exists an (innermost) rewriting derivation:

$$\rho(t_0) \longrightarrow^{R}_{[m_1,g_1\to d_1]} t'_1 \ldots \longrightarrow^{R}_{[m_n,g_n\to d_n]} t'_n,$$

with t_n irreducible, a constraint narrowing derivation:

$$(\exists W_0, t_0 \parallel c_0) \overset{c}{\leadsto}^{R}_{[m_1,g_1\to d_1]} (\exists W_1, t_1 \parallel c_1) \ldots \overset{c}{\leadsto}^{R}_{[m_n,g_n\to d_n]} (\exists W_n, t_n \parallel c_n)$$

and substitutions μ_i such that for all $i = 1..n$:

1. $\mu_0 = \rho =^{\mathcal{V}ar(t_0)\cup\mathcal{V}ar(c_0)} \mu_i$,
2. $\mu_i(t_i) = t'_i$,
3. μ_i is R-normalized,
4. μ_i is $\emptyset$-solution of $(\exists W_i, c_i)$.

We are now ready to apply these results to get completeness constraint narrowing with respect to R-unification.

Theorem 14.3 *Let R be a terminating and confluent term rewriting system and $\bar{R} = R \cup \{x =^? x \to \mathbf{T}\}$. If the substitution σ is a R-unifier of the terms s and t, then there exists a constraint narrowing derivation:*

$$(\exists\emptyset, s =^?_R t \parallel \mathbf{T}) \overset{c}{\leadsto}^{\bar{R}} \ldots \overset{c}{\leadsto}^{\bar{R}} (\exists W_n, \mathbf{T} \parallel c_n),$$

such that $\sigma \in \mathcal{U}_\emptyset(c_n)$.

Proof: Since σ is a R-unifier of s and t, we have by confluence and termination of $\bar{R}$:

$$\sigma(s =^?_R t) = \sigma(s) =^?_R \sigma(t) \xrightarrow{*}^{R} u =^?_R u \longrightarrow^{\bar{R}} \mathbf{T},$$

and thus $\sigma(s =^?_R t)$ is R-reducible. We can apply Corollary 14.3 which leads to conclude that there exists a constraint narrowing derivation leading to $\mathbf{T}$:

$$(\exists\emptyset, s =^?_R t \parallel \mathbf{T}) \overset{c}{\leadsto}^{R}_{[m_1,g_1\to d_1]} (\exists W_1, s_1 =^?_R t_1 \parallel c_1) \ldots \overset{c}{\leadsto}^{R}_{[m_n,g_n\to d_n]} (\exists W_n, \mathbf{T} \parallel c_n)$$

since the last reduction (and thus also narrowing) should be applied with the rule $x =^?_R x \to \mathbf{T}$. Furthermore, taking the notations of Corollary 14.3, $\mu_n =^{\mathcal{V}ar(s=^?_R t)} \sigma$ and μ is an $\emptyset$-unifier of $(\exists W_n, c_n)$, This yields the conclusion that $\sigma \in \mathcal{U}_\emptyset(c_n)$. □

Let us consider now the *constraint narrowing tree* whose root is labelled with the constrained term $(\exists\emptyset, s =^?_R t \parallel \mathbf{T})$ and whose edges are all possible constraint narrowing derivations issued from a given node. In this tree, which is in general infinite, a *successful leave* is by definition a node labelled by a constrained term of the form: $(\exists W, \mathbf{T} \parallel c)$. For a given equation $s =^?_R t$, we denote $\mathcal{SNT}(s =^?_R t)$ the set of all successful nodes of the constraint narrowing tree issued from $(\exists\emptyset, s =^?_R t \parallel \mathbf{T})$.

Thanks to Theorem 14.3, we have:

$$\mathcal{U}_R(s =^?_R t) \subseteq \bigcup_{(\exists W, \mathbf{T} \parallel c)\in\mathcal{SNT}(s=^?_R t)} \mathcal{U}_\emptyset(c),$$

and since constraint narrowing is correct (by Lemma 14.2) we get the equality:

$$\mathcal{U}_R(s =^?_R t) = \bigcup_{(\exists W, \mathbf{T} \parallel c)\in\mathcal{SNT}(s=^?_R t)} \mathcal{U}_\emptyset(c).$$

This justifies the following main result about constraint narrowing:

Corollary 14.4 The transformation rules described in Figure 14.1, applied in a non deterministic and fair way to the constraint equation $(\exists\emptyset, s =^?_R t \parallel \mathbf{T})$, yield constraint equations of the form $(\exists W, \mathbf{T} \parallel c)$ such that the mgus of the c's form altogether a complete set of R-unifiers of $s =^?_R t$.

Narrow $(\exists W, s =^?_R t \parallel c)$
$\mapsto\!\!\!\rightarrow$
$(\exists W \cup \mathcal{V}ar(l), s[r]_p =^?_R t \parallel c \wedge s|_p =^?_\emptyset l)$
if $(c \wedge (s_{|p} =^?_\emptyset l))$ is satisfiable

Block $(\exists W, s =^?_R t \parallel c)$
$\mapsto\!\!\!\rightarrow$
$(\exists W, \mathbf{T} \parallel c \wedge s =^?_\emptyset t)$
if $(c \wedge (s =^?_\emptyset t))$ is satisfiable

Figure 14.1: **Narrowing**: Unification via constrained narrowing

Note that in the set of rules **Narrowing**, the **Block** rule mimics exactly the application of the rule $x =^?_R x \to \mathbf{T}$.

Example 14.5 If we consider the rewrite system R reduced to the only following rule: $f(f(y)) \to y$, then the constraint equation $(\exists\emptyset, f(x) =^?_R x \parallel \mathbf{T})$ is rewritten, using the rules**Narrowing** as follows:

$$\begin{array}{ll}
 & (\exists\emptyset, f(x) =^?_R x \parallel \mathbf{T}) \\
\mapsto\!\!\!\rightarrow_{\mathbf{Narrow}} & (\exists\{y\}, y =^?_R x \parallel f(f(y)) =^?_\emptyset f(x)) \\
\mapsto\!\!\!\rightarrow_{\mathbf{Block}} & (\exists\{y\}, \mathbf{T} \parallel y =^?_\emptyset x \wedge f(f(y)) =^?_\emptyset f(x)) \\
\mapsto\!\!\!\rightarrow_{\mathbf{SyntacticUnification}} & (\exists\{y\}, \mathbf{T} \parallel \mathbf{F})
\end{array}$$

and thus the equation $f(x) =^?_R x$ has no R-unifier.

Exercice 50 — Use the system BasicArithmetic on page 118 to solve the equation $x * x =^? x + x$ using *constraint* narrowing.

Notice that the previous proof of completeness of constraint narrowing can be extended to equational constraints. This allows dealing in particular with narrowing modulo associativity and commutativity.
Further Readings:

In detecting redundant narrowing derivations by the LSE-SL reducibility test [KB91, BKW93b], Krisher, Bockmayr and Werner show that without further restriction on the canonical term rewriting system, a complete strategy with better efficiency is achieved through reducibility tests that allow to detect redundancies in the derivations. A study of narrowing strategies can be found in [BKW93a].

In the same context, another proposition is done by J.Chabin and P.Réty in Narrowing directed by a graph of terms *[CR91]. Their idea is to generalize the graph of top symbols used by Sivakumar and Dershowitz [DS88], by a graph of terms built from the rewrite system and the equation to solve.*

Combining narrowing and ordered completion is proposed in [Cha94] in order to design a goal oriented approach.

Narrowing has been extended to narrowing in conditional equational theories [Hus85, Boc93].

Narrowing is also complete for weakly terminating rewrite systems as shown by A. Werner [Wer94].

14.2.5 Applications

Many approaches on the amalgamation of logic programming and functional programming are narrowing-based [GM86, RKKL85, BL86, DG89]. For a survey, see [Han94]. Narrowing subsumes SLD-resolution and constitutes a complete inference system if function definitions, considered as a rewrite system, are confluent and terminating. In the framework of theories with constructors, refined strategies of narrowing can be designed, such as innermost narrowing [Der83a, Fri85b, BGM87], basic innermost narrowing [Höl88, Han90], and lazy narrowing [Red85, LPB$^+$87, You89]. These restrictions have the following completeness results and implementations:

- Innermost narrowing is complete in the case of total functions, provided the underlying theory has a convergent rewrite system. An implementation of innermost narrowing coupled with rewriting is provided in SLOG [Fri85b].

- Basic innermost narrowing is similar to innermost narrowing except that axioms of the form $f(x) = f(y)$ are used for every incompletely defined function f. Basic innermost narrowing is complete provided the underlying theory has a convergent rewrite system [Höl88]. An implementation of basic innermost narrowing coupled with rewriting is provided in the language ALF [Han90].

- Lazy narrowing is complete provided the underlying rewrite system is left-linear and non-ambiguous. The languages K-LEAF [LPB$^+$87] and BABEL [KLMR90] use two implementations of lazy narrowing.

From other points of view, narrowing can also be applied to solving equational disunification [Fer92] and more generaly to the solving of constraint in combined algebraic domains [KR94b].

Many implementation of narrowing have been designed. Let us mention in particular [JD86, Lin89, OS88].

Part IV

Proving

Chapter 15

Proof reduction

15.1 Introduction

This chapter introduces the main concepts underlying all completion processes presented in this book. The goal is to reach a uniform point of view for all of them, which relies on standard notions of formulas, transition rules, axioms and proofs.

In a first approximation, completion procedures transform sets of formulas, by applying transition rules on them. But this view is too simple and must be refined to take into account the great variety of completion processes. This goal is not so easy to achieve, because of apparently opposed notions that can be pointed out:

- The different purposes of completion procedures: some of them transform the presentation of a theory. They are aimed for instance at producing a confluent and terminating rewrite systems, or more generally at saturating a set of formulas. Others are goal-directed processes for the proof of a given formula and try to derive a contradiction or an inconsistency.

- Transition rules are of different kinds: some of them are expansion rules and increase the search space by creating new formulas to be proved, others are contraction rules and make the search space smaller, by eliminating redundant formulas. The last ones are crucial for efficiency but their use must be carefully controlled in order to keep the property of completeness of the whole set of transition rules with respect to the intended kind of proofs.

Section 15.2 introduces the notions of proofs and their transformation. The important notion is the proof ordering that supports the view of completion procedures as proof reduction processes. Section 15.3 defines in general the concept of completion process and points out its relation with transformation of proofs. Distinction is made between completion procedures for generating saturated presentations and completion procedures for theorem proving by refutation. Common concepts of correctness, completeness and fairness are defined.

15.2 Proof transformation

Given a finite set of sentences S called a presentation, the set of formulas that hold in S is called the theory of S and denoted by $Th(S)$.

Assume now that the presentation is composed of a set of rules R and a set of equalities P. So $S = (R, P)$ and we first focus on equational proofs.

Definition 15.1 A *proof* in $S = (R, P)$ of an equality $t = t'$ is a sequence of terms $\{t_0, ..., t_n\}$ such that $t_0 = t, t_n = t'$ and for $0 < i \leq n$, one of $t_{i-1} \longleftrightarrow_P t_i$, $t_{i-1} \rightarrow_R t_i$ or $t_{i-1} \leftarrow_R t_i$ holds. Each step is justified by an axiom $l = r \in P \cup R$, a position ω_i in t_i and a substitution σ such that $t_{i|\omega} = \sigma(l)$ and $t_{i+1} = t_i[\omega \leftarrow \sigma(r)]$.

Example 15.1 Consider the rewrite system R:

$$\begin{array}{rcl} f(x) & \rightarrow & a \\ c & \rightarrow & b. \end{array}$$

and the set of equalities P:

$$a = b.$$

A proof of $f(a) = c$ is for instance

$$f(a) \to_R a \longleftrightarrow_P b \leftarrow_R c$$

where axioms, position and substitutions are easy to find.

Letters $\mathcal{P}, \mathcal{Q}$ will be used to denote proofs and Λ will denote the empty proof, that is the sequence of terms reduced to one term t. If $\mathcal{P}$ is the proof $\{t_0, ..., t_n\}$, $\mathcal{P}^{-1}$ then denotes the converse proof $\{t_n, ..., t_0\}$, $\sigma(\mathcal{P})$ the proof $\{\sigma(t_0), ..., \sigma(t_n)\}$ and $t[\mathcal{P}]$ the proof $\{t[t_0], ..., t[t_n]\}$. A subproof of $\mathcal{P}$ is any proof $\{t_i, ..., t_j\}$ with $0 \leq i \leq j \leq n$. We write $\mathcal{P}[\mathcal{Q}]$ to denote that $\mathcal{P}$ contains $\mathcal{Q}$ as subproof.

So equational proofs may be defined as terms built on an extended alphabet in which $\leftarrow$, $\rightarrow$, $\longleftrightarrow$ are function symbols. A precise formalisation of this point of view can be found in [Dev91].

In general we are interested in specific kinds of proofs, for instance in rewrite proofs, of the form $s \xrightarrow{*}_R u \xleftarrow{*}_R t$, or in characteristic inconsistencies such as $true \overset{*}{\longleftrightarrow} false$. Such proofs will be characterized as minimal proofs according to the notion of proof orderings described below. A proof ordering defines a relation on proofs that is closed under substitutions and under contexts. A context may be a term context but also a proof context if we consider $\longleftrightarrow$, $\leftarrow$ and $\rightarrow$ as constructor operators for proofs.

Definition 15.2 [Bac87, BD94] A *proof ordering* is a well-founded binary relation $\Longrightarrow$ on the set of proofs which satisfies

- for all proofs $\mathcal{Q}, \mathcal{Q}', \mathcal{P}$,
$$\mathcal{Q} \Longrightarrow \mathcal{Q}' \textit{ implies } \mathcal{P}[\mathcal{Q}] \Longrightarrow \mathcal{P}[\mathcal{Q}'].$$
- for all proofs $\mathcal{Q}, \mathcal{Q}'$ and any term t,
$$\mathcal{Q} \Longrightarrow \mathcal{Q}' \textit{ implies } t[\mathcal{Q}] \Longrightarrow t[\mathcal{Q}'].$$
- for all proofs $\mathcal{Q}, \mathcal{Q}'$ and any substitution σ,
$$\mathcal{Q} \Longrightarrow \mathcal{Q}' \textit{ implies } \sigma(\mathcal{Q}) \Longrightarrow \sigma(\mathcal{Q}').$$

Proofs orderings are built by formalizing a rewrite relation applied on proofs (considered as terms) and characterized by *proof patterns* (considered as rewrite rules). To avoid confusion, we will call this relation *proof transformation*.

Definition 15.3 A *proof pattern* is a schema describing a class of subproofs.

A *rewrite proof* is characterized by the proof pattern $s \xrightarrow{*}_R u \xleftarrow{*}_R t$ where s, t, u denote arbitrary terms.

A *peak* is characterized by the proof pattern $t'' \longleftarrow_R t \longrightarrow_R t'$.

A *proof transformation rule* is a pair of proof patterns $\mathcal{P} \Longrightarrow \mathcal{Q}$. A proof transformation system $\mathcal{R}$ is a set of proof transformation rules.

The *proof transformation relation* is the rewrite relation generated by $\mathcal{R}$, denoted $\Longrightarrow_{\mathcal{R}}$ (or $\Longrightarrow$ when $\mathcal{R}$ is clear from context).

Example 15.2 Exemples of proof patterns that will be useful are

$$\begin{array}{rcl} t \longleftrightarrow_P^{p=q} t' & \Longrightarrow & t \to_R^{p\to q} t' \\ t' \leftarrow_R^{l\to r} t \to_R^{g\to d} t'' & \Longrightarrow & t' \longleftrightarrow_P^{p=q} t'' \end{array}$$

Consider again the rewrite system R:

$$\begin{array}{rcl} f(x) & \to & a \\ c & \to & b. \end{array}$$

and the set of equalities P:

$$a = b.$$

The proof of $f(a) = c$ in $S = (R, P)$

$$f(a) \to_R a \longleftrightarrow_P b \leftarrow_R c$$

is transformed by $\Longrightarrow$ into the proof of $f(a) = c$ in $S' = (R', P')$

$$f(a) \rightarrow_{R'} a \rightarrow_{R'} b \leftarrow_{R'} c$$

where $P' = \emptyset$ and R' is the rewrite system

$$\begin{array}{rcl} f(x) & \rightarrow & a \\ c & \rightarrow & b \\ a & \rightarrow & b. \end{array}$$

The relation $\Longrightarrow$ is defined here on proofs of the presentation $S = (R \cup R', P \cup P')$.

A proof transformation relation $\Longrightarrow$ is a proof ordering as soon as it is well-founded. Given a presentation $S = (R, P)$, proving that the proof transformation relation $\Longrightarrow$ terminates implies that any proof has a normal form, which is minimal in the proof ordering. Termination will be proved by finding a complexity measure $c(\mathcal{P})$ for each proof $\mathcal{P}$ and a well-founded ordering $>_c$ on these measures, that satisfy the following property: $\mathcal{P} \Longrightarrow \mathcal{Q}$ implies $c(\mathcal{P}) >_c c(\mathcal{Q})$, for each proof transformation rule. Minimal proofs w.r.t. $\Longrightarrow$ will correspond to proofs whose complexity measure is minimal.

As an example of the introduced formalism, the Church-Rosser property can now be expressed in the following way: the rewriting relation $\rightarrow_R$ is Church-Rosser on a set $\mathcal{T}$ of elements $t, t', ...,$ if any proof $t \overset{*}{\longleftrightarrow}_R t'$ has a rewrite proof. The rewriting relation $\rightarrow_R$ is locally confluent if any peak $t'' \longleftarrow_R t \longrightarrow_R t'$ has a rewrite proof.

Example 15.3 Consider the rewrite system R:

$$\begin{array}{rcl} f(x) & \rightarrow & a \\ c & \rightarrow & b. \end{array}$$

and the set of equalities P:

$$a = b.$$

The proof of $f(b) = a$

$$f(b) \leftarrow_R f(c) \rightarrow_R a$$

can be transformed into a rewrite proof

$$f(b) \rightarrow_R a.$$

The notion of proof ordering will be most often used to compare two proofs of the same theorem in two different specifications. But the definition also holds for comparing two proofs of different theorems. These two uses correspond to different purposes of completion processes, explained in the next sections.

15.3 Completion procedures

Several completion procedures will be considered in the following chapters. They have different purposes but comon features can be outlined and this section is intended to propose a general definition of a completion procedure.

First a deduction process applied to a set of formulas of interest is characterized by transition rules managed by a specific control.

Definition 15.4 A *deduction process* in a theory T is defined by

- a set of transition rules I that transform presentations S of T and sets of goals (that is formulas of interest) G.

A transition rule is written as:

$$\begin{array}{llcl} \textbf{Name} & S; G & \mapsto\!\!\!\!\rightarrow & S; G' \\ & & & \text{if } \textit{Condition} \end{array}$$

- a control ST (also called search plan or deduction strategy) on transition rules.

The data given to a deduction process are pairs $(S; G)$ where S is a presentation of the theory T, and G is the set of formulas we are interested in. When either the presentation or the set of considered formulas remains constant, we may omit to mention it in the transition rules.

A derivation by a deduction process is a sequence

$$(S_0; G_0) \mapstochar\twoheadrightarrow (S_1; G_1) \mapstochar\twoheadrightarrow \ldots \mapstochar\twoheadrightarrow (S_i; G_i) \mapstochar\twoheadrightarrow (S_{i+1}; G_{i+1}) \ldots$$

where $(S_{i+1}; G_{i+1})$ is deduced from $(S_i; G_i)$ by applying a transition rule.

A proof reduction process can be associated to such a derivation. This point of view relies on the correspondance between the application of transition rules on presentations and goals and the proof transformation on a set of proofs Φ. The proof transformation relation *reflects* the transition rules if, at each transition step $\mapstochar\twoheadrightarrow$, a given proof either does not change or is transformed into another one by $\Longrightarrow$.

Definition 15.5 The proof transformation relation $\Longrightarrow$ on Φ *reflects* $\mapstochar\twoheadrightarrow$ if whenever $(S_i; G_i) \mapstochar\twoheadrightarrow (S_{i+1}; G_{i+1})$ and $\mathcal{P}$ is a minimal proof in Φ using $S_i \cup G_i$, then there is a proof $\mathcal{P}'$ using $S_{i+1} \cup G_{i+1}$, such that $\mathcal{P} \overset{*}{\Longrightarrow} \mathcal{P}'$.

Example 15.4 To illustrate how proofs of formulas are transformed concurrently with the transformation of presentations, let us consider a very simple example of a theory initially defined by two constant rewrite rules $\{a \to b, a \to c\}$ and the proofs of two different formulas $\phi_1 = (f(b) = f(c))$ and $\phi_2 = (g(a) = g(b))$. Presentations in the first column result from the application of some transition rules of the Knuth-Bendix completion (namely **Deduce**, **Orient**, **Simplify**). Each transition rule is reflected by a proof transformation that reduces the complexity of proofs, indicated below each proof.

Assume that $>$ is a reduction ordering on terms such that $a > b > c$. For the first proof, it is enough to choose as complexity measure of a rewrite step the first term: $c(f(a) \to f(c)) = \{f(a)\}$, $c(f(a) \to f(b)) = \{f(a)\}$ and $c(f(b) \leftarrow f(a) \to f(c)) = \{\{f(a)\}, \{f(a)\}\}$. The complexity of an equality step is given by the two terms: $c(f(b) \longleftrightarrow f(c)) = \{f(b), f(c)\}$. Then using the multiset extension of $>$, $\{\{f(a)\}, \{f(a)\}\} >_{mult} \{\{f(b), f(c)\}\}$.

But this complexity is not powerful enough in the case of the second proof to establish that $c(g(a) \to g(b))$ is greater than $c(g(a) \to g(c) \leftarrow g(b))$. So a second component is introduced which is here the rewrite rule itself. $c(g(a) \to g(b)) = \{(g(a), a \to b)\}$ and $c(g(a) \to g(c) \leftarrow g(b)) = \{\{(g(a), a \to c)\}, \{g(b), b \to c\}\}$. We need in addition to compare rewrite rule to state that $(a \to b)$ is greater than $(a \to c)$, which can be done by stating an ordering on rewrite rules defined lexicographically comparing the left- and right-hand sides with $>$: $(a, b) >_{lex} (a, c)$.

S	ϕ_1	ϕ_2
$a \to b$ $a \to c$	$f(b) \leftarrow f(a) \to f(c)$	$g(a) \to g(b)$
	$\{\{f(a)\}, \{f(a)\}\}$	
$a \to b$ $a \to c$ $b = c$	$f(b) \longleftrightarrow f(c)$	$g(a) \to g(b)$
	$\{f(b), f(c)\}$	
$a \to b$ $a \to c$ $b \to c$	$f(b) \to f(c)$	$g(a) \to g(b)$
	$\{f(b)\}$	$\{(g(a), a \to b)\}$
$a \to c$ $b \to c$	$f(b) \to f(c)$	$g(a) \to g(c) \leftarrow g(b)$
	$\{f(b)\}$	$\{\{(g(a), a \to c)\},$ $\{g(b), b \to c\}\}$

Informally, a completion procedure $\mathcal{C}$ is a deduction process that preserves validity of formulas and reduces proofs.

Definition 15.6 A *completion procedure* is a a deduction process $\mathcal{C} = (I, ST)$ such that, for any presentation S_0 of the theory T and any set of goal formulas G_0, the derivation

$$(S_0; G_0) \underset{C}{\mapstochar\twoheadrightarrow} (S_1; G_1) \underset{C}{\mapstochar\twoheadrightarrow} \ldots \underset{C}{\mapstochar\twoheadrightarrow} (S_i; G_i)$$

has the following properties:

- monotonicity: $\forall i \geq 0$, $Th(S_{i+1}) \subseteq Th(S_i)$

- relevance: $\forall i \geq 0$, for any $\phi \in \Phi$, $\phi \in Th(S_i)$ iff $\phi \in Th(S_{i+1})$
- reduction: The proof transformation relation $\Longrightarrow$ that reflects $\mapsto\!\!\!\rightarrow$ terminates.

Let us know consider three completion processes and see how they are expressed in this framework.

Example 15.5 The Knuth-Bendix completion procedure transforms an equational presentation S_0 of the theory T into a rewrite rule presentation with the Church-Rosser property. In this resultig presentation, a decision procedure for equational theorems is provided by rewriting. In that case, a derivation

$$S_0 \mapsto\!\!\!\rightarrow S_1 \mapsto\!\!\!\rightarrow \ldots \mapsto\!\!\!\rightarrow S_i \ldots$$

is reflected on the set Φ of all equational formulas, and the completion procedure is aimed at finding a rewrite proof for each equational theorem.

A similar case is unfailing (or ordered) completion for which close results hold but restricted to the set of ground terms.

Example 15.6 A different purpose of completion procedures is provided by the example of an unfailing completion procedure used as a refutational theorem prover: the theorem to be proved $(s = t)$ is first negated, then skolemized and added as a goal to the presentation of the theory. Then the whole set of formulas $(S_0; \{\overline{s} \neq \overline{t}\})$ is transformed by the transition rules in order to derive a set of formulas $(S_i; \{u \neq u\})$ that contains the contradiction $\{u \neq u\}$. In that case, the proof space Φ is the set of ground equational theorems plus formulas of the form $\{\overline{s} \neq \overline{t}\}$ which must be reduced to a normal form $\{u \neq u\}$.

Example 15.7 Another completion procedure, inductive completion, is aimed to prove an equational formula $(s = t)$ valid in the inductive theory of T, using a presentation of T which is a terminating and confluent presentation on ground terms. In order to perform a proof by consistency (or more appropriately by lack of inconsistency), the conjecture is added to the presentation R which is not modified during the process. Sets of formulas S_i are derived until no more transition rule is applicable, provided no inconsistency is detected during the process. Different notions of inconsistency provide different kinds of inductive completion processes. There the proof space is the set of equational theorems valid in the inductive theory of $R \cup \{s = t\}$ but not in the inductive theory of R.

Two classes of transition rules can be distinguished: expansion and contraction rules. Expansion rules strictly increases the sets of formulas in S_i, while contraction ones are aimed at discarding redundant formulas. Intuitively speaking, a formula is redundant if it can be deduced from other ones which are smaller, for some well-chosen ordering on formulas.

Definition 15.7 Let $\succ$ be a well-founded ordering on formulas. A formula F is *redundant* in S if there are formulas $F_1, F_2, \ldots F_n \in S$ such that $F_1, F_2, \ldots F_n \models F$ and $F \succ F_1, F_2, \ldots F_n$.

Using the proof transformation relation $\Longrightarrow$, the redundancy criterion may be formulated as follows [BD89b].

Definition 15.8 Let $\Longrightarrow$ be a well-founded proof transformation relation. An equality $s = t$ is *redundant* in E if $s \longleftrightarrow_{s=t} t \overset{*}{\Longrightarrow} s \overset{*}{\longleftrightarrow}_E t$.

Redundancy allows characterizing the notion of saturated presentation. A saturated set of formulas is a set to which no non-trivial consequence can be added.

Definition 15.9 A presentation S is *saturated* under the deductive transition rules $D \subseteq I$ if all formulas deduced from S by applying D are redundant in S.

Usually non-redundancy and saturation will be required for the resulting sets of formulas in a completion procedure. Let us consider the sets of all generated formulas $S_* = \bigcup_{i \geq 0} S_i$ and $G_* = \bigcup_{i \geq 0} G_i$. Let also S_∞ and G_∞ be respectively the sets of persisting formulas, i.e. the sets effectively generated by completion starting from $(S_0; G_0)$. Formally

$$S_\infty = \bigcup_{i \geq 0} \bigcap_{j > i} S_j \quad \text{and} \quad G_\infty = \bigcup_{i \geq 0} \bigcap_{j > i} G_j$$

Definition 15.10 A derivation is *fair* if all formulas deduced from $(S_\infty; G_\infty)$ by applying D are redundant in $(S_*; G_*)$.

The concept of fairness was already present in the formal proof of completion given in [Hue81], but it only recently emerges as a new research topic for completion-based provers, studied by itself and no more as an accessory hypothesis. M.P. Bonacina and J. Hsiang state in [BH91] a framework to define fairness of completion-based theorem proving strategies. They dissociate completeness of the transition rules and fairness of the strategy (called search plan) and illustrate the notion of fairness by various completion-based methods. A different approach is proposed by M. Hermann in [Her91] where fairness and correctness properties of transition rule-based completion strategies are expressed in the context of process logic. Additional notions of success, failure and justice of a completion strategy are necessary to support the proofs of the previous properties.

Chapter 16

Completion of rewrite systems

16.1 Introduction

The Church-Rosser property of a terminating term rewriting system is equivalent to the local confluence property. This property in turn can be checked on special patterns computed from pairs of rules and called *critical pairs*. When a set of rewrite rules fails this test because some critical pair is not joinable, a special rule tailored for this case is added to the system. This is the basic idea of the so-called *completion procedure*, designed by Knuth and Bendix [KB70], building on ideas of Evans [Eva51]. Of course, adding a new rule implies computing new critical pairs and the process is recursively applied. Whenever it stops, it comes up with a locally confluent term rewriting system and if, in addition, termination has been incrementally checked, the system is even confluent. Two other issues are possible: the process can find an equality that cannot be oriented and then terminates in failure. It can also indefinitely discover non joinable critical pairs and consequently endless add new rewrite rules.

The completion process was first studied from an operational point of view and several implementations are running [HKK89]. It became clear, from experimentations, that a completion process is composed of elementary tasks, such as rewriting a term in a rule or an equality, orienting an equality, computing a critical pair. Moreover the efficiency of the general process is related to the strategies chosen for combining these tasks. The next step was to consider each task as a transition rule that transforms sets of equalities and rules, strategies being expressed by a control for application of these transition rules. Separating control from transition rules made the proofs of correctness and completeness independent from strategies. As in traditional proof theory, these transition rules are studied by looking at their effect on equational proofs. The completion process then appears as a way to modify the axioms in order to reduce proofs in some normal form, called a *rewrite proof*. This most abstract point of view is adopted here. It is based on the relation between the transition rules system that describes the relation between successive pairs of sets R and P computed by the process on one hand, and a transformation proof relation that reduces proofs to rewrite proofs.

16.2 Critical pairs

Critical pairs are produced by overlaps of two redexes in a same term.

Definition 16.1 [Hue80] A non-variable term t' and a term t *overlap* if there exists a position ω in $\mathcal{G}rd(t)$ such that $t_{|\omega}$ and t' are unifiable.

Let $(g \to d)$ and $(l \to r)$ be two rules with disjoint sets of variables, such that l and g overlap at position ω of $\mathcal{G}rd(g)$ with the most general unifier ψ. The *overlapped term* $\psi(g)$ produces the *critical pair* (p, q) defined by $p = \psi(g[\omega \hookleftarrow r])$ and $q = \psi(d)$.

If ω is not a position in g or if $g_{|\omega}$ is a variable, then the rule $(l \to r)$ applies in the substitution part of $\psi(g)$ and the overlap is then called a *variable overlap*.

Note that a rule overlaps itself at the outermost position ϵ, producing a trivial critical pair.

Exercice 51 — Find all the critical pairs of the system

$$\begin{array}{rcl} (x * y) * z & \to & x * (y * z) \\ f(x) * f(y) & \to & f(x * y) \end{array}$$

Answer: Hint: Do not forget that a left-hand side can overlap itself.

Lemma 16.1 Critical pairs lemma [KB70, Hue80]
Let t, t', t'' be terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ such that

$$t' \leftarrow_R^{\omega,\alpha,l\to r} t \rightarrow_R^{\epsilon,\beta,g\to d} t''$$

with $\omega \in \mathcal{G}rd(g)$. Then, there exists a critical pair

$$(p,q) = (\psi(g[\omega \hookleftarrow r]), \psi(d))$$

of the rule $(l \to r)$ on the rule $(g \to d)$ at position ω such that $\psi = mgu(l, g_{|\omega})$ and $\beta\alpha \succeq_V \psi$ with $V = \mathcal{V}ar(g) \cup \mathcal{V}ar(l)$. Therefore there exists a substitution τ such that $t' = \tau(p)$ and $t'' = \tau(q)$.

Proof: Since $t = \beta(g)$ and $t_{|\omega} = \beta(g)_{|\omega} = \alpha(l)$, $g_{|\omega}$ and l are unifiable by the unifier $\sigma = \beta\alpha$. Let $\psi = mgu(l, g_{|\omega})$ and τ such that $\tau(\psi(x)) = \sigma(x), \forall x \in \mathcal{V}ar(g) \cup \mathcal{V}ar(l)$. By definition, there exists a critical pair $(p,q) = (\psi(g[\omega \hookleftarrow r]), \psi(d))$ and $t' = \tau(p)$ and $t'' = \tau(q)$. □

Notation: $CP(R)$ will denote the set of all critical pairs between rules in R.

Definition 16.2 A critical pair (p, q) is said *joinable* (or sometimes convergent), which is denoted by $u \downarrow_R v$, if there exists a rewrite proof

$$p \xrightarrow{*}_R w \xleftarrow{*}_R q.$$

The next theorem is also called Newman's lemma in the literature.

Theorem 16.1 *[KB70, Hue80] A rewrite system R is locally confluent iff any critical pair of R is joinable.*

Proof: The 'only if' part is obvious. For the 'if' part, consider a peak

$$t' \leftarrow_R^{\omega,\alpha,l\to r} t \rightarrow_R^{\upsilon,\beta,g\to d} t''$$

and the relative positions of the redexes $t_{|\omega}$ and $t_{|\upsilon}$.

Disjoint case: Then the two reductions commute:

$$t' \rightarrow_R^{\upsilon,\beta,g\to d} u \leftarrow_R^{\omega,\alpha,l\to r} t''.$$

Variable overlap case: We can assume without lost of generality that $\upsilon = \epsilon$. Thus $t = \beta(g)$ and there exists a variable x in g whose position in t is prefix of ω. Then $\beta(x)$ contains the redex $\alpha(l)$ at some position υ'. Let σ be the substitution defined by $\sigma(y) = \beta(y)$ if $y \neq x$ and $\sigma(x) = \beta(x)[\upsilon' \hookleftarrow \alpha(r)]$. Then

$$t' \xrightarrow{*}_R \sigma(g) \rightarrow_R \sigma(d) \xleftarrow{*}_R \beta(d) = t''.$$

Critical overlap case: Again we can assume without lost of generality that $\upsilon = \epsilon$. Now $\omega \in \mathcal{G}rd(g)$ and Lemma 16.1 applies. Since all critical pairs are joinable, by monotony of the rewrite relation,

$$t' \xrightarrow{*}_R u \xleftarrow{*}_R t''.$$

□

Since finite systems have a finite number of critical pairs, their local confluence is decidable. Moreover, according to Theorem 16.1, a terminating rewrite system R is confluent iff all its critical pairs are joinable. This was the idea of the *superposition test* proposed by Knuth and Bendix [KB70].
Exercice 52 — Prove that the system

$$\begin{array}{rcl} (x*y)*z & \to & x*(y*z) \\ f(x*y) & \to & f(x)*f(y) \end{array}$$

is confluent.
Answer: Hint: check termination and joinability of the only critical pair.
A system without (non-trivial) critical pair is said *non-overlapping* (or sometimes non-ambiguous). However the confluence needs termination.

Example 16.1 Consider the rewrite system R:

$$\begin{array}{rcl} f(x,x) & \rightarrow & a \\ f(x,g(x)) & \rightarrow & b \\ c & \rightarrow & g(c). \end{array}$$

The corresponding rewrite relation does not terminate, since the term c has no normal form. The rewrite relation is not confluent because $f(c,c)$ has two normal forms a and b. However the system is non-overlapping and thus locally confluent.

Exercice 53 — Consider the rewrite system R:

$$\begin{array}{rcl} f(x,x) & \rightarrow & g(x) \\ f(x,g(x)) & \rightarrow & b \\ h(c,y) & \rightarrow & f(h(y,c),h(y,y)). \end{array}$$

Prove that it is locally confluent, that each term has at least one normal form. Find a counter-example of confluence. Prove that the system is not terminating.

Exercice 54 — The goal of this exercise is to prove and illustrate the following result: the union of two terminating rewrite systems R_1 and R_2 is terminating if R_1 is left-linear, R_2 is right-linear and there is no overlap between left-hand sides of R_1 and right-hand sides of R_2.

1. Prove the following result:
   If R_1 is left-linear, R_2 is right-linear and there is no overlap between left-hand sides of R_1 and right-hand sides of R_2, then R_1 quasi-commutes (see Definition 8.6) over R_2.
2. Let R_1 be:

$$\begin{array}{rcl} a(0,x) & \rightarrow & x \\ a(s(x),y) & \rightarrow & s(a(x,y)) \\ m(0,x) & \rightarrow & 0 \\ m(s(x),y) & \rightarrow & a(m(x,y),y), \end{array}$$

   and R_2 be:

$$\begin{array}{rcl} x \leq x & \rightarrow & true \\ 0 \leq s(x) & \rightarrow & true \\ s(x) \leq 0 & \rightarrow & false \\ s(x) \leq s(y) & \rightarrow & x \leq y. \end{array}$$

   Prove that $R_1 \cup R_2$ is terminating using the previous questions.
3. Prove that $R_1 \cup R_2$ is confluent.

Answer:

1. see [RV80].
2. The termination of R_1 can be proved using a recursive path ordering with the precedence $m > a > s$.
   The termination of R_2 can be proved using a recursive path ordering with the precedence $\leq > true$ and $\leq > false$.
3. The only critical pair of $R_1 \cup R_2$ comes from the first and fourth rules of R_2 and is convergent.

16.3 Transition rules for completion

Following [BD87] the completion process is described by a set of transition rules.

A *transition rule*, in the completion framework, transforms pairs (P, R) where P is a set of equalities and R a set of rewrite rules. This transition rule is usually followed by a condition that specifies in which case the transition rule applies.

Thus a completion procedure is described by a set of transition rules that allow computing (P_{i+1}, R_{i+1}) from (P_i, R_i) using a derivation relation $\mapsto\!\!\!\rightarrow$.

Let $R_* = \bigcup_{i\geq 0} R_i$ be the set of all generated rules and $P_* = \bigcup_{i\geq 0} P_i$ the set of all generated equalities.

Let R_∞ and P_∞ be respectively the set of persisting rules and pairs, i.e. the sets effectively generated by completion starting from a set of axioms P_0. Formally:

$$P_\infty = \bigcup_{i\geq 0} \bigcap_{j>i} P_j \quad \text{and} \quad R_\infty = \bigcup_{i\geq 0} \bigcap_{j>i} R_j.$$

Equalities are ordered according to a given reduction ordering, denoted $>$. Rewrite rules are compared by the following ordering:
$l \rightarrow r >> g \rightarrow d$ iff

- either $l \sqsupset g$ (l is strictly greater than g in the encompassment ordering),
- or l and g are subsumption equivalent ($l \equiv g$) and $r > d$ in the given reduction ordering.

The completion procedure is expressed by the set $\mathcal{C}$ of transition rules presented in Figure 16.1.

Orient	$P \cup \{p = q\}, R$	$\mapstochar\twoheadrightarrow$	$P, R \cup \{p \rightarrow q\}$ if $p > q$
Deduce	P, R	$\mapstochar\twoheadrightarrow$	$P \cup \{p = q\}, R$ if $(p, q) \in CP(R)$
Simplify	$P \cup \{p = q\}, R$	$\mapstochar\twoheadrightarrow$	$P \cup \{p' = q\}, R$ if $p \rightarrow_R p'$
Delete	$P \cup \{p = p\}, R$	$\mapstochar\twoheadrightarrow$	P, R
Compose	$P, R \cup \{l \rightarrow r\}$	$\mapstochar\twoheadrightarrow$	$P, R \cup \{l \rightarrow r'\}$ if $r \rightarrow_R r'$
Collapse	$P, R \cup \{l \rightarrow r\}$	$\mapstochar\twoheadrightarrow$	$P \cup \{l' = r\}, R$ if $l \rightarrow_R^{g \rightarrow d} l'$ & $l \rightarrow r >> g \rightarrow d$

Figure 16.1: Standard completion rules

Orient turns an orientable equality into a rewrite rule $l \rightarrow r$ such that $l > r$. **Deduce** adds equational consequences derived from overlaps between rules. **Simplify** uses rules to simplify both sides of equalities and could be written more precisely as two transition rules **Left-Simplify** and **Right-Simplify** where $q \rightarrow_R q'$. **Delete** removes any trivial equality. **Compose** simplifies right-hand sides of rules with respect to other rules. **Collapse** simplifies the left-hand side of a rule and turns the result into a new equality, but only when the simplifying rule $g \rightarrow d$ is smaller than the disappearing rule $l \rightarrow r$ in the ordering $>>$. Note that if any equality is simplified by existing rules before being oriented, the condition of the **Collapse** transition rule is always satisfied.

These transition rules are sound in the following sense: they do not change the equational theory.

Lemma 16.2 If $(P, R) \mapstochar\twoheadrightarrow (P', R')$ then $\overset{*}{\longleftrightarrow}_{P \cup R}$ and $\overset{*}{\longleftrightarrow}_{P' \cup R'}$ are the same.

Proof: Proving that these two congruences coincide amounts to prove that:

$$\forall (p = q) \in (P \cup R) - (P' \cup R'), p \overset{*}{\longleftrightarrow}_{P' \cup R'} q$$

and

$$\forall (p = q) \in (P' \cup R') - (P \cup R), p \overset{*}{\longleftrightarrow}_{P \cup R} q.$$

Consider each transition rule.

1. Orient: $P \cup R = P' \cup R'$.
2. Deduce: $P' - P = \{p = q \mid (p, q) \in CP(R)\}$. By definition of a critical pair, there exists a peak of R: $p \leftarrow_R t \rightarrow_R q$ so $p \overset{*}{\longleftrightarrow}_R q$.
3. Simplify: $P - P' = \{p = q\}$. Then $p \rightarrow_{R'} p' \longleftrightarrow_{P'} q$.
   $P' - P = \{p' = q\}$. Then $p' \leftarrow_R p \longleftrightarrow_P q$.
4. Delete: the result is obvious.
5. Compose: $R - R' = \{l \rightarrow r\}$. Then $l \rightarrow_{R'} r' \leftarrow_{R'} r$.
   $R' - R = \{l \rightarrow r'\}$. Then $l \rightarrow_R r \rightarrow_R r'$.

6. Collapse: $(P \cup R) - (P' \cup R') = \{l \to r\}$. Then $l \to_{R'} l' \longleftrightarrow_{P'} r$.
   $(P' \cup R') - (P \cup R) = \{l' = r\}$. Then $l' \leftarrow_R l \to_R r$.

$\Box$

A completion procedure is aimed at transforming an initial set of equalities P_0 into a rewrite rule system R_∞ that is convergent and interreduced.

But the procedure can also be studied by looking at its effect on the set of proofs. Let us consider the set of all provable equalities $(t = t')$. To each (P_i, R_i) is associated a proof of $(t = t')$ using rules in R_i and equalities in P_i. Following [Bac87, Jou87], to each transition rule is associated a proof transformation rule on some kind of proofs using rules in R_* and equalities in P_*. Proving a completion procedure involves the following steps:

- each sequence of proof transformation terminates and produces a minimal proof for some complexity measure.
- every minimal proof is in the desired normal form, in this case a rewrite proof. For that, a completion procedure must satisfy a fairness requirement, which states that all necessary proof transformations are performed.

Let us now give the set of transformation rules on proofs. Two proof transformation rules reduce peaks without overlap or with variable overlap: remind that, given a rewrite system R, the different kinds of peaks $t' \leftarrow_R t \to_R t''$ that can occur are:
- either $\leftarrow_R$ and $\to_R$ do not overlap and $t' \to_R t_0 \leftarrow_R t''$,
- or $\leftarrow_R$ and $\to_R$ overlap on variables and $t' \xrightarrow{*}_R t_0 \xleftarrow{*}_R t''$,
- or $\leftarrow_R$ and $\to_R$ overlap and critical pairs computation is needed. Lemma 16.1 allows reducing such peaks with overlap by a proof transformation rule of the form

$$t' \leftarrow_R t \to_R t'' \Longrightarrow t'' \longleftrightarrow_P t'.$$

Other rules come from the transition rules for completion: remind that t, t', t'' denote any terms.

1. <u>Orient:</u> $t \overset{*}{\longleftrightarrow}{}_P^{p=q} t' \Longrightarrow t \to_R^{p\to q} t'$
2. <u>Deduce:</u> $t' \leftarrow_R^{l\to r} t \to_R^{g\to d} t'' \Longrightarrow t' \longleftrightarrow_P^{p=q} t''$
3. <u>Simplify:</u> $t \longleftrightarrow_P^{p=q} t' \Longrightarrow t \to_R^{l\to r} t'' \longleftrightarrow_P^{p'=q} t'$ if $p \to_R^{l\to r} p'$.
4. <u>Delete:</u> $t \longleftrightarrow_P^{p=p} t \Longrightarrow \Lambda$
5. <u>Compose:</u> $t \to_R^{l\to r} t' \Longrightarrow t \to_R^{l\to r'} t'' \leftarrow_R^{g\to d} t'$ if $r \to_R^{g\to d} r'$.
6. <u>Collapse:</u> $t \to_R^{l\to r} t' \Longrightarrow t \to_R^{g\to d} t'' \longleftrightarrow_P^{l'=r} t'$ if $l \to_R^{g\to g} l'$.
7. <u>Peak without overlap:</u> $t' \leftarrow_R^{l\to r} t \to_R^{g\to d} t'' \Longrightarrow t' \to_R^{g\to d} t_1 \leftarrow_R^{l\to r} t''$
8. <u>Peak with variable overlap:</u> $t' \leftarrow_R^{l\to r} t \to_R^{g\to d} t'' \Longrightarrow t' \xrightarrow{*}_R t_1 \xleftarrow{*}_R t''$

The next step is to prove that $\Longrightarrow$ is well-founded.

Lemma 16.3 [Bac87] The proof transformation relation $\Longrightarrow$ is well-founded.

Proof: Define the complexity measure of elementary proof steps by:

$$\begin{array}{rcl} c(s \longleftrightarrow_P^{p=q} t) & = & (\{s, t\}) \\ c(s \longleftrightarrow_R^{l\to r} t) & = & (\{s\}, l \to r) \end{array}$$

By convention, the complexity of the empty proof Λ is $c(\Lambda) = (\emptyset)$. Complexities of elementary proof steps are compared using the lexicographic combination, denoted $>_{ec}$ of the multiset extension $>^{mult}$ of the reduction ordering for the first component, and the ordering $>>$ for the second component. Since both $>$ and $>>$ are well-founded, so is $>_{ec}$. The complexity of a non-elementary proof is the multiset of the complexities of its elementary proof steps. Complexities of non-elementary proofs are compared using the multiset extension $>_c$ of $>_{ec}$, which is also well-founded.

Then the relation $\succ_c$ defined on proofs by $\mathcal{P} \succ_c \mathcal{P}'$ if $c(\mathcal{P}) >_c c(\mathcal{P}')$ is well-founded. Now $\Longrightarrow$ is well-founded since $\Longrightarrow \subseteq \succ_c$:

1. <u>Orient:</u> $c(t \overset{p=q}{\longleftrightarrow}_P t') = (\{t,t'\}) >_c c(t \rightarrow_R^{p\rightarrow q} t') = (\{t\}, p \rightarrow q)$,
since $\{t,t'\} >^{mult} \{t\}$.
2. <u>Deduce:</u>
$c(t' \leftarrow_R^{l\rightarrow r} t \rightarrow_R^{g\rightarrow d} t'') = \{(\{t\}, l \rightarrow r), (\{t\}, g \rightarrow d)\} >_c c(t' \overset{p=q}{\longleftrightarrow}_P t'') = (\{t', t''\})$
just by comparing the first components: $t > t'$ and $t > t''$.
3. <u>Simplify:</u> $c(t \overset{p=q}{\longleftrightarrow}_P t') = (\{t,t'\}) >_c$
$c(t \rightarrow_R^{l\rightarrow r} t'' \overset{p'=q}{\longleftrightarrow}_P t') = \{(\{t\}, l \rightarrow r), (\{t'', t'\})\}$
since $\{t,t'\} >^{mult} \{t\}$ and $\{t,t'\} >^{mult} \{t'', t'\}$.
4. <u>Delete:</u> $c(t \overset{p=p}{\longleftrightarrow}_P t) = (\{t,t\}) >_c c(\Lambda) = (\emptyset)$
since $\{t,t\} >^{mult} \emptyset$.
5. <u>Compose:</u> $c(t \rightarrow_R^{l\rightarrow r} t') = (\{t\}, l \rightarrow r) >_c c(t \rightarrow_R^{l\rightarrow r'} t'' \leftarrow_R^{g\rightarrow d} t') = \{(\{t\}, l \rightarrow r'), (\{t'\}, g \rightarrow d)\}$
since $(l \rightarrow r) >> (l \rightarrow r')$ and $t > t'$.
6. <u>Collapse:</u> $c(t \rightarrow_R^{l\rightarrow r} t') = (\{t\}, l \rightarrow r) >_c$
$c(t \rightarrow_R^{g\rightarrow d} t'' \overset{l'=r}{\longleftrightarrow}_P t') = \{(\{t\}, g \rightarrow d), (\{t'', t'\})\}$
since $(l \rightarrow r) >> (g \rightarrow d)$ and $t > t'$, $t > t''$.
7. <u>Peak without overlap:</u> $c(t' \leftarrow_R^{l\rightarrow r} t \rightarrow_R^{g\rightarrow d} t'') = \{(\{t\}, l \rightarrow r), (\{t\}, g \rightarrow d)\} >_c c(t' \rightarrow_R^{g\rightarrow d} t_1 \leftarrow_R^{l\rightarrow r} t'') = \{(\{t'\}, l' \rightarrow r'), (\{t\}, g' \rightarrow d')\}$
just by comparing the first components: $t > t'$ and $t > t''$.
8. <u>Peak with variable overlap:</u> $c(t' \leftarrow_R^{l\rightarrow r} t \rightarrow_R^{g\rightarrow d} t'') >_c c(t' \xrightarrow{*}_R t_1 \xleftarrow{*}_R t'')$
again just by comparing the first components of each elementary step appearing in these proofs.

□

The exact correspondence between the proof transformation $\Longrightarrow$ and the derivation $\mapsto\!\!\!\!\rightarrow$ is stated by the following results.

These proof transformation rules must *reflect* the transition rules of $\mathcal{C}$ in the following sense: at each transition step $\mapsto\!\!\!\!\rightarrow$, a given proof either does not change or is transformed into another one by $\Longrightarrow$.

Definition 16.3 [Bac87] $\Longrightarrow$ *reflects* $\mapsto\!\!\!\!\rightarrow$ if whenever $(P_i, R_i) \mapsto\!\!\!\!\rightarrow (P_{i+1}, R_{i+1})$ and $\mathcal{P}$ is a proof in $(P_i \cup R_i)$, then there is a proof $\mathcal{P}'$ in $(P_{i+1} \cup R_{i+1})$ such that $\mathcal{P} \overset{*}{\Longrightarrow} \mathcal{P}'$.

Because the proof transformation rules have been built from the transition rules, it is easy to verify that

Proposition 16.1 $\Longrightarrow$ reflects $\mapsto\!\!\!\!\rightarrow$.

The fairness hypothesis states that any proof reducible by $\Longrightarrow$ will eventually be reduced. In other words, no reducible proof is forgotten. Fairness specifies under which conditions a control strategy is correct.

Definition 16.4 A derivation $(P_0, R_0) \mapsto\!\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\!\rightarrow \cdots$ is *fair* if whenever $\mathcal{P}$ is a proof in $(P_i \cup R_i)$ reducible by $\Longrightarrow$, then there is a proof $\mathcal{P}'$ in $(P_j \cup R_j)$ at some step $j \geq i$ such that $\mathcal{P} \overset{+}{\Longrightarrow} \mathcal{P}'$.

A sufficient condition to satisfy the fairness hypothesis can be given:

Proposition 16.2 A derivation $(P_0, R_0) \mapsto\!\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\!\rightarrow \cdots$ is fair if $CP(R_\infty) \subseteq P_*$, R_∞ is reduced and P_∞ is empty.

Proof: We have to prove that if $\mathcal{P}$ is a proof in $(P_i \cup R_i)$ reducible by $\Longrightarrow$, then there is $\mathcal{P}'$ in $(P_j \cup R_j)$ at some step $j \geq i$ such that $\mathcal{P} \overset{+}{\Longrightarrow} \mathcal{P}'$.

If one of the transformation rules <u>Orient</u>, <u>Simplify</u>, <u>Delete</u>, <u>Compose</u>, or <u>Collapse</u> applies to $\mathcal{P}$, then $P_i \cup R_i \neq P_\infty \cup R_\infty$. So one of the transition rule of $\mathcal{C}$ will apply.

If the transformation rule <u>Deduce</u> applies with a non-persisting rule, then for some $j \geq i$, $(P_i, R_i) \mapsto\!\!\!\!\rightarrow \cdots \mapsto\!\!\!\!\rightarrow (P_j, R_j)$ where R_j does not contain this rule any more. If the transformation rule <u>Deduce</u> applies with persisting rules, then by hypothesis there exists k such that the computed critical pair is in P_k. Since $P_\infty = \emptyset$ by hypothesis, then for some $j \geq i, j > k$, $(P_i, R_i) \mapsto\!\!\!\!\rightarrow \cdots \mapsto\!\!\!\!\rightarrow (P_j, R_j)$ where P_j does not contain the critical pair any more.

For all these cases, since $\Longrightarrow$ reflects $\mapsto\!\!\!\!\rightarrow$, $\mathcal{P} \overset{+}{\Longrightarrow} \mathcal{P}'$.

If one of the transformation rules <u>Peak without overlap</u>, or <u>Peak with variable overlap</u> applies, then by Theorem 16.1, $\mathcal{P}$ contains the peak $t' \leftarrow_{R_i} t \rightarrow_{R_i} t''$ that can be replaced by a rewrite proof $t'' \xrightarrow{*}_{R_i} t \xleftarrow{*}_{R_i} t''$. Then $j = i$ and $\mathcal{P} \overset{+}{\Longrightarrow} \mathcal{P}'$. □

Note that the critical pairs criteria (see Section 19.3) can be used to improve the above requirements for fairness by considering less critical pairs.

When fairness is ensured, proofs have normal forms w.r.t. $\Longrightarrow$. The following proof normalization theorem is the basis of the Church-Rosser property.

Theorem 16.2 *If a derivation $(P_0, R_0) \mapsto\!\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\!\rightarrow \cdots$ is fair, then any proof $t \overset{*}{\longleftrightarrow}_{P_i \cup R_i} t'$ for $i \geq 0$, has a rewrite proof $t \overset{*}{\longrightarrow}_{R_\infty} s \overset{*}{\longleftarrow}_{R_\infty} t'$.*

Proof: By induction on $\Longrightarrow$. Assume that $t \overset{*}{\longleftrightarrow}_{P_i \cup R_i} t'$ is not a rewrite proof $t \overset{*}{\longrightarrow}_{R_\infty} s \overset{*}{\longleftarrow}_{R_\infty} t'$. Then it must contain a peak or a non-persistent step, and thus is reducible by $\Longrightarrow$. By fairness, $t \overset{*}{\longleftrightarrow}_{P_i \cup R_i} t' \overset{+}{\Longrightarrow} t \overset{*}{\longleftrightarrow}_{P_j \cup R_j} t'$ for some step $j \geq i$. By induction hypothesis, there is a rewrite proof $t \overset{*}{\longrightarrow}_{R_\infty} s \overset{*}{\longleftarrow}_{R_\infty} t'$. □

Theorem 16.3 *If the derivation $(P_0, R_0) \mapsto\!\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\!\rightarrow \cdots$ satisfies $CP(R_\infty)$ is a subset of P_*, R_∞ is reduced and P_∞ is empty, then R_∞ is Church-Rosser and terminating. Moreover $\overset{*}{\longleftrightarrow}_{P_0 \cup R_0}$ and $\overset{*}{\longleftrightarrow}_{R_\infty}$ coincide on terms.*

Proof: The termination property of R_∞ is obvious since the test is incrementally processed for each rule added in R_*, thus in R_∞.

The Church-Rosser property results from the fact that any proof has a normal form for $\Longrightarrow$ which is a rewrite proof.

The fact that $\overset{*}{\longleftrightarrow}_{P_0 \cup R_0} = \overset{*}{\longleftrightarrow}_{R_\infty}$ is a consequence of Lemma 16.2. □

Exercice 55 — The following exercise is aimed at showing the usefulness of the condition in **Collapse**. It is inspired from an example of [BD89b]. Indeed it is tempting to strenghten the **Collapse** rule, so that one can reduce the left-hand side of a rule by another rule with the same left side. Let **WCollapse** be such a rule and $\mathcal{W}$ be the set of transition rules presented in Figure 16.1 in which **Collapse** is replaced by **WCollapse**.

Consider the initial set of equalities:

$$\begin{aligned} c &= a \\ g(x) &= x \\ f(x,b) &= x \\ f(x,g(y)) &= f(g(x),y) \\ f(b,z) &= c \end{aligned}$$

1. Find a lexicographic path ordering that allows orienting these equalities from left to right.
2. Prove that with the set $\mathcal{C}$, there exists a finite confluent and terminating system.
3. Justify with the set $\mathcal{W}$ the computation of the following rewrite rules:

$$\begin{aligned} c &\rightarrow b \\ f(g(b),z) &\rightarrow c \\ c &\rightarrow g(b) \\ f(g(g(b)),z) &\rightarrow c \end{aligned}$$

   Show that $c \rightarrow b$ is collapsed away in this deduction system. How is transformed the proof of the theorem $b = a$?

   Justify the computation of:

$$\begin{aligned} c &\rightarrow g(g(b)) \\ f(g(g(g(b))),z) &\rightarrow c \end{aligned}$$

   Conclude that the process never reach a convergent system. How is transformed the proof of the theorem $b = a$?

Answer: See [BD89b].

1. The precedence is $f > c > g > b > a$.
2. The system is

$$\begin{aligned} f(x,a) &\rightarrow x \\ f(a,z) &\rightarrow a \\ b &\rightarrow a \end{aligned}$$

3. The proof of the theorem $b = a$ grows indefinitely.

16.4 A completion procedure

A *standard completion procedure* is a program that takes a finite set of equalities P_0 and a reduction ordering $>$, and that generates from $(P_0, \emptyset)$ a derivation $(P_0, R_0) \mapsto\!\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\!\rightarrow \cdots$, using the transition rules in $\mathcal{C}$.

A fairness sufficient condition is ensured by marking a rule whenever its critical pairs with other rules have been computed and added to P has in the completion procedure described in Figure 16.2.

```
PROCEDURE COMPLETION (P, R, >)
IF P is not empty
THEN choose a pair (p,q) in P; P := P - {(p,q)};
     p':= R-normal form(p); q':= R-normal form(q);
     CASE p' = q'  THEN  R := COMPLETION (P, R, >)
          p' > q' THEN l:=p'; r:=q';
                       (P,R) := SIMPLIFICATION (P, R, l -> r);
                       R := COMPLETION (P, R U {l -> r}, >)
          q' > p' THEN l:=q'; r:=p';
                       (P,R) := SIMPLIFICATION (P, R, l -> r);
                       R := COMPLETION (P, R U {l -> r}, >)
          ELSE STOP with FAILURE
     END CASE;
ELSE IF all rules in R are marked
     THEN RETURN R; STOP with SUCCESS
     ELSE Choose an unmarked rule l -> r fairly;
          P := CRITICAL-PAIRS (l -> r, R);
          Mark the rule l -> r in R;
          R := COMPLETION (P, R, >)
     END IF
END IF
END COMPLETION
```

Figure 16.2: A standard completion procedure

The `SIMPLIFICATION` procedure applies transition rules **Compose** and **Collapse**. Note that the condition in the **Collapse** transition rule is useless in this procedure, due to the chosen strategy. The `CRITICAL-PAIRS` procedure computes the critical pairs of the given rule $l \to r$ with other rules in R and add them to P.

Example 16.2 Starting with P_0 as

$$\begin{array}{rcl} x+e &=& x \\ (x+y)+z &=& x+(y+z) \end{array}$$

the completion procedure yields the system

$$\begin{array}{rcl} x+e &\to& x \\ (x+y)+z &\to& x+(y+z) \\ x+(e+y) &\to& x+y. \end{array}$$

Example 16.3 An additive group G is defined by the set of equalities

$$\begin{array}{rcl} x+e &=& x \\ x+(y+z) &=& (x+y)+z \\ x+i(x) &=& e \end{array}$$

The completion process produces

$$\begin{array}{rcl} x+e &\to& x \\ e+x &\to& x \\ x+(y+z) &\to& (x+y)+z \\ x+i(x) &\to& e \end{array}$$

$$\begin{array}{rcl} i(x)+x & \rightarrow & e \\ i(e) & \rightarrow & e \\ (y+i(x))+x & \rightarrow & y \\ (y+x)+i(x) & \rightarrow & y \\ i(i(x)) & \rightarrow & x \\ i(x+y) & \rightarrow & i(y)+i(x) \end{array}$$

Example 16.4 Starting with P_0 as

$$\begin{array}{rcl} x*1 & = & x \\ 1*x & = & x \\ i(x)*(x*y) & = & y \end{array}$$

the completion procedure yields the system

$$\begin{array}{rcl} x*1 & \rightarrow & x \\ 1*x & \rightarrow & x \\ i(x)*(x*y) & \rightarrow & y \\ i(x)*x & \rightarrow & 1 \\ x*i(x) & \rightarrow & 1 \\ i(1) & \rightarrow & 1 \\ i(i(x)) & \rightarrow & x \\ x*(i(x)*y) & \rightarrow & y. \end{array}$$

Example 16.5 Starting with P_0 as

$$\begin{array}{rcl} 1*x & = & x \\ x*i(x) & = & 1 \\ (x*y)*z & = & x*(y*z) \end{array}$$

the completion procedure yields the system for the so-called L-R theory.

$$\begin{array}{rcl} 1*x & \rightarrow & x \\ x*i(x) & \rightarrow & 1 \\ (x*y)*z & \rightarrow & x*(y*z) \\ i(1) & \rightarrow & 1 \\ i(x*y) & \rightarrow & i(y)*i(x) \\ x*(i(x)*y) & \rightarrow & y \\ i(x)*(x*y) & \rightarrow & y \\ x*1 & \rightarrow & i(i(x)) \\ i(i(i(x))) & \rightarrow & i(x) \\ i(i(x))*y & \rightarrow & x*y. \end{array}$$

Starting with P_0 as

$$\begin{array}{rcl} x*1 & = & x \\ i(x)*x & = & 1 \\ (x*y)*z & = & x*(y*z) \end{array}$$

the completion procedure yields the system for the so-called R-L theory.

$$\begin{array}{rcl} x*1 & \rightarrow & x \\ i(x)*x & \rightarrow & 1 \\ (x*y)*z & \rightarrow & x*(y*z) \end{array}$$

$$\begin{array}{rcl}
i(1) & \rightarrow & 1 \\
i(x*y) & \rightarrow & i(y)*i(x) \\
1*x & \rightarrow & i(i(x)) \\
x*i(i(y)) & \rightarrow & x*y \\
i(i(i(x))) & \rightarrow & i(x) \\
x*(y*i(y)) & \rightarrow & x \\
x*(i(i(y))*z) & \rightarrow & x*(y*z) \\
x*(y*(i(y)*z)) & \rightarrow & x*z \\
i(x)*(x*y) & \rightarrow & i(i(y))
\end{array}$$

Comparing with group completion, it clearly follows that a group is an L-R algebra and also an R-L algebra. Also there exists an L-R algebra that is not an R-L algebra: consider for instance the axiom $x * 1 = x$ that is not derivable in L-R theory. In the same way, it can be proved that there exists an R-L algebra that is not an L-R algebra

Further Readings: Many other examples of the complete set of rules conputation can be found in the literature. Let us mention in particular [Hul80b, Hul80c].

Exercice 56 — Let $\mathcal{F} = \{c, f\}$ where c is a constant and f a unary operator. Complete the set of equalities

$$\begin{array}{rcl}
f(f(f(f(f(x))))) & = & x \\
f(f(f(x))) & = & x
\end{array}$$

and simultaneously transform the two following proofs

$$\begin{array}{c}
f(c) \longleftrightarrow f(f(f(f(f(f(c)))))) \longleftrightarrow f(f(f(c))) \longleftrightarrow c \\
f(f(c)) \longleftrightarrow f(f(f(f(f(c))))).
\end{array}$$

Finally, this section is concluded with a unicity result, stating that the result of the completion process is unique, up to variable renamings.

Theorem 16.4 *Let R_1 and R_2 be two interreduced convergent rewrite systems that generate the same equivalence relation ($\overset{*}{\longleftrightarrow}_{R_1} = \overset{*}{\longleftrightarrow}_{R_2}$). If both are contained into the same reduction ordering, then they are identical up to variable renaming.*

Proof: See [Met83]. □

16.5 Issues of completion

Three outcomes are possible for a completion procedure: success, failure or divergence.

Definition 16.5 A n-step derivation $(P_0, \emptyset) \mapsto\!\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\!\rightarrow \cdots (P_n, R_n)$, is said *successful* if it is fair, P_n is empty, R_n is convergent and reduced.

The derivation *fails* if there is no fair derivation that begins with these n-steps.

A completion procedure is *correct* if it generates only fair successful derivations when it does not fail.

Note that a correct completion procedure may generate infinite fair derivations or finite failing derivations. Cases of failure are due to non-orientable equalities, for instance permutative axioms, a simple example being the commutativity axiom. In general there is no reduction ordering that can orient all critical pairs. However when equalities are made of ground terms, a total simplification ordering can be used and then failure is impossible and completion is guaranteed to terminate.

Theorem 16.5 *[Lan75b] Given a total reduction ordering on ground terms, and a finite set E of equalities made of ground terms, a fair completion derivation always generate a finite and convergent system R_∞.*

Proof: Since the ordering is total, any equality can be ordered. Because there is no variable, **Deduce** never applies and new equalities are created only by **Collapse**. If a rule $l \rightarrow r$ is reduced by $g \rightarrow d$, then the new created rule is either $l[d] \rightarrow r$ or $r \rightarrow l[d]$. Since $l > l[d]$ and $l > r$, the process of creating new rules cannot go on indefinitely. So any fair derivation is finite. □

The ground completion process can even be achieved in $O(n\ log\ n)$ steps has shown by W. SnyderSnyder [Sny89] improving on a polynomial bound given by [GNP$^+$88].

Example 16.6 Let $\mathcal{F} = \{c, f\}$ where c is a constant and f a unary operator. For any total ground ordering, the completion of the set of ground equalities

$$\begin{array}{rcl} f(f(f(f(f(c))))) & = & c \\ f(f(f(c)))) & = & c \end{array}$$

yields the convergent system

$$f(c) \rightarrow c$$

Note that two successful derivations starting with the same P_0 and the same reduction ordering $>$ must output the same convergent and reduced system, up to a renaming of variables. This is because there is only one convergent and reduced system, up to a renaming of variables, that is contained in $>$ [BL80, Met83]. Thus if there exists a convergent and reduced system R for P_0 contained in $>$, then a correct procedure starting with P_0 and $>$ cannot succed with any system but R. However it may fail without finding it. Furthermore, if R is finite, an infinite fair completion derivation is impossible.

Success may depend on the choice of orientation, as shown in the next example.

Example 16.7 Consider P_0 to be

$$\begin{array}{rcl} (x * y) * z & = & x * (y * z) \\ f(x * y) & = & f(x) * f(y) \end{array}$$

If these equalities are oriented from left to right, the completion procedure terminates with

$$\begin{array}{rcl} (x * y) * z & \rightarrow & x * (y * z) \\ f(x * y) & \rightarrow & f(x) * f(y) \end{array}$$

But if the second equality is oriented from right to left, and if we use a recursive path ordering with a precedence such that $* >_{\mathcal{F}} f$ and a left-to-right status for $*$, the rules

$$\begin{array}{rcl} (x * y) * z & \rightarrow & x * (y * z) \\ f(x) * f(y) & \rightarrow & f(x * y) \end{array}$$

generate an infinite set of rules.

Example 16.8 The theory of idempotent semi-groups (sometimes called bands) is defined by a set E of two axioms:

$$\begin{array}{rcl} (x * y) * z & = & x * (y * z) \\ x * x & = & x \end{array}$$

From $P_0 = E$ the completion generates

$$\begin{array}{rcl} (x * y) * z & \rightarrow & x * (y * z) \\ x * x & \rightarrow & x \\ x * (x * z) & \rightarrow & x * z \\ x * (y * (x * y)) & \rightarrow & x * y \\ x * (y * (x * (y * z))) & \rightarrow & x * (y * z) \\ & \dots & \\ x * (y * (z * (y * (x * (y * (z * x)))))) & \rightarrow & x * (y * (z * x)) \\ & \dots & \end{array}$$

If the completion does not terminate and generates an infinite set of rewrite rules, it can be used as a semi-decision procedure for proving the validity of a theorem.

Theorem 16.6 *[Hue81]: Let $(P_0, R_0) \mapsto\!\!\!\twoheadrightarrow (P_1, R_1) \mapsto\!\!\!\twoheadrightarrow \cdots$ be a fair non-failing derivation. The theorem $(t = t')$ is valid in $(P_0 \cup R_0)$ $((t = t') \in \mathcal{T}h(P_0 \cup R_0))$ iff there exists a step i and thus a set of rewrite rules R_i such that $t \downarrow_{R_i} = t' \downarrow_{R_i}$.*

Of course, it is not possible to prove that a theorem is not valid using Theorem 16.6, because it is not possible to prove that there is no step i satisfying the given condition. So Theorem 16.6 only provides a semi-decision method.

Example 16.9 [Der89] In the theory of idempotent semi-groups defined in Example 16.8, in order to prove the theorem

$$((x * ((y * z) * x)) * x) * ((y * z) * x) = (x * y) * (y * ((x * y) * (z * x)))$$

it is enough to check that the partial system shown in Example 16.8 reduces both terms of the theorem to $x * (y * (z * x))$.

Exercice 57 — Complete the set of equalities $\{f(g(f(x))) = g(f(x))\}$.

Example 16.10 An homomorphism h on an additive group G is defined by the property that $\forall x, y \in G,\ h(x + y) = h(x) + h(y)$. From the set of equalities defining a group and an homomorphism h

$$\begin{array}{rcl} x + e & = & x \\ x + (y + z) & = & (x + y) + z \\ x + i(x) & = & e \\ h(x + y) & = & h(x) + h(y) \end{array}$$

the completion process produces

$$\begin{array}{rcl} x + e & \to & x \\ e + x & \to & x \\ x + (y + z) & \to & (x + y) + z \\ x + i(x) & \to & e \\ i(x) + x & \to & e \\ i(e) & \to & e \\ (y + i(x)) + x & \to & y \\ (y + x) + i(x) & \to & y \\ i(i(x)) & \to & x \\ i(x + y) & \to & i(y) + i(x) \\ h(x + y) & \to & h(x) + h(y) \\ h(e) & \to & e \\ h(i(x)) & \to & i(h(x)) \end{array}$$

The kernel of an homomorphism h defined on a group G is the set

$$ker(h) = \{u \in G | h(u) = e\}$$

To prove that the kernel of h is a normal subgroup, two equalities have to be proved:

$$\begin{array}{rcl} h(a + i(b)) & = & e \\ h(i(x) + a + x) & = & e \end{array}$$

In order to perform these proofs by completion, two new identities are first added to the previous set of rewrite rules, stating that a and b are two elements of the kernel:

$$\begin{array}{rcl} h(a) & = & e \\ h(b) & = & e \end{array}$$

The completion process turned them into rewrite rules:

$$\begin{array}{rcl} h(a) & \rightarrow & e \\ h(b) & \rightarrow & e \end{array}$$

During the completion, no other rule is generated and the process stops with a complete set R. Then

$$\begin{array}{rcl} h(a+i(b)) & \xrightarrow{*}_R & e \\ h(i(x)+a+x) & \xrightarrow{*}_R & e \end{array}$$

which proves the theorems.

Example 16.11 An interesting application of rewriting and completion techniques is the study of several calculi defined for handling explicit substitutions in λ-calculus. The λ-calculus was defined by Church in the 30's in order to develop a general theory for computable functions. The λ-calculus contains one mere rule, β called β-reduction, which consists in replacing some variable occurrences by a term. This substitution is described in the meta-language of λ-calculus and introduces the need for cautious renaming operations to take into account the binding rules environment. This was the motivation for the notation introduced by de Bruijn in the 70's for counting the binding depth of a variable. Another improvement has been introduced in the 85's to deal with substitutions: instead of defining them in the meta-language, a new operator is introduced to explicitely denote the substitution to perform. So in addition to a rule (*Beta*) which starts the substitution, other rules are added to propagate it down to variables. Based on this idea, several calculi, regrouped under the name $\lambda\sigma$-calculi, have been studied and progressively refined in order to obtain suitable properties like termination and confluence.

Historicaly a first rewrite system called CCL (Categorical Combinatory Logic) was proposed to axiomatize substitutions [Cur86] but it is not confluent. The confluence can be retrieved by using a system $\lambda\sigma$ with two sorts, *term* and *substitution*, but only on ground terms and substitutions.

$$\begin{array}{lrcl}
(Beta) & (\lambda a)b & \rightarrow & a[b \cdot id] \\
(VarId) & \mathtt{1}[id] & \rightarrow & \mathtt{1} \\
(VarCons) & \mathtt{1}[a \cdot s] & \rightarrow & a \\
(App) & (ab)[s] & \rightarrow & (a[s])(b[s]) \\
(Abs) & (\lambda a)[s] & \rightarrow & \lambda(a[\mathtt{1} \cdot (s \circ \uparrow)]) \\
(Clos) & (a[s])[t] & \rightarrow & a[s \circ t] \\
(IdL) & id \circ s & \rightarrow & s \\
(ShiftId) & \uparrow \circ id & \rightarrow & \uparrow \\
(ShiftCons) & \uparrow \circ (a \cdot s) & \rightarrow & s \\
(Map) & (a \cdot s) \circ t & \rightarrow & a[t] \cdot (s \circ t) \\
(Ass) & (s_1 \circ s_2) \circ s_3 & \rightarrow & s_1 \circ (s_2 \circ s_3)
\end{array}$$

The study of critical pairs suggests to add four rules to get a system $\lambda\sigma'$ with the local confluence property; then confluence is destroyed in general, due to a non-linear rule, but is true for semi-ground terms that is terms that may contain variables of sort *term* but no variable of sort *substitution*.

$$\begin{array}{lrcl}
(Beta) & (\lambda a)b & \rightarrow & a[b \cdot id] \\
(Id) & a[id] & \rightarrow & a \\
(VarCons) & \mathtt{1}[a \cdot s] & \rightarrow & a \\
(App) & (ab)[s] & \rightarrow & (a[s])(b[s]) \\
(Abs) & (\lambda a)[s] & \rightarrow & \lambda(a[\mathtt{1} \cdot (s \circ \uparrow)]) \\
(Clos) & (a[s])[t] & \rightarrow & a[s \circ t] \\
(IdL) & id \circ s & \rightarrow & s \\
(IdR) & s \circ id & \rightarrow & s \\
(ShiftCons) & \uparrow \circ (a \cdot s) & \rightarrow & s \\
(Map) & (a \cdot s) \circ t & \rightarrow & a[t] \cdot (s \circ t) \\
(Ass) & (s_1 \circ s_2) \circ s_3 & \rightarrow & s_1 \circ (s_2 \circ s_3) \\
(ShiftId) & \mathtt{1} \cdot \uparrow & \rightarrow & id \\
(SCons) & \mathtt{1}[s] \cdot (\uparrow \circ s) & \rightarrow & s
\end{array}$$

Eventually in 89, T. Hardin and J.-J. Lévy found a calculus totally confluent with the same good properties as $\lambda\sigma$. The set of rules $\lambda\sigma_{\Uparrow}$ introduces an additional unary operator $\Uparrow$ to avoid the critical pair at the origin of the problematic rule.

$(Beta)$	$(\lambda a)b$	$\rightarrow$	$a[b \cdot id]$
(App)	$(ab)[s]$	$\rightarrow$	$(a[s])(b[s])$
$(Lambda)$	$(\lambda a)[s]$	$\rightarrow$	$\lambda(a[\Uparrow(s)])$
$(Clos)$	$(a[s])[t]$	$\rightarrow$	$a[s \circ t]$
$(Varshift1)$	$\mathtt{n}[\uparrow]$	$\rightarrow$	$\mathtt{n+1}$
$(Varshift2)$	$\mathtt{n}[\uparrow \circ s]$	$\rightarrow$	$ttn + 1[s]$
$(FVarCons)$	$\mathtt{1}[a \cdot s]$	$\rightarrow$	a
$(FVarLift1)$	$\mathtt{1}[\Uparrow(s)]$	$\rightarrow$	$\mathtt{1}$
$(FVarLift2)$	$\mathtt{1}[\Uparrow(s) \circ t]$	$\rightarrow$	$\mathtt{1}[t]$
$(RVarCons)$	$\mathtt{n+1}[a \cdot s]$	$\rightarrow$	$\mathtt{n}[s]$
$(RVarLift1)$	$\mathtt{n+1}[\Uparrow(s)]$	$\rightarrow$	$\mathtt{n}[s \circ \uparrow]$
$(RVarLift2)$	$\mathtt{n+1}[\Uparrow(s) \circ t]$	$\rightarrow$	$\mathtt{n}[s \circ (\uparrow \circ t)]$
$(AssEnv)$	$(s \circ t) \circ u$	$\rightarrow$	$s \circ (t \circ u)$
$(MapEnv)$	$(a \cdot s) \circ t$	$\rightarrow$	$a[t] \cdot (s \circ t)$
$(ShiftCons)$	$\uparrow \circ (a \cdot s)$	$\rightarrow$	s
$(ShiftLift1)$	$\uparrow \circ \Uparrow(s)$	$\rightarrow$	$s \circ \uparrow$
$(ShiftLift2)$	$\uparrow \circ (\Uparrow(s) \circ t)$	$\rightarrow$	$s \circ (\uparrow \circ t)$
$(Lift1)$	$\Uparrow(s) \circ \Uparrow(t)$	$\rightarrow$	$\Uparrow(s \circ t)$
$(Lift2)$	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\rightarrow$	$\Uparrow(s \circ t) \circ u$
$(LiftEnv)$	$\Uparrow(s) \circ (a \cdot t)$	$\rightarrow$	$a \cdot (s \circ t)$
(IdL)	$id \circ s$	$\rightarrow$	s
(IdR)	$s \circ id$	$\rightarrow$	s
$(LiftId)$	$\Uparrow(id)$	$\rightarrow$	id
(Id)	$a[id]$	$\rightarrow$	a

16.6 Conclusion

More recent research on completion process focuses on the study of divergence. The practical interest of completion processes is limited by the possibility of generating infinite sets of rewrite rules. Moreover the uniqueness of the result of the completion procedure [Hue81, JK86c], given a fixed ordering for orienting equalities, implies that it cannot be expected to find another completion strategy for which the completion terminates. However, changing the ordering can have an influence on the behaviour of the Knuth-Bendix procedure and, in some cases, it can even be possible to avoid its divergence, for example by means described in [Her88, Les86].

The problem of the generation of an infinite set of rewrite rules has been attacked using two different approaches. On one hand, sufficient conditions for predicting the divergence have been given in [Her89, MP88]. They are based on the notion of forward or backward crossed rewrite systems. On the other hand, the notion of schematization using meta-rules has been proposed in [Kir89b]. Given an infinite set of rules, the problem is to find a finite set of schemas, called meta-rules, where some variables, called meta-variables, may have infinite sets of possible values. A formal notion of schematization was proposed in [Kir89b], but discovering the schemas was yet a matter of heuristics. These results are brought together in [KH90] to deduce, from the syntactic conditions of divergence, the automatic generation of meta-rules. This provides, in a reasonably large class of divergent systems, a way to avoid guessing the schemas.

Other schematizations are proposed in [CHK90] through the notion of hyperterms, in [Gra88] through term schemes and in [Sal92] through recursive terms. All these works study the unification problem on these new structures.

In [SK92], it is shown that, even in the restricted case of string rewriting systems, there exist convergent systems generated by completion which are not recursively enumerable, although they may haave a decidable word problem.

Chapter 17

Ordered completion

17.1 Introduction

The problem considered in this chapter is again to prove an equational theorem in an equational theory described by a set of universally quantified equalities.

The completion procedure transforms this given set of equalities into a set of rewrite rules that allows deciding the validity of any equational theorem by normalisation. However, the completion procedure must be supplied with a well-founded ordering used to determine in which direction an equality must be directed. Even when such an ordering is provided, the procedure may fail to find a canonical set of rules though one exists [DMT88]. The way out is to backtrack and try another completion strategy.

The first problem addressed here is the case of failure when a non-orientable critical pair is generated. In this issue nothing can be said on the set of rewrite rules generated so far. Practical implementations of a completion procedure often postpone the equality, with the hope that a further generated rule will simplify it. An ordered (or unfailing) completion procedure pushes this solution even further, since it does not stop with a non-orientable equality and may terminate with a non-empty set of equalities. This amounts to work with the notion of ordered rewriting that does not require that an equality always be used from left to right, but satisfies nevertheless a decreasing property with respect to a given ordering. For proving equalities, the considered Church-Rosser property must take into account the ordered rewrite relation described by all orientable instances of equalities. These notions are defined in Section 7.2 of Chapter 7.

An additional property of an ordered completion procedure is to obviate the need for backtracking, since it is guaranteed to find a convergent system, provided it exists. The only assumption is that no persisting equality can be simplified. This is the second point handled in this chapter.

If the completion does not terminate and generates an infinite set of rewrite rules, it can be used as a semi-decision procedure for proving the validity of a theorem: the theorem $(t = t')$ is valid iff there exists a step i and thus a set of rewrite rules R_i such that $t \downarrow_{R_i} = t' \downarrow_{R_i}$. Ordered completion is based on the same idea and provides also a semi-decision procedure for the word problem. The last point developed here is the use of the ordered completion as a refutationally complete equational theorem prover. As such, it is possible to prove existentially quantified theorems by negating them and deriving a contradiction by ordered completion.

17.2 Ordered critical pairs

Remind that an ordered rewrite system $(E, >)$ is a set of equalities E together with a reduction ordering $>$. The ordered rewriting relation is just rewriting with ordered instances of equalities.

As in standard completion, peaks are eliminated thanks to a critical pairs lemma. However the notion of critical pairs is extended to equalities.

Definition 17.1 Let $(g = d)$ and $(l = r)$ two equalities in E with disjoint sets of variables. If there exists a position ω in g such that $g_{|\omega}$ is not a variable, $g_{|\omega}$ and l are unifiable with the most general unifier ψ, and if in addition $\psi(d) \not> \psi(g)$ and $\psi(r) \not> \psi(l)$, then $(\psi(g[\omega \hookleftarrow r]), \psi(d))$ is an *ordered critical pair* of $(l = r)$ into $(g = d)$.

More intuitively, ordered critical pairs are such that the steps $\psi(d) \longleftrightarrow_E \psi(g) \longleftrightarrow_E \psi(g[\omega \hookleftarrow r])$ can form a peak, i.e. there exists a substitution μ such that $\mu(\psi(d)) \leftarrow_{E^>} \mu(\psi(g)) \rightarrow_{E^>} \mu(\psi(g[\omega \hookleftarrow r]))$.

Note that the conditions $\psi(d) \not> \psi(g)$ and $\psi(r) \not> \psi(l)$ in Definition 17.1 are less restrictive than requiring $\psi(g) > \psi(d)$ and $\psi(l) > \psi(r)$, since $\psi(g), \psi(d), \psi(l), \psi(r)$ may be non-ground terms.

Let $OCP(E)$ denote the set of ordered critical pairs between two equalities of E. Note that when E is actually a rewrite system R contained in $>$, then $CP(R) = OCP(R)$.

The critical pair lemma can be adapted to ordered critical pairs.

Lemma 17.1 Let t, t', t'' be terms in $\mathcal{T}(\mathcal{F})$ such that

$$t' \longleftarrow^{\omega,\alpha,l=r}_{E^>} t \longrightarrow^{\Lambda,\beta,g=d}_{E^>} t''$$

with $\omega \in \mathcal{G}(g)$. Then, there exists an ordered critical pair

$$(p, q) = (\psi(g[\omega \hookleftarrow r]), \psi(d))$$

of the equality $(l = r)$ on $(g = d)$ at position ω such that $\psi = mgu(l, g_{|\omega})$ and $\beta\alpha \succeq_V \psi$ with $V = \mathcal{V}(g) \cup \mathcal{V}(l)$. Therefore there exists a substitution τ such that $t' = \tau(p)$ and $t'' = \tau(q)$.

Proof: Since $t = \beta(g)$ and $t_{|\omega} = \beta(g)_{|\omega} = \alpha(l)$, $g_{|\omega}$ and l are unifiable by the unifier $\sigma = \beta\alpha$. Let $\psi = mgu(l, g_{|\omega})$ and τ such that $\tau(\psi(x)) = \sigma(x), \forall x \in \mathcal{V}(g) \cup \mathcal{V}(l)$. By definition, there exists an ordered critical pair $(p, q) = (\psi(g[\omega \hookleftarrow r]), \psi(d))$ and $t' = \tau(p)$ and $t'' = \tau(q)$. □

Proposition 17.1 For any ground terms $t, t', t'' \in \mathcal{T}(\mathcal{F})$ such that $t' \leftarrow_{E^>} t \rightarrow_{E^>} t''$, either there exists s satisfying $t' \overset{*}{\longrightarrow}_{E^>} s \overset{*}{\longleftarrow}_{E^>} t''$, or else $t' \longleftrightarrow_{OCP(E)} t''$.

Proof: The only difference with the proof of the similar result for standard rewriting is the case of variable overlapping where the hypothesis of an ordering total on ground terms is crucial to ensure that the diagram can be closed. Indeed in this case $t' \overset{*}{\longrightarrow}_{E^>} \sigma(g)$, $t'' \overset{*}{\longrightarrow}_{E^>} \sigma(d)$, and both instances need to be comparable with the ordering, in order to close the diagram. □

Of course, when an equality $g = d$ is not oriented, superposition of other equalities must be performed on both g and d. Moreover superposition at position Λ of the equality on itself must also be performed, as shown in the next example.

Example 17.1 Let E be the following set of equalities:

$$x + g(y) \quad = \quad x + f(z)$$

Assume that the given ordering is the lexicographic path ordering defined by the precedence $g > f > b > a$. Then there is a peak

$$a + f(b) \leftarrow_{E^>} a + g(a) \rightarrow_{E^>} a + f(a)$$

and $a + f(b) > a + f(a)$. However there is no equation allowing to rewrite $a + f(b) \rightarrow_{E^>} a + f(a)$.

But there is a critical pair of the equality on its variant

$$x' + g(y') = x' + f(z')$$

that gives

$$x + f(z) = x + f(z').$$

Thus with this new equality $a + f(b) \rightarrow a + f(a)$.

This example also makes clear that superposition at position Λ does not produce trivial consequences when the set of variables of d is not included in the set of variables of g.

17.3 Transition rules for ordered completion

The ordered completion process has only two possible issues: either it generates a finite ordered rewrite system or it loops forever. In the first case, it provides a decision procedure for validity of any equational theorem $t = t'$ where $t, t' \in \mathcal{T}(\mathcal{F})$. In the second case, it only provides a semi-decision procedure.

Let P be a set of equalities and $>$ a reduction ordering. For interreduction of equalities an ordering on equalities is needed. Let $>>$ be defined by $(p = q) >> (g = d)$ if $p \sqsupset g$ or $p \equiv g$ and $q > d$ in the given reduction ordering.

The ordered completion procedure is expressed by the set $\mathcal{U}$ of transition rules given in Figure 17.1.

This transition rule system is evidently sound, since the class of provable theorems on ground terms is unchanged by any of these transitions. More precisely

Deduce	P
	$\mapsto\!\!\!\!\twoheadrightarrow$
	$P \cup \{p = q\}$
	if $(p, q) \in OCP(P)$
Delete	$P \cup \{p = p\}$
	$\mapsto\!\!\!\!\twoheadrightarrow$
	P
Collapse	$P \cup \{p = q\}$
	$\mapsto\!\!\!\!\twoheadrightarrow$
	$P \cup \{p' = q\}$
	if $(p \rightarrow_{P>}^{g=d} p'$ & $p = q >> g = d$

Figure 17.1: Ordered completion rules

Lemma 17.2 If $P \mapsto\!\!\!\!\twoheadrightarrow P'$, then $\overset{*}{\longleftrightarrow}_P$ and $\overset{*}{\longleftrightarrow}_{P'}$ coincide on $\mathcal{T}(\mathcal{F})$.

Proof: Consider each transition rule.

1. Deduce: $P' - P = \{p = q \mid (p, q) \in OCP(P)\}$. By definition of an ordered critical pair, there exists a ground instantiation of $(p = q)$ that corresponds to a peak of $P^>$.
2. Delete: the result is obvious.
3. Collapse: $P' - P = \{p' = q \mid q \longleftrightarrow_P p \rightarrow_{P>} p'\}$. Thus any ground instance σ satisfies $\sigma(q) \longleftrightarrow_P \sigma(p) \longleftrightarrow_P \sigma(p')$.
   $P - P' = \{p = q \mid q \longleftrightarrow_{P'} p' \leftarrow_{P'>} p\}$. Again any ground instance σ satisfies $\sigma(q) \longleftrightarrow_{P'} \sigma(p') \longleftrightarrow_{P'} \sigma(p)$.

□

Considering now the proof transformation that reflects the transition rule system $\mathcal{U}$, we get the following transformation rules on proofs.

1. Deduce: $t' \leftarrow_{P>}^{l=r} t \rightarrow_{P>}^{g=d} t'' \Longrightarrow t' \longleftrightarrow_{P}^{p=q} t''$
2. Delete: $t \longleftrightarrow_{P}^{p=p} t \Longrightarrow \Lambda$
3. Collapse: $t \longleftrightarrow_{P}^{p=q} t' \Longrightarrow t \rightarrow_{P>}^{l=r} t'' \longleftrightarrow_{P}^{p'=q} t'$.
4. Peak without overlap: $t' \leftarrow_{P>}^{l=r} t \rightarrow_{P>}^{g=d} t'' \Longrightarrow t' \rightarrow_{P>}^{g=d} t_1 \leftarrow_{P>}^{l=r} t''$
5. Peak with variable overlap: $t' \leftarrow_{P>}^{l=r} t \rightarrow_{P>}^{g=d} t'' \Longrightarrow t' \overset{*}{\longrightarrow}_{P>} t_1 \overset{*}{\longleftarrow}_{P>} t''$

The next step is to prove that $\Longrightarrow$ is well-founded.

Lemma 17.3 [Bac87] If $>$ is a reduction ordering that can be extended into a ground-total reduction ordering $\gg$, then the proof transformation relation $\Longrightarrow$ is well-founded.

Proof: Define the complexity measure of elementary proof steps by:

$$\begin{array}{rcl} c(s \longleftrightarrow_{P}^{g=d} t) & = & (\{s\}, g, t) \text{ if } s \gg t \\ c(s \longleftrightarrow_{P}^{g=d} t) & = & (\{t\}, d, s) \text{ if } t \gg s \end{array}$$

By convention, the complexity of the empty proof Λ is $c(\Lambda) = (\emptyset)$. Complexities of elementary proof steps are compared using the lexicographic combination, denoted $>_{ec}$ of the multiset extension $\gg^{mult}$ of the ground-total reduction ordering for the first component, the ordering $\sqsupset$ for the second component, and the ground-total reduction ordering for the third component. Since both $\gg$ and $\sqsupset$ are well-founded, so is $>_{ec}$. The complexity of a non-elementary proof is the multiset of the complexities of its elementary proof steps. Complexities of non-elementary proofs are compared using the multiset extension $>_c$ of $>_{ec}$, which is also well-founded.

Then the relation $\succ_c$ defined on proofs by $\mathcal{P} \succ_c \mathcal{P}'$ iff $c(\mathcal{P}) >_c c(\mathcal{P}')$ is well-founded. Now $\Longrightarrow$ is well-founded since $\Longrightarrow \subseteq \succ_c$:

1. Deduce:
   $c(t' \xleftarrow{l=r}_{P>} t \xrightarrow{g=d}_{P>} t'') = \{(\{t\}, l, t'), (\{t\}, g, t'')\} >_c c(t' \xleftrightarrow{p=q}_{P} t'')$
   just by comparing the first components: $t \gg t'$ and $t \gg t''$.
2. Delete: $c(t \xleftrightarrow{p=p}_{P} t) = (\{t\}, p, t) >_c c(\Lambda) = (\emptyset)$
   since $\{t\} \gg^{mult} \emptyset$.
3. Collapse: If $t \gg t'$, $c(t \xleftrightarrow{p=q}_{P} t') = (\{t\}, p, t') >_c c(t \xrightarrow{g=d}_{P>} t'' \xleftrightarrow{p'=q}_{P} t')$. Indeed this last complexity measure is
   - either $\{(\{t\}, g, t''), (\{t''\}, p', t')\}$, if $t'' \gg t'$ and the result holds since in a first case, $p \sqsupset g$ and $t \gg t''$, and in a second case, $p \equiv g$ and $q > d$ implies $t' \gg t''$.
   - or $\{(\{t\}, g, t''), (\{t'\}, q, t'')\}$, if $t' \gg t''$ and the result holds since in a first case, $p \sqsupset g$ and $t \gg t''$, and in a second case, $p \equiv g$ and $q > d$ implies $t' \gg t''$.

   If $t' \gg t$, $c(t \xleftrightarrow{p=q}_{P} t') = (\{t'\}, q, t) >_c c(t \xrightarrow{l=r}_{P>} t'' \xleftrightarrow{p'=q}_{P} t') = \{(\{t\}, l, t''), (\{t'\}, q, t'')\}$, since $t' \gg t \gg t''$.
4. Peak without overlap: $c(t' \xleftarrow{l=r}_{P>} t \xrightarrow{g=d}_{P>} t'') = \{(\{t\}, l, t'), (\{t\}, g, t'')\} >_c c(t' \xrightarrow{g=d}_{P>} t_1 \xleftarrow{l=r}_{P>} t'') = \{(\{t'\}, g, t_1), (\{t''\}, l, t_1)\}$
   just by comparing the first components: $t > t'$ and $t > t''$.
5. Peak with variable overlap: again just by comparing the first components of each step which comes up in the proof $t' \xrightarrow{*}_{P>} t_1 \xleftarrow{*}_{P>} t''$.

□

The connection between ordered completion and the proof transformation relation can be stated as follows:

Proposition 17.2 $\Longrightarrow$ reflects $\mapsto\!\!\!\rightarrow$, i.e. whenever $P_i \mapsto\!\!\!\rightarrow P_{i+1}$ and $\mathcal{P}$ is a ground proof in P_i, then there is a ground proof $\mathcal{P}'$ in P_{i+1} such that $\mathcal{P} \overset{*}{\Longrightarrow} \mathcal{P}'$.

An ordered completion procedure that handles all the ordered critical pairs, produces an ordered rewrite set which is ground Church-Rosser with respect to the given reduction ordering.

Theorem 17.1 *[Bac87, BDP89] Let $P_0 = E$ be the set of equalities, and $>$ be a reduction ordering that can be extended to a ground-total reduction ordering $\gg$. If $P_0 \mapsto\!\!\!\rightarrow P_1 \mapsto\!\!\!\rightarrow \ldots$ is a derivation such that $OCP(P_\infty)$ is a subset of P_*, then P_∞ is Church-Rosser with respect to $\gg$ on ground terms.*

Proof: We prove by induction on $\Longrightarrow$ that, whenever there is a proof $\mathcal{P}$ of $(t = t')$ in P_i, for some i, then there is a ground rewrite proof of $(t = t')$ in P_∞. Lemma 17.2 then implies that P_∞ is ground Church-Rosser w.r.t. $>$.

The assertion holds trivially if $\mathcal{P}$ is a ground rewrite proof that uses only persisting equalities. Otherwise, if non-persisting equalities are used, according to Proposition 17.2, there exists a proof $\mathcal{P}'$ in some P_j, $j \geq i$, such that $\mathcal{P} \overset{+}{\Longrightarrow} \mathcal{P}'$. Then the induction hypothesis applied to $\mathcal{P}'$ yields the result.

If $\mathcal{P}$ is a persisting proof that contains a peak, say $t' \leftarrow_{P_i^>} t \rightarrow_{P_i^>} t''$, then Proposition 17.1 applies. Either the peak can be replaced by a rewrite proof $t' \xrightarrow{*}_{P_i^>} t \xleftarrow{*}_{P_i^>} t''$. Then $j = i$ and $\mathcal{P} \overset{+}{\Longrightarrow} \mathcal{P}'$. Or $t' \longleftrightarrow_{OCP(P_\infty)} t''$ with an ordered critical pair between persisting equalities. Since $OCP(P_\infty)$ is a subset of P_*, there exists $j \geq i$ such that $t' \longleftrightarrow_{P_j} t''$. Again there exists a proof $\mathcal{P}'$ in P_j such that $\mathcal{P} \overset{+}{\Longrightarrow} \mathcal{P}'$. Then the induction hypothesis yields the result. □

Ordered completion generalizes standard completion in that for each derivation in standard completion, there is a corresponding derivation in ordered completion [Bac87].

17.4 An unfailing completion procedure

An *unfailing completion procedure* is a program that takes a finite set of equalities P_0 and a reduction ordering $>$ that can be extended to a ground-total reduction ordering, and that generates from P_0 a derivation $P_0 \mapsto\!\!\!\rightarrow P_1 \mapsto\!\!\!\rightarrow \ldots$, using the transition rules in $\mathcal{U}$. An unfailing completion procedure is given in Figure 17.2.

The `SIMPLIFICATION` procedure computes simplified forms of l and r, using orientable instances of equalities in P, according to transition rule *Collapse*.

```
PROCEDURE UNFAIL-COMPLETION (P, >)
IF all equalities in P are marked
THEN RETURN P; STOP with SUCCESS
ELSE Choose an unmarked equality (l = r) fairly;
     P := P-{(l=r)};
     (l'= r') := SIMPLIFICATION (l = r, P);
     IF l' = r'  THEN  P := UNFAIL-COMPLETION (P, >)
     ELSE  P := P U {(l'= r')} U ORDERED CRITICAL-PAIRS (l'= r',P);
           Mark the equality (l'= r') in P;
           P := UNFAIL-COMPLETION (P, >)
     END IF
END IF
END UNFAIL-COMPLETION
```

Figure 17.2: An ordered completion procedure

The ORDERED CRITICAL-PAIRS procedure computes all ordered critical pairs between $(l = r)$ and other equalities in P. An equality is marked whenever its ordered critical pairs with other equalities have been computed and added to P.

Chosing an unmarked equality *fairly* can be implemented by labelling equalities increasingly when they are introduced. Chosing the equality with the smallest label is fair. If an equality persists, it has been marked so its critical pairs have been computed.

Example 17.2 This example of the theory of entropic groupoids comes from [HR87, Rus87a]:

$$(x*y)*(z*w) = (x*z)*(y*w) \tag{17.1}$$
$$(x*y)*x = x. \tag{17.2}$$

By superposition of the second equality on the first one at position $\wedge$, a critical pair

$$(((z*w)*z)*(y*w), z*w)$$

is produced. This pair is then reduced by 17.2 into

$$z*(y*w) = z*w. \tag{17.3}$$

Now 17.1 and 17.3 produce the critical pair

$$((y*w)*x)*w = y*w. \tag{17.4}$$

17.3 reduces 17.1 to a new equality

$$(x*y)*w = (x*z)*w. \tag{17.5}$$

Assuming that 17.5 has instances orientable from left to right, it can be used to reduce 17.4 to

$$((y*z)*x)*w = y*w. \tag{17.6}$$

The ordered completion then terminates with the set:

$$\begin{aligned}(x*y)*x &= x\\ z*(y*w) &= z*w\\ ((y*z)*x)*w &= y*w\\ (x*y)*w &= (x*z)*w.\end{aligned}$$

Exercice 58 — Ternary Boolean algebras are defined by two rules and one equality:

$$\begin{aligned}f(y,x,x) &\rightarrow x\\ f(x,y,g(y)) &\rightarrow x\\ f(f(v,w,x),y,f(v,w,z)) &= f(v,w,f(x,y,z)).\end{aligned}$$

Show that ordered completion can derive the following rules and equality:

$$\begin{array}{rcl} f(x,x,y) & \rightarrow & x \\ f(g(x),x,y) & \rightarrow & y \\ f(x,y,x) & \rightarrow & x \\ g(g(x)) & \rightarrow & x \\ f(x,y,z)) & = & f(z,y,x)). \end{array}$$

Answer:

When the ordered completion terminates with such a finite ordered rewriting system, it can be used to prove or disprove any equational theorem ($t = t'$) in this theory. Since the Church-Rosser property only holds for ground terms, variables, in the theorem to be proved, must be considered as new constants in an enriched signature. These constants must be linearily ordered and smaller than any other symbol for the extended ordering. Then normal forms of both terms are computed and are equals iff the theorem holds. Since the target theorem is only used in the reduction process but not in the deduction process, its variables will never be instantiated and may be treated as constants. This justifies including them specifically in the ordering.

Example 17.3 Coming back to the previous Example 17.2, assume that the theorem

$$\forall x_1, x_2, x_3, \quad x_1 * (x_1 * x_2) = ((x_1 * x_1) * x_3) * x_2$$

is to be proved. Negating this theorem, we get

$$\exists x_1, x_2, x_3, \quad x_1 * (x_1 * x_2) \neq ((x_1 * x_1) * x_3) * x_2$$

Considering $x_1 < x_2 < x_3$ as three ordered constants less than $*$, this theorem is valid since both sides of the equality reduce to a same term $(x_1 * x_2)$, using respectively the equalities

$$\begin{array}{rcl} z * (y * w) & = & z * w \\ ((y * z) * x) * w & = & y * w. \end{array}$$

Exercice 59 — Consider again the theory defined in Example 17.2. and prove the universally quantified equational theorem:

$(x_1 * (y_1 * u_1)) * v_1 = (x_1 * x_1) * v_1$.

Answer: The ordering is extended to take into account the variables of the theorem. Consider the lexicographic path ordering based on the following precedence: $* > x_1 > y_1 > u_1 > v_1$.

Each side of the theorem matches with the left-hand side of the fourth rule. Replacing z in the right-hand side of this rule by the smallest symbol v_1, two ordered rewriting steps can be applied. The normal form of each side of the theorem is: $(x_1 * v_1) * v_1$.

Ordered completion can be used as a semi-decision procedure, as in the next example.

Example 17.4 Consider the following set of equalities E:

$$\begin{array}{rcl} x + y & = & x + x \\ (x - y) + z & = & (x + z) - y \\ (x + y) - y & = & x. \end{array}$$

Assume that we want to prove the universally quantified theorem $((x_1 - y_1) + z_1 = x_1)$.

Using a recursive path ordering such that $+ > -$, the two first equalities give the critical pair $((x - y) + (x - y) = (x + z) - y)$, whose left-hand side is reduced by the second equality, giving a new equality

$$(e): \ (x + (x - y)) - y = (x + z) - y.$$

Then the third equality, superposed on the right-hand side of (e), produces a critical pair

$$(e'): \ (x + (x - y)) - y = x.$$

(e') simplifies (e) into $(x = (x + z) - y)$.

At this point the theorem to be proved is reduced, using $(x - y) + z = (x + z) - y$ to

$$(x_1 + z_1) - y_1 = x_1$$

which is in turn reduced, using (e'') to a trivial equality $x_1 = x_1$.

17.5 Construction of canonical systems

According to the previous results, an ordered completion procedure is aimed to build a ground Church-Rosser term rewriting system. However if a convergent and interreduced system exists for the theory E, it is possible to design an ordered completion procedure that will find it. The additional hypothesis is that the set of persisting equalities is interreduced, as stated by the next definition.

Definition 17.2 A derivation $P_0 \mapsto\!\!\!\rightarrow P_1 \mapsto\!\!\!\rightarrow \dots$ is *simplifying* if

- $OCP(P_\infty)$ is a subset of P_*,
- for any equality $(s = t) \in P_\infty$, s and t are irreducible.

An ordered completion procedure is *simplifying* if it generates only simplifying derivations.

Provided there exists a convergent and interreduced system R that presents E, a simplifying ordered completion procedure produces a set of equalities that is the same as R, up to a renaming.

Theorem 17.2 *Let R be an interreduced convergent system for E and $>$ a reduction ordering containing R. If $>$ can be extended to a ground-total reduction ordering, then a simplifying ordered completion procedure generates, from $P_0 = E$ and $>$, a derivation such that P_∞ is the same as R up to a variable renaming.*

Proof: see [Bac87, BDP89]. □

This result requires that the given reduction ordering, containing R, extends to a ground-total reduction ordering. However there exist some reduction orderings induced by reduced canonical rewrite systems that cannot be extended to a ground-total reduction ordering, as proved by the following example.

Example 17.5 Consider the rewrite rules set R:

$$\begin{array}{rcl} f(h(x)) & \rightarrow & f(i(x)) \\ g(i(x)) & \rightarrow & g(h(x)) \\ h(a) & \rightarrow & c \\ i(a) & \rightarrow & c. \end{array}$$

A reduction ordering for R must contain $h(a) > i(a)$ by considering the first rule. Then it must contain $g(h(a)) > g(i(a))$. But considering now the second rule, it must contain $g(i(a)) > g(h(a))$, which is incompatible with the previous requirement.

A possible implementation of a simplifying ordered completion procedure is obtained by splitting the set P of equalities into a set R of oriented equalities (or rules) and a set P' of non orientable ones. Whenever P'_∞ is empty and R_∞ is interreduced, then R_∞ is convergent.

Example 17.6 In [MN90], from the two laws

$$\begin{array}{rcl} (x * x) * y & = & y \\ (x * y) * z & = & (y * z) * x \end{array}$$

that axiomatize groups of exponent two, the ordered completion procedure produces the following set of critical pairs

$$\begin{array}{rcl} (x * y) * x & \rightarrow & y \\ (x * y) * y & \rightarrow & x \\ x * x & = & y * y \\ (x * y) * z & \rightarrow & x * (y * z) \\ x * y & = & y * x \\ x * (y * z) & = & y * (x * z) \\ x * x & \rightarrow & 1 \\ x * (x * y) & \rightarrow & y \\ x * 1 & \rightarrow & x \\ 1 * x & \rightarrow & x \end{array}$$

After elimination of joinable critical pairs, the following ordered rewrite system E is obtained:

$$\begin{array}{rcl}
(x*y)*z & \to & x*(y*z) \\
x*y & = & y*x \\
x*(y*z) & = & y*(x*z) \\
x*x & \to & 1 \\
x*(x*y) & \to & y \\
x*1 & \to & x \\
1*x & \to & x
\end{array}$$

Exercice 60 — Consider the initial set of equalities E_0:

$$\begin{array}{rcl}
1*(-x+x) & = & 0 \\
1*(x+-x) & = & x+-x \\
-x+x & = & y+-y
\end{array}$$

1. Show that a standard completion procedure fails for any reduction ordering given as input.
2. Prove that the following set of rules is a convergent rewrite system for the equational theory E_0:

$$\begin{array}{rcl}
-x+x & \to & 0 \\
x+-x & \to & 0 \\
1*0 & \to & 0
\end{array}$$

3. Apply ordered completion to E_0.
4. Unorientable equality as the last one in E_0, can sometimes be dealt with by adding a new (minimal) constant c and replacing $-x+x=y+-y$ by two rewrite rules $-x+x\to c$ and $y+-y\to c$. Prove that standard completion starting from

$$\begin{array}{rcl}
1*(-x+x) & = & 0 \\
1*(x+-x) & = & x+-x \\
-x+x & \to & c \\
y+-y & \to & c
\end{array}$$

then succeeds in constructing a convergent system in a conservative extension of E_0.

Answer: See [BD89b].

1.
2.
3.
4. The convergent system is

$$\begin{array}{rcl}
-x+x & \to & c \\
x+-x & \to & c \\
1*c & \to & c \\
0 & \to & c
\end{array}$$

17.6 Proofs by refutation

Ordered completion can also be adapted to act as a refutational theorem prover.

Let E be a set of equalities and $(t=t')$ an equational theorem to be proved in the theory described by E, with $t,t'\in\mathcal{T}(\mathcal{F},\mathcal{X})$. Assume that t_0 and t'_0 are the skolemized versions of respectively t and t', that is terms whose variables are now considered as a set $\mathcal{H}$ of new constants disjoint from $\mathcal{F}$. Introduce also two new constants T and F together with a new binary function symbol eq. The problem on $\mathcal{T}(\mathcal{F},\mathcal{X})$ is now translated into a problem on

$$\mathcal{T}_{\mathcal{H}}=\mathcal{T}(\mathcal{F}\cup\mathcal{H},\mathcal{V})\cup\{T,F\}\cup\{eq(t,t')\mid t,t'\in\mathcal{T}(\mathcal{F}\cup\mathcal{H},\mathcal{V})\}.$$

Let also

$$E^*=E\cup\{eq(x,x)=T,eq(t_0,t'_0)=F\}.$$

Note that T and F can be any constants. This is why we do not write them *true* and *false*, which could let think to boolean values.

The intended property is refutational completeness in the following sense.

Definition 17.3 The transition rule system $\mathcal{U}$ is *refutationally complete* if $(t = t')$ is valid in E implies $(T = F)$ can be derived from E^* using $\mathcal{U}$.

So a refutation will be found if the equality $(T = F)$ is generated.

Definition 17.4 A *refutation* is any derivation $P_0 \mapsto\!\!\!\rightarrow P_1 \mapsto\!\!\!\rightarrow \dots$ for which P_* contains the equality $(T = F)$.

A *fair refutation* is any derivation $P_0 \mapsto\!\!\!\rightarrow P_1 \mapsto\!\!\!\rightarrow \dots$ for which P_* contains the equality $(T = F)$ and such that $OCP(P_\infty)$ is a subset of P_*.

An additional hypothesis on the reduction ordering is necessary for proofs by refutation.

Definition 17.5 A reduction ordering is called *admissible* if it can be extended to a ground-total reduction ordering in which T and F are the two smallest elements.

The refutational completeness of ordered completion is expressed in the following result:

Theorem 17.3 *[Rus87a, Bac87] Let E be a set of equalities and $>$ be an admissible reduction ordering. Then the equality $(t = t')$ is valid in E (i.e. $(t \overset{*}{\longleftrightarrow}_E t')$) iff the ordered completion procedure generates a refutation from $P_0 = E^*$ and $>$.*

Proof: Let us first prove that $(T \overset{*}{\longleftrightarrow}_E F)$ iff the ordered completion procedure generates a fair refutation.

By soundness and correctness of ordered completion, $(T \overset{*}{\longleftrightarrow}_E F)$ iff there is a ground rewrite proof with respect to $>$ of $(T = F)$ in P_i for some i. Since T and F are minimal for $>$, this rewrite proof must be $(T \longleftrightarrow_{P_i} F)$, which implies that $(T = F) \in P_i$.

Now because $\overset{*}{\longleftrightarrow}_{P_i} \subseteq \overset{*}{\longleftrightarrow}_{P_0}$, there also exists a proof of $(T = F)$ in P_0. Since E contains only equalities between terms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, any shortest proof of $(T = F)$ in P_0 must be of the form $T \longleftrightarrow_{P_0} eq(t'', t'') \overset{*}{\longleftrightarrow}_E eq(t_0, t'_0) \longleftrightarrow_{P_0} F$. Moreover $eq(t'', t'') \overset{*}{\longleftrightarrow}_E eq(t_0, t'_0)$ iff $t_0 \overset{*}{\longleftrightarrow}_E t'_0$ iff $t \overset{*}{\longleftrightarrow}_E t'$. So $t \overset{*}{\longleftrightarrow}_E t'$ iff the ordered completion procedure generates a refutation. □

The next example illustrates this refutational method on a satisfiability proof.

Example 17.7 Consider the following set of equalities E [HR87, Rus87a]:

$$\begin{array}{rcl} (x-y)+z & = & (x+z)-y \\ (x+y)-y & = & x. \end{array}$$

In order to prove the existentially quantified theorem

$$\exists x, y, z, \ \ (x-y)+z = x$$

in the theory E, the following equations are added to E:

$$\begin{array}{rcl} eq(x,x) & = & T \\ eq((x-y)+z, x) & = & F. \end{array}$$

The last equality is the negated theorem. Using a recursive path ordering such that $+ > -$, the last equality reduces, using $(x-y)+z = (x+z)-y$, to

$$eq((x+z)-y, x) = F.$$

Now $(x+y)-y = x$ can be superposed in the left-hand side of this last equality and gives the critical pair

$$eq(x,x) = false.$$

A last superposition with the equality $(eq(x,x) = F)$ produces the critical pair $(T = F)$.

Example 17.8 A ground complete ordered rewrite system for Boolean rings is given by a lexicographic path ordering generated by a total ordering of the operators and constants, together with the following set of equalities.

$$\begin{array}{rclrcl} x+0 & \rightarrow & x & 0+x & \rightarrow & x \\ x+x & \rightarrow & 0 & x+(x+y) & \rightarrow & y \\ x*0 & \rightarrow & 0 & 0*x & \rightarrow & 0 \\ x*1 & \rightarrow & x & 1*x & \rightarrow & x \\ x*x & \rightarrow & x & x*(x*y) & \rightarrow & x*y \\ (x+y)+z & \rightarrow & x+(y+z) & (x*y)*z & \rightarrow & x*(y*z) \\ x*(y+z) & \rightarrow & (x*y)+(x*z) & (y+z)*x & \rightarrow & (y*x)+(z*x) \\ x+y & = & y+x & x*y & = & y*x \\ y+(x+z) & = & x+(y+z) & y*(x*z) & = & x*(y*z) \end{array}$$

Ordered completion, starting with this system and a skolemized negation of a theorem in first-order predicate calculus with equality, where $+$ is the exclusive-or and $*$ is the conjunction, can be used as the basis of a refutationally complete theorem prover [HD83].

17.7 Conclusion

The notion of proof by refutation comes from first-order logic with equality. In order to prove a formula, the inconsistency of its negation is proved by deducing a contradiction.

In 1965, J.A.Robinson proposed an inference rule, called *resolution*, that allows deriving a contradiction (the empty clause) from formulas that are inconsistent [Rob65].

In 1970, G.Robinson and Wos introduced the *paramodulation*, which was the first step to handle the equality relation in a specific way [RW69]. If ordered completion is used with the "empty" ordering, it degenerates to paramodulation (restricted of course to the purely equational case). The advantage of ordered completion over paramodulation is its use of simplification that considerably reduces the search space and the restriction of equational deduction to the computation of ordered critical pairs, which avoids superposition on variables contrary to paramodulation.

The idea of extending completion by computing equational consequences of non-orientable equalities can be traced back to the work of Brown (1975) [Bro75] and Lankford (1975) [Lan75b].

In 1983, Peterson proved the refutation completeness of an inference rule system composed of resolution, paramodulation and simplification with respect to a class of orderings isomorphic to integers [Pet83]. He already proved that it is not necessary to paramodulate on variables and introduced the simplification rule, that allows keeping terms into reduced forms.

In 1987, Rusinowitch proposed a transition rule system that is refutationally complete for first-order logic with equality [HR86, Rus87a]. Restricted to the equational case, this system behaves like an ordered completion procedure [Rus88].

Chapter 18

Completion modulo a set of equalities

18.1 Introduction

The Knuth-Bendix completion procedure is based on using equalities as rewrite rules and computing critical pairs when left-hand sides of rules overlap. If a critical pair has distinct irreducible forms, a new rule must be added and the procedure recursively applies until it maybe stops. This procedure requires the termination property of the set of rules, which can be proved by various tools. However some equalities, such as commutativity, cannot be oriented into rewrite rules without losing the termination property and cause the failure of the completion procedure. To remedy this problem requires the concepts of *class rewrite systems* and *rewriting modulo a set of axioms* presented in Section 7.3 of Chapter 7, where adequate notions of *confluence and coherence modulo* a set of equalities are defined too.

The Knuth-Bendix completion method was extended to handle the case of class rewrite system. A first approach by Lankford and Ballantyne [LB77c, LB77a, LB77b] handles the case of commutative, or more generally permutative, axioms that generate finite congruence classes. The case of infinite congruence classes is studied in [PS81, Hue80, Jou83]. Huet's approach [Hue80] is restricted to sets R of left-linear rules, while Peterson and Stickel's one [PS81] is restricted to linear equalities A for which a finite and complete unification algorithm is known. Both [LB77a] and [PS81] results yield an efficient associative-commutative completion procedure, with a few more restrictions needed for applying the first approach. A complete unification algorithm is required to compute complete sets of A-overlappings, providing complete sets of A-critical pairs. These results are unified in [Kir85b, JK86c] and sources of inefficiency inherent to unification and matching modulo a set of axioms are minimized. Finally all these completion techniques were put into the uniform framework of transition rules and proof transformation by L.Bachmair [Bac87].

Examples handled by these methods can be found in [KK86, Hul80a]. Theories with associative-commutative axioms are the most common.

18.2 Completion modulo A for left-linear rules

Huet's method for left-linear rules is presented in this section. In this context R_A is the standard rewriting relation R and the Church-Rosser property that will be obtained is the following: Remind that the rewriting relation R is Church-Rosser modulo A if

$$\overset{*}{\longleftrightarrow}_{R\cup A} \;\subseteq\; \overset{*}{\longrightarrow}_R \circ \overset{*}{\longleftrightarrow}_A \circ \overset{*}{\longleftarrow}_R \,.$$

The adequate notion of proofs in normal-form comes from this Church-Rosser property. The simplest forms of proofs which are not in normal form come from the properties of local coherence modulo A of R with R and with A.

Definition 18.1 A proof of $(t = t')$ is a *rewrite proof modulo* A for R iff $\exists t_1, t'_1, t \overset{*}{\longrightarrow}_R t_1,\ t' \overset{*}{\longrightarrow}_R t'_1$ *and* $t_1 \overset{*}{\longleftrightarrow}_A t'_1$.

A *peak* is a proof $t_1 \leftarrow_R t \rightarrow_R t_2$ and a *cliff* is a proof $t_1 \leftarrow_R t \longleftrightarrow_A t_2$ or $t_1 \longleftrightarrow_A t \rightarrow_R t_2$.

A left-linear rewrite system R is Church-Rosser modulo A iff any theorem $(t = t')$ has a rewrite proof modulo A. A proof in $A \cup R$ is a rewrite proof modulo A iff it contains no peak and no cliff. This will be achieved if R is locally coherent modulo A with both R and A.

By the critical pair lemma, every peak $t_1 \leftarrow_R t \rightarrow_R t_2$ can be replaced by a rewrite proof unless it is a proper overlap. The same result holds for cliffs, provided R is left-linear.

18.2.1 Critical pairs of rules and axioms

The notion of critical pair between rules extends to rules and axioms.

Definition 18.2 Given two rules or axioms $g \to d$ and $l \to r$ such that $\mathcal{V}ar(g) \cap \mathcal{V}ar(l) = \emptyset$ and l and g overlap at occurrence ω of $\mathcal{G}rd(g)$, then (p, q) defined by $p = \psi(g[\omega \hookleftarrow r]), q = \psi(d)$ is called a *critical pair* of $l \to r$ on $g \to d$ at occurrence ω.

If ω is not an occurrence in g or if $g_{|\omega}$ is a variable, then the rule $l \to r$ applies in the variable or substitution part of g and (p, q) is then called a *variable overlap.*

Let $\vec{A}$ be the set of oriented axioms built from A:

$$\vec{A} = \bigcup_{(g=d)\in A} \{(g \to d), (d \to g)\}.$$

The set of all critical pairs obtained by overlapping two rules or axioms in $R \cup \vec{A}$ but not both in $\vec{A}$ is denoted by $CP(R \cup A) - CP(A)$.

Lemma 18.1 Let $t' \longleftrightarrow_A t \to_R t''$ be a cliff such that $t \to_R^{\epsilon,\sigma,g\to d} t''$ and $t \longleftrightarrow_A^{\upsilon,\sigma,l\to r} t'$ (or $t \longleftrightarrow_A^{\epsilon,\sigma,g\to d} t''$ and $t \to_R^{\upsilon,\sigma,l\to r} t'$), with $\upsilon \in \mathcal{G}rd(g)$.

Then there exist a critical pair (p, q) in the set of critical pairs $CP(R \cup A) - CP(A)$ and a substitution τ such that $t' = \tau(p)$ and $t" = \tau(q)$. Therefore $t' \longleftrightarrow_{CP(R\cup A)-CP(A)} t''$.

Proof: The proof is identical to the case of a true overlapping in the critical pair lemma for standard rewriting. □

Theorem 18.1 *[Hue80] R is locally confluent and locally coherent modulo A iff every critical pair of $CP(R \cup A) - CP(A)$ is convergent.*

Proof: Assume $t' \longleftrightarrow_A t \to_R t''$ be a cliff. Then

- either $\longleftrightarrow_A$ and $\to_R$ do not overlap and $t' \to_R t_0 \longleftrightarrow_A t''$ since the two relations commute.
- or $\longleftrightarrow_A$ and $\to_R$ overlap on variables and $t' \xrightarrow{*}_R t_0 \xleftrightarrow{*}_A t_1 \xleftarrow{*}_R t''$ assuming that R is left-linear. This case has to be detailed:
  - Assume that the rule $l \to r$ applies in the subterm $\alpha(g)$ created by applying the axiom $g \to d$ in $\vec{A}$, and more precisely in a subterm $\alpha(x)$ with x being a variable of g. Let t_0 be defined as the term produced by rewriting $\alpha(x)$ using $l \to r$. Let α' defined by $\alpha'(x) = t_0$ and $\alpha'(x) = \alpha(y)$ for any $y \neq x$. Then the subterm $\alpha(d)$ in t'' rewrites to $\alpha'(d)$, t' rewrites to a term containing $\alpha'(g)$ and the two results are A-equivalent.
  - Assume that the axiom $g \to d$ applies in the subterm $\alpha(l)$ created by applying the rule $l \to r$, and more precisely in a subterm $\alpha(x)$ with x being a variable of l. Let t_0 be defined as the term produced by rewriting $\alpha(x)$ using $g \to d$. Let α' defined by $\alpha'(x) = t_0$ and $\alpha'(x) = \alpha(y)$ for any $y \neq x$. Since l is linear, t'' contains as subterm $\alpha'(l)$ that can be reduced to $\alpha'(r)$, producing a term which is A-equivalent to t'.
- The last case of cliff is when there is a true overlap and critical pairs computation is needed.

□

Note that the left-linearity of rules is crucial in this proposition. For instance, for $R = \{x + x \to x\}$ and $A = \{a = b\}$, the variable overlap $a + b \ \longleftrightarrow_A \ a + a \ \to_R \ a$ cannot be transformed in a rewrite proof modulo A but only to the proof $a + b \ \longleftrightarrow_A \ b + b \ \to_R \ b \longleftrightarrow_A \ a$.

18.2.2 Transition rules for completion modulo A with left-linearity

Let P be a set of equalities (quantified pairs of terms), R be the current rewrite system, and $>$ an A-compatible reduction ordering. Equalities are ordered according to $>$. Rewrite rules are compared by the following ordering: $l \to r >> g \to d$ iff either $g \sqsubset l$ (l is strictly greater than t in the encompassment ordering), or l and g are subsumption equivalent ($l \equiv g$) and $r > d$ in the given reduction ordering.

The completion procedure modulo A for left-linear rules is expressed by the set $\mathcal{MLC}$ of transition rules presented in Figure 18.1.

Orient	$P \cup \{p = q\}, R$	$\mapsto\!\!\!\rightarrow$	$P, R \cup \{p \to q\}$ if $p > q$
Deduce	P, R	$\mapsto\!\!\!\rightarrow$	$P \cup \{p = q\}, R$ if $(p, q) \in CP(R \cup A) - CP(A)$
Simplify	$P \cup \{p = q\}, R$	$\mapsto\!\!\!\rightarrow$	$P \cup \{p' = q\}, R$ if $p \to_R p'$
Delete	$P \cup \{p = q\}, R$	$\mapsto\!\!\!\rightarrow$	P, R if $p \overset{*}{\longleftrightarrow}_A q$
Compose	$P, R \cup \{l \to r\}$	$\mapsto\!\!\!\rightarrow$	$P, R \cup \{l \to r'\}$ if $r \to_R r'$
Collapse	$P, R \cup \{l \to r\}$	$\mapsto\!\!\!\rightarrow$	$P \cup \{l' = r\}, R$ if $l \to_R^{g \to d} l'$ & $l \to r >> g \to d$

Figure 18.1: Completion modulo A for left-linear rules

Compared to the transition rules for standard completion, the differences only appear in the computation of critical pairs in which axioms have to be taken into account, and in the *Delete* rule where syntactic equality is replaced by equality modulo A.

This transition system is evidently sound, since the class of provable theorems is unchanged by any of these transitions.

Lemma 18.2 If $(P, R) \mapsto\!\!\!\rightarrow (P', R')$ then $\overset{*}{\longleftrightarrow}_{P \cup R \cup A}$ and $\overset{*}{\longleftrightarrow}_{P' \cup R' \cup A}$ are the same.

Moreover only rules contained in $>$ are added. So the system R_∞/A is terminating for any derivation for which the intial set of rules R_0 is contained in the A-compatible reduction ordering $>$.

Theorem 18.2 *Let $R_0 = \emptyset$, P_0 be a set of equalities, and $>$ be an A-compatible reduction ordering. If $(P_0, R_0) \mapsto\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\rightarrow ....$ is a derivation such that*

- *R_∞ is left-linear,*
- *$P_\infty = \emptyset$,*
- *$CP(R_\infty \cup A) - CP(A)$ is a subset of P_**

then R_∞ is Church-Rosser modulo A and R_∞/A is terminating.

Proof: see [Bac87] $\square$

18.2.3 Completion procedure for left-linear rules

A completion procedure modulo A for left-linear rules is now presented. It is a recursive form of the procedure informally described by Huet in [Hue80].

The `SIMPLIFICATION` procedure is just applying the transition rules **Simplify**, **Compose** and **Collapse** with the rule $l \to r$. The test in the rule **Collapse** is actually useless here because equalities are always fully simplified by rules before orientation.

The `CRITICAL-PAIRS` procedure computes all critical pairs between $l \to r$ and other rules in R and with axioms in $\vec{A}$. A rule is marked whenever its critical pairs with other rules and axioms have been added computed and added to P.

Choosing an unmarked rule *fairly* can be implemented by labelling rules increasingly when they are introduced. Chosing the rule with the least label is fair.

If a rule persists, it has been marked so its critical pairs have been computed. P_∞ is empty because $\forall i, \exists j > i$ such that $P_j = \emptyset$. So $\forall i, \bigcap_{j>i} P_j = \emptyset$. R_∞ is left-linear, because this condition is checked before the introduction if any rule in P_*. Actually the procedure described above could be improved in relaxing the condition of checking left-linearity of each rewrite rule. Only left-linearity of R_∞ is required by Theorem 18.2 and non-left-linear rules could be introduced at some step j in R_j and later simplified into a linear equality giving rise to a left-linear rule.

```
PROCEDURE COMPLETION-MOD-LIN (P, R, >)
IF P is not empty
THEN choose a pair (p,q) in P ; p':=R-normal form(p);
                                 q':=R-normal form(q);
     CASE p' <-*->A q'  THEN  R := COMPLETION-MOD-LIN(P-{(p,q)},R,>)
          p' > q' AND p' linear
                  THEN  l:=p'; r:=q';
                        (P,R) := SIMPLIFICATION(P-{(p,q)},R,l -> r);
                        R := COMPLETION-MOD-LIN(P,R U {l -> r},>)
          q' > p' AND q' linear
                  THEN l:=q'; r:=p';
                       (P,R) := SIMPLIFICATION(P-{(p,q)},R,l -> r);
                       R := COMPLETION-MOD-LIN(P,R U {l -> r},>)
          ELSE STOP with FAILURE
     END CASE;
ELSE IF all rules in R are marked
     THEN RETURN R; STOP with SUCCESS
     ELSE Choose an unmarked rule l -> r fairly;
          P := CRITICAL-PAIRS (l -> r,R);
          Mark the rule l -> r in R;
          R := COMPLETION-MOD-LIN(P,R,>)
     END IF
END IF
END COMPLETION-MOD-LIN
```

Figure 18.2: A completion procedure modulo A for left-linear rules

Example 18.1 This example comes from [Hue80]:

$$\begin{array}{rcl} E(x+y) & \rightarrow & E(x).E(y) \\ E(0) & \rightarrow & 1 \\ x+0 & \rightarrow & x \\ 0+x & \rightarrow & x \\ x.1 & \rightarrow & x \\ 1.x & \rightarrow & x \end{array}$$

$$\begin{array}{rcl} x+y & = & y+x \\ (x+y)+z & = & x+(y+z) \\ x.y & = & y.x \\ (x.y).z & = & x.(y.z) \end{array}$$

Example 18.2 In [ML92], an axiomatization of the category of monoids is proposed. A monoid is a set equipped with a binary associative operator $*$ and a unique unit element e. A monoid in a category which posseses binary product and a terminal object 1, is an object K equipped with an associative multiplication m and an identity a. In the category of monoids, whose objects are monoids and whose morphims are monoids homomorphisms, there is a terminal object 1, the trivial monoid, and there exists a monoid homomorphism $a : 1 \mapsto M$. A monoid in this category is thus a monoid M equipped with morphisms $m : M \times M \mapsto M$ and $a : 1 \mapsto M$. This is axiomatized with the following set of equalities.

$$\begin{array}{rcl} x*e & = & x \\ e*x & = & x \\ (x*y)*z & = & x*(y*z) \\ m(x,a) & = & x \\ m(a,x) & = & x \\ m(m(x,y),z) & = & m(x,m(y,z)) \\ m(x*x_1, y*y_1) & = & m(x,y)*m(x_1,y_1) \end{array}$$

A standard completion process stops with a non-orientable equality

$$m(x_2, m(x_4, x_1)) \quad = \quad m(x_2, m(x_1, x_4))$$

and a set of rewrite rules

$$\begin{array}{rcl} a & \rightarrow & e \\ x * y & \rightarrow & m(x, y) \\ m(x, e) & \rightarrow & x \\ m(e, x) & \rightarrow & x \\ m(m(x, y), z) & \rightarrow & m(x, m(y, z)) \end{array}$$

If the following instance of the non-orientable equality

$$m(e, m(x_4, x_1)) \quad = \quad m(e, m(x_1, x_4))$$

is added to the system, it is simplified and the commutativity of m is deduced:

$$m(x, y) \quad = \quad m(y, x)$$

With m declared as associative and commutative,

$$\begin{array}{rcl} m(m(x, y), z) & = & m(x, m(y, z)) \\ m(x, y) & = & m(y, x) \end{array}$$

the complete system is

$$\begin{array}{rcl} a & \rightarrow & e \\ x * y & \rightarrow & m(x, y) \\ m(x, e) & \rightarrow & x \end{array}$$

The main limitation of this method is that it guarantees only the Church-Rosser property of systems with left-linear rules. This is due to the problem of elimination of variable overlap. However such problems disappear if a more powerful reduction relation modulo the A-equivalence is adopted. This idea and the corresponding completion method, are due to Peterson and Stickel [PS81] and developed in the next section.

18.3 Completion modulo A with extensions

Another rewriting relation on terms has been introduced by Peterson and Stickel [PS81] and uses matching modulo A.

Remind that a term t (R, A)-rewrites to t', denoted $t \rightarrow_{R,A} t'$, if there exist a rule $l \rightarrow r \in R$, a position ω in t and a substitution σ such that $t_{|\omega} \overset{*}{\longleftrightarrow}_A \sigma(l)$ and $t' = t[\omega \hookleftarrow \sigma(r)]$.

Note that $\rightarrow_R \;\subseteq\; \rightarrow_{R,A} \;\subseteq\; \rightarrow_{R/A}$.

Consider again $R = \{x + x \rightarrow x\}$ and $A = \{a = b\}$. the term $a + b$ obtained by the variable overlapping $a + b \;\longleftrightarrow_A\; a + a \;\rightarrow_R\; a$ is now R, A-reducible, since $a + b \;\longleftrightarrow_A\; b + b \;= \sigma(x + x)$ and reduces to b.

The rewriting relation R, A is Church-Rosser modulo A if

$$\overset{*}{\longleftrightarrow}_{R \cup A} \;\subseteq\; \overset{*}{\longrightarrow}_{R,A} \circ \overset{*}{\longleftrightarrow}_A \circ \overset{*}{\longleftarrow}_{R,A} .$$

The adequate notion of proofs in normal-form comes from this Church-Rosser property. The simplest forms of proofs which are not in normal form come from the properties of local coherence modulo A of R with R, A and with A.

Definition 18.3 A proof of $(t = t')$ is a *rewrite proof modulo* A for R iff $\exists t_1, t'_1, t \overset{*}{\longrightarrow}_{R,A} t_1,\; t' \overset{*}{\longrightarrow}_{R,A} t'_1$ *and* $t_1 \overset{*}{\longleftrightarrow}_A t'_1$.

A *peak* is a proof $t_1 \leftarrow_{R,A} t \rightarrow_R t_2$ and a *cliff* is a proof $t_1 \leftarrow_{R,A} t \longleftrightarrow_A t_2$ or $t_1 \longleftrightarrow_A t \rightarrow_{R,A} t_2$.

18.3.1 A-Critical pairs of rules

Elimination of peaks now needs another notion of critical pairs.

Definition 18.4 A non-variable term t' and a term t A- *overlap* at position ω in $\mathcal{G}rd(t)$ with a complete set Ψ of A-overlappings iff Ψ is a $CSU(t_{|\omega}, t', A)$.

Given two rules $g \to d$ and $l \to r$ such that $\mathcal{V}ar(g) \cap \mathcal{V}ar(l) = \emptyset$ and l and g A-overlap at position ω of $\mathcal{G}rd(g)$ with the complete set of A-overlappings Ψ, then the set $\{(p,q) | p = \psi(g[\omega \hookleftarrow r]), q = \psi(d), \forall \psi \in \Psi\}$ is called a *set of A-critical pairs* of the rule $l \to r$ on the rule $g \to d$ at position ω. The set of A-critical pairs of R is denoted $CP_A(R)$.

Note that $l \to r$ and $g \to d$ do not play symmetric roles in this definition.

Example 18.3 Let $(x+y)+z \to x+(y+z)$ and $(e+x') \to x'$ be rules and $(y'+e) = y'$ be an axiom. Then:

$(p,q) = (y+z, e+(y+z))$ is a critical pair of the second rule on the first at position 1, associated with the overlapped term $(e+y)+z$. Note that $q \to_R p$.

$(p,q) = (z, e+(e+z))$ is an A-critical pair of the second rule on the first at position ϵ, associated with the A-overlapped term $(e+e)+z \overset{*}{\longleftrightarrow}_A e+z$. Note that $q \overset{*}{\longrightarrow}_R p$.

Contrary to the case where A is the empty theory, superposition of a rule on itself at position ϵ is needed. Otherwise the completion process could be incomplete as shown by the following example:

Example 18.4 Let $(x+y = y+x)$ be the commutativity axiom for $+$ and consider a rewrite rule that redefines $+$ with $*$: $(x+y \to x*y)$. Obviously, the commutativity of $*$ is an equational consequence of the theory, but can only be generated by superposing the rule on itself, modulo the commutativity of $+$, at position ϵ.

Let $t' \leftarrow_R t \to_{R,A} t''$ be a peak.

- Either $\to_{R,A}$ and $\to_R$ do not overlap and $t' \to_{R,A} t_0 \leftarrow_R t''$
- or $\to_{R,A}$ and $\to_R$ overlap on variables and $t' \overset{*}{\longrightarrow}_{R,A} t_0 \overset{*}{\longleftarrow}_{R,A} t''$
- or there is a true A-overlap and computation of A-critical pairs is necessary.

Lemma 18.3 Let $t' \leftarrow_R t \to_{R,A} t''$ be a peak such that $t \to_R^{\epsilon,\sigma,g\to d} t'$ and $t \to_{R,A}^{\upsilon,\sigma,l\to r} t''$, $\upsilon \in \mathcal{G}rd(g)$. Then there exists a critical pair $(p,q) = (\psi(g[\upsilon \hookleftarrow r]), \psi(d))$ in a set of A-critical pairs of the rule $l \to r$ on the rule $g \to d$ at position υ and a substitution τ such that $\psi \in CSU(g_{|\upsilon}, l, A)$ and $\forall x \in \mathcal{V}ar(g) \cup \mathcal{V}ar(l), \sigma(x) \overset{*}{\longleftrightarrow}_A \tau(\psi(x))$, therefore $t' \overset{*}{\longleftrightarrow}_A \tau(q)$ and $t'' \overset{*}{\longleftrightarrow}_A \tau(q)$.

Proof: see [Jou83]. □

Note that we use the same substitution σ for both redexes t and $t_{|\upsilon}$. This is legal, since we can always assume that $\mathcal{V}ar(l) \cap \mathcal{V}ar(g) = \emptyset$ without loss of generality: just rename the variables when needed.

The case where $\to_R$ applies at a position υ and $\to_{R,A}$ at the outermost position ϵ does not need to be considered. Lemma 18.3 would be false for such cases!

Definition 18.5 Assume that the rule $l \to r$ overlaps the axiom $g \to d$ in $\vec{A}$ at position ω in $\mathcal{G}rd(g)$, and $\mathcal{V}ar(g) \cap \mathcal{V}ar(l) = \emptyset$. Then the *extension* or *extended rule* of $l \to r$ with respect to $g \to d$ is the rule $(g[\omega \hookleftarrow l] \to g[\omega \hookleftarrow r])$.

In general extensions of extended rules have also to be recursively computed. All these extended rules are gathered in a set denoted $EXT_A(l \to r)$. Let $EXT_A(R)$ be the set of all extensions (recursively computed) of rules in R with respect to A; R^{ext} denotes $R \cup EXT_A(R)$, i.e. the saturation of R under adjunction of extended rules.

Note that to compute an extension only needs to know the position ω for which l and $g_{|\omega}$ are A-unifiable, but does not need to compute A-unifiers. Moreover such an extension, also denoted by $g[l] \to g[r]$ for short, satisfies $g[l] > g[r]$ and its adjunction does not violate the termination property.

Example 18.5 Assume that A contains the associativity axiom of $+$, $x+(y+z) = (x+y)+z$. A rule $l \to r$ will have an extension with respect to associativity if and only if its top symbol is $+$. The extensions are $x+l \to x+r$ and $l+x \to r+x$.

18.3.2 Transition rules for completion modulo A with extensions

The simplification process on rules must be re-examined when extended rules are introduced. Actually an extended rule $g[l] \to g[r]$ is by construction reducible by $l \to r$. So unlimited simplification would collapse any extended rule to a trivial equality $g[r] = g[r]$. To prevent this phenomenon, extensions are *protected*. This means that an extension will never be simplified and can only disappear if the initial rule itself disappears. In practice, a new set of protected rules EX is introduced. Every extension will be added to EX.

Whenever a new rule is introduced its extensions with axioms in A are computed. Recursively extensions of extensions are also computed and all these extended rules are gathered in a set denoted $EXT_A(l \to r)$.

Let P be a set of equalities (quantified pairs of terms), N be the current rewrite system, EX the set of protected rules and $>$ an A-compatible reduction ordering. R will denote $N \cup EX$.

Rewrite rules are again compared by the following ordering: $l \to r >> g \to d$ iff either $g \sqsubset l$ (l is strictly greater than t in the encompassment ordering), or l and g are subsumption equivalent ($l \equiv g$) and $r > d$ in the given reduction ordering.

The completion procedure modulo A with extended rules is expressed by the the set $\mathcal{MEC}$ of transition rules presented in Figure 18.3.

$$
\begin{array}{lll}
\textbf{Orient} & P \cup \{p = q\}, N, EX & \\
\mapsto\!\!\!\!\twoheadrightarrow & P, N \cup \{p \to q\}, EX & \text{if } p > q \\
\textbf{Deduce} & P, N, EX & \\
\mapsto\!\!\!\!\twoheadrightarrow & P \cup \{p = q\}, N, EX & \text{if } (p, q) \in CP_A(R) \\
\textbf{Extend} & P, N, EX & \\
\mapsto\!\!\!\!\twoheadrightarrow & P, N, EX \cup \{l \to r\} & \text{if } (l \to r) \in EXT_A(R) \\
\textbf{Simplify} & P \cup \{p = q\}, N, EX & \\
\mapsto\!\!\!\!\twoheadrightarrow & P \cup \{p' = q\}, N, EX & \text{if } p \to_{R,A} p' \\
\textbf{Delete} & P \cup \{p = q\}, N, EX & \\
\mapsto\!\!\!\!\twoheadrightarrow & P, N, EX & \text{if } p \overset{*}{\longleftrightarrow}_A q \\
\textbf{Compose} & P, N \cup \{l \to r\}, EX \cup EXT_A(l \to r) & \\
\mapsto\!\!\!\!\twoheadrightarrow & P, N \cup \{l \to r'\}, EX \cup EXT_A(l \to r') & \text{if } r \to_{R,A} r' \\
\textbf{Collapse} & P, N \cup \{l \to r\}, EX \cup EXT_A(l \to r) & \\
\mapsto\!\!\!\!\twoheadrightarrow & P \cup \{l' = r\}, N, EX & \text{if } l \to^{g \to d}_{R,A} l' \;\&\; l \to r >> g \to d
\end{array}
$$

Figure 18.3: Completion modulo A with extensions

Let N_*, P_*, N_∞, P_∞ be defined as previously and let EX_* be the set of all generated protected rules, EX_∞ be the set of persisting protected rules:

$$EX_\infty = \bigcup_i \bigcap_{j>i} EX_j.$$

Let also $R_\infty = N_\infty \cup EX_\infty$.

This transition system is evidently sound, since the class of provable theorems is unchanged by any of these transitions.

Moreover only rules contained in $>$ are added. So the system R_∞/A is terminating for any derivation for which the initial set of rules N_0 is contained in the A-compatible reduction ordering $>$.

Theorem 18.3 *Let A be a set of axioms with a finite complete unification algorithm. Let $N_0 = \emptyset$, $EX_0 = \emptyset$, P_0 be a set of equalities, and $>$ be a reduction ordering compatible with A. If $(P_0, N_0, EX_0) \mapsto\!\!\!\!\twoheadrightarrow (P_1, N_1, EX_1) \mapsto\!\!\!\!\twoheadrightarrow ....$ is a derivation such that*

- *$CP_A(R_\infty)$ is a subset of P_*,*
- *$EXT_A(R_\infty)$ is a subset of EX_*,*
- *P_∞ is empty,*

then (R_∞, A) is Church-Rosser modulo A and R_∞/A is terminating.

Proof: see [Bac87] □

18.3.3 Completion procedure with extensions

A completion procedure modulo A with systematic adjunction of extensions is presented in Figure 18.4. It is a recursive form of the procedure described by Peterson and Stickel in [PS81].

```
PROCEDURE COMPLETION-MOD-EXT (P,R,EX,>)
IF P is not empty
THEN choose a pair (p,q) in P ; p':= R,A-normal form(p);
                                q':= R,A-normal form(q);
     CASE p' <-*->A q'  THEN  R := COMPLETION-MOD-EXT(P-{(p,q)},R,EX,>)
          p' > q' THEN  l:=p'; r:=q';
                        (P,R) := SIMPLIFICATION(P-{(p,q)},R,EX,l -> r);
                        R := COMPLETION-MOD-EXT(P,R U {l -> r},
                                                  EX U EXT(l -> r),>)
          q' > p' THEN l:=q'; r:=p';
                       (P,R) := SIMPLIFICATION(P-{(p,q)},R,EX,l -> r);
                       R := COMPLETION-MOD-EXT(P,R U {l -> r},
                                                 EX U EXT(l -> r),>)
          ELSE STOP with FAILURE
     END CASE;
ELSE IF all rules in R are marked
     THEN RETURN R; STOP with SUCCESS
     ELSE Choose an unmarked rule l -> r fairly;
          P := A-CRITICAL-PAIRS (l -> r, R U EX);
          Mark the rule l -> r in R;
          R := COMPLETION-MOD-EXT(P,R,EX,>)
     END IF
END IF
END COMPLETION-MOD-EXT
```

Figure 18.4: Completion modulo A with extensions

The `SIMPLIFICATION` procedure applies the transition rules **Simplify**, **Compose** and **Collapse** with the rule $l \to r$. The `A-CRITICAL-PAIRS` procedure computes all A-critical pairs between $l \to r$ and other rules in $R \cup EX$. A rule is marked whenever its critical pairs with other rules have been computed and added to P. The `EXT` procedure recursively computes, for a given rule, its extensions with axioms in A and extensions of extensions.

If a rule persists, it has been marked so its critical pairs have been computed. So $CP_A(N_\infty \cup EX_\infty)$ is a subset of P_*. $EXT_A(R_\infty)$ is a subset of EX_*, because extensions of a given rule and extensions of extensions are introduced simultaneously with the rule itself and only deleted if the rule does not persist. P_∞ is empty because $\forall i, \exists j > i$ such that $P_j = \emptyset$. So $\forall i, \bigcap_{j>i} P_j = \emptyset$.

The condition that all the recursively computed extensions of persisting rules are added may require an infinite set of extended rules. However this condition can be weakened for special theories A and especially for associativity and commutativity (AC for short).

Definition 18.6 Let R be a rewrite system and A be the axioms of commutativity and associativity of the symbol f. Let R^{ext} be R plus all extensions $f(l,x) \to f(r,x)$ of rules $l \to r \in R$ for which there exist no rule $l' \to r'$ in R and no substitution σ, such that $f(l,x) \overset{*}{\longleftrightarrow}_{AC} \sigma(l')$ and $f(r,x) \overset{*}{\longrightarrow}_{AC\cup R/AC} \sigma(r')$.

Example 18.6 Assume that R is

$$\begin{aligned} x+0 &\to x \\ x+(-x) &\to 0 \end{aligned}$$

Then R^{ext} consists in R plus only one extended rule $(x+(-x))+y \to y$.

The extended rule $(x+0)+y \to x+y$ is not included in R^{ext} because it is equivalent modulo AC to an instance $(x+y)+0 \to (x+y)$ of the first rule.

Actually every extension of an extended rule is equivalent modulo AC to an instance of the original rule.

Lemma 18.4 Let A be the axioms of commutativity and associativity of a symbol f. For any rewrite system R, $(R^{ext})^{ext} = R^{ext}$.

Proof: Let $l \rightarrow r \in R^{ext}$. We prove that there exist $u \rightarrow v \in R^{ext}$ and a substitution σ such that $f(l,x) \overset{*}{\longleftrightarrow}_{AC} \sigma(u)$ and $f(r,x) \overset{*}{\longrightarrow}_{AC\cup(R/AC)} \sigma(v)$.

- If $l \rightarrow r \in R$, then $f(l,x) \rightarrow f(r,x) \in R^{ext}$.
- Else $l = f(s,y) \rightarrow r = f(t,y)$ is an extension of the rule $s \rightarrow t$ in R. Let σ defined by $\sigma(y) = f(y,x)$ and $\sigma(z) = z, \forall z \neq x$. Then $f(l,x) = f(f(s,y),x) \overset{*}{\longleftrightarrow}_{AC} f(s,f(y,x)) = \sigma(l)$ and $f(r,x) = f(f(t,y),x) \overset{*}{\longleftrightarrow}_{AC} f(t,f(y,x)) = \sigma(r)$.

□

Another improvement specific to AC-completion is that AC-critical pairs obtained by overlapping a rule $l' \rightarrow r'$ on a proper subterm of an extended rule $f(l,x) \rightarrow f(r,x)$ are superfluous. This is due to the fact that such overlaps are already obtained by overlapping the same rule $l' \rightarrow r'$ on the original rule $l \rightarrow r$.

A last specificity of AC-completion is that the proper encompassment ordering modulo AC, denoted $\sqsubset_{AC}$ and defined by $g \sqsubset_{AC} l$ iff $\exists\sigma, \sigma(g) \overset{*}{\longleftrightarrow}_{AC} l_{|\omega}$ with $\omega \neq \epsilon$, or $\sigma(g) = l$ and σ is not a renaming, can be used instead of the proper encompassment ordering in the definition of $>>$ on rewrite rules for allowing more simplifications of left-hand sides of rules. For more detailled proofs, see [Bac87].

Example 18.7 This Church-Rosser axiomatization of free distributive lattices is borrowed from [PS81]:

$$\begin{array}{rcl} x \cup (x \cap y) & \rightarrow & x \\ x \cap (y \cup z) & \rightarrow & (x \cap y) \cup (x \cap z) \\ x \cap x & \rightarrow & x \\ x \cup x & \rightarrow & x \end{array}$$

$$\begin{array}{rcl} x \cup y & = & y \cup x \\ (x \cup y) \cup z & = & x \cup (y \cup z) \\ x \cap y & = & y \cap x \\ (x \cap y) \cap z & = & x \cap (y \cap z) \end{array}$$

R^{ext}, AC is Church-Rosser modulo AC and R/AC is terminating.

Example 18.8 Considering now the larger class of lattices, a different situation occurs. Lattices have the following equational axiomatization:

$$\begin{array}{rclrcll} x \cup (y \cup z) & = & (x \cup y) \cup z & x \cap (y \cap z) & = & (x \cap y) \cap z & \text{(associative)} \\ x \cup y & = & y \cup x & x \cap y & = & y \cap x & \text{(commutative)} \\ x & = & x \cup x & x & = & x \cap x & \text{(idempotent)} \\ x & = & x \cup (y \cap x) & x & = & x \cap (y \cup x) & \text{(absorptive)} \end{array}$$

Although each term in the free lattice has a canonical form unique up to AC [Whi41, Whi42], there is no finite convergent class rewrite system modulo AC for the equational theory of lattices [FJN93].

Exercice 61 — Find sufficient conditions on axioms in A to ensure that the recursive process of computing extensions can be limited to a finite number of iterations.
Answer:

18.4 An alternative to extensions

The disadvantage of systematically adding extensions, in the case of a general theory A, is that it can lead to an infinite set of rules. At least, it must be proved that recursively adding extensions of extensions is unnecessary for a given set of axioms A. An alternative is to come back to the notion of critical pairs to eliminate cliffs.

18.4.1 A-critical pairs of rules on axioms

Definition 18.7 Given an axiom $g \rightarrow d$ and a rule $l \rightarrow r$ such that $\mathcal{V}ar(g) \cap \mathcal{V}ar(l) = \emptyset$ and l and g A-overlap at position ω of $\mathcal{G}rd(g)$ with the complete set of A-overlappings Ψ, then the set

$$\{(p,q) \ \mid \ p = \psi(g[\omega \hookleftarrow r]),\ q = \psi(d),\ \forall \psi \in \Psi\}$$

is called a *set of A-critical pairs* of the rule $l \rightarrow r$ on the axiom $g \rightarrow d$ at position ω. The set of A-critical pairs of R on A is denoted $CP_A(R, A)$.

Let $t' \longleftrightarrow_A t \rightarrow_{R,A} t''$ be a cliff.

- Either $\rightarrow_{R,A}$ and $\longleftrightarrow_A$ do not overlap and $t' \rightarrow_{R,A} t_0 \longleftrightarrow_A t''$
- or $\longleftrightarrow_A$ occurs below $\rightarrow_{R,A}$ and $t'' \rightarrow_{R,A} t'$
- or $\rightarrow_{R,A}$ and $\longleftrightarrow_A$ overlap on variables and $t' \xrightarrow{*}_{R,A} t_0 \overset{*}{\longleftrightarrow}_A t_1 \xleftarrow{*}_{R,A} t''$
- or $\rightarrow_{R,A}$ and $\longleftrightarrow_A$ overlap and critical pairs computation is needed.

Lemma 18.5 Let $t' \longleftrightarrow_A t \rightarrow_{R,A} t''$ be a cliff such that $t \longleftrightarrow_A^{[\epsilon,\sigma,g\rightarrow d]} t'$ and $t \rightarrow_{R,A}^{[\upsilon,\sigma,l\rightarrow r]} t''$, with $\upsilon \in \mathcal{G}rd(g)$. Then there exists a critical pair $(p,q) = (\psi(g[\upsilon \hookleftarrow r]), \psi(d))$ in a set of A-critical pairs of the rule $l \rightarrow r$ on the axiom $g \rightarrow d$ at position υ and a substitution τ such that $\psi \in CSU(l, g_{|\upsilon}, A)$ and $\forall x \in \mathcal{V}ar(g) \cup \mathcal{V}ar(l), \sigma(x) \overset{*}{\longleftrightarrow}_A \tau(\psi(x))$, therefore $t' \overset{*}{\longleftrightarrow}_A \tau(q)$ and $t'' \overset{*}{\longleftrightarrow}_A \tau(q)$.

Proof: see [Jou83]. □

Note that extended rules reduce at the outermost position all right members of A-critical pairs since q is an instance of $g[\omega \hookleftarrow l]$ modulo A: $q = \psi(d) \overset{*}{\longleftrightarrow}_A \psi(g) = \psi(g[\omega \hookleftarrow g_{|\omega}]) = \psi(g)[\omega \hookleftarrow \psi(g_{|\omega})] \overset{*}{\longleftrightarrow}_A \psi(g)[\omega \hookleftarrow \psi(l)] = \psi(g[\omega \hookleftarrow l])$ using the homomorphic properties of substitutions. Extended rules thus appear just as a special way to enforce convergence of A-critical pairs of R on A.

The previous lemma would suggest to reduce cliffs with true overlap by adding a proof transformation rule of the form $t'' \longleftrightarrow_A t \rightarrow_{R,A} t' \Longrightarrow t'' \overset{*}{\longleftrightarrow}_A t_1 \longleftrightarrow_P t_2 \overset{*}{\longleftrightarrow}_A t'$. However introducing such a reduction would make the proof reduction relation $\Longrightarrow$ non-terminating. So another weaker rule is used: $t'' \longleftrightarrow_A t \rightarrow_{R,A} t' \Longrightarrow t'' \overset{*}{\longleftrightarrow}_A t_1 \rightarrow_R t_2 \overset{*}{\longleftrightarrow}_A t'$, provided that if the equality step $t'' \longleftrightarrow_A t$ occurs at some position ω in t'', then $t'' \rightarrow_{R,A} t_2$ at some position below ω, with A-equality steps strictly below ω. A careful examination of Lemma 18.5, where $t'' = \sigma(d)$, shows that A-equality steps actually take place in the substitution part of t'' and not at a non-variable position of d. Whenever A contains no axiom $t = x$, these A-equality steps never occur at the outermost position ϵ in t''.

18.4.2 Transition rules for completion without extensions

Completion modulo A without adjunction of extensions is formulated by splitting the transition rule for adjunction of a critical pair into two parts. The completion procedure modulo A without extended rules is expressed by the set $\mathcal{MWEC}$ of transition rules presented in Figure 18.5.

Here the ordering $l \rightarrow r >>_A g \rightarrow d$ is defined thanks to the proper encompassment ordering modulo A, defined by $g \sqsubseteq_A l$ iff $\exists \sigma, \sigma(g) \overset{*}{\longleftrightarrow}_A l_{|\omega}$ with $\omega \neq \epsilon$, or $\sigma(g) = l$ and σ is not a renaming.

The termination proof of the proof transformation relation $\Longrightarrow$ is then conditionned by the termination of the proper subterm ordering modulo A denoted $\lhd_A^{sub}$ and defined by $t \lhd_A^{sub} t'$ iff $t \overset{*}{\longleftrightarrow}_A t_0 \lhd^{sub} t'_0 \overset{*}{\longleftrightarrow}_A t'$ where $t_0 \lhd^{sub} t'_0$ means that t_0 is a proper subterm of t'_0.

Lemma 18.6 [Bac87] The proper encompassment ordering modulo A is well-founded iff the proper subterm ordering modulo A is well-founded.

If the proper subterm ordering modulo A is well-founded, the proof transformation relation $\Longrightarrow$ is well-founded.

Theorem 18.4 *Let A be a set of axioms with a finite complete unification algorithm, such that the proper subterm ordering modulo A is well-founded. Let $R_0 = \emptyset$, P_0 be a set of equalities and $>$ be an A-compatible reduction ordering. If $(P_0, R_0) \mapsto\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\rightarrow ....$ is a derivation such that*

- *$CP_A(R_\infty)$ is a subset of P_*,*
- *$CP_A(R_\infty, A)$ is a subset of R_*,*

Orient	$P \cup \{p = q\}, R$	$\mapsto\!\!\!\!\twoheadrightarrow$	$P, R \cup \{p \to q\}$ if $p > q$
Deduce	P, R	$\mapsto\!\!\!\!\twoheadrightarrow$	$P \cup \{p = q\}, R$ if $(p, q) \in CP_A(R)$
Extend	P, R	$\mapsto\!\!\!\!\twoheadrightarrow$	$P, R \cup \{l \to r\}$ if $(l, r) \in CP_A(R, A)$
Simplify	$P \cup \{p = q\}, R$	$\mapsto\!\!\!\!\twoheadrightarrow$	$P \cup \{p' = q\}, R$ if $p \to_{R,A} p'$
Delete	$P \cup \{p = q\}, R$	$\mapsto\!\!\!\!\twoheadrightarrow$	P, R if $p \overset{*}{\longleftrightarrow}_A q$
Compose	$P, R \cup \{l \to r\}$	$\mapsto\!\!\!\!\twoheadrightarrow$	$P, R \cup \{l \to r'\}$ if $r \to_{R,A} r'$
Collapse	$P, R \cup \{l \to r\}$	$\mapsto\!\!\!\!\twoheadrightarrow$	$P \cup \{l' = r\}, R$ if $l \to_{R,A}^{g \to d} l'$ & $l \to r >>_A g \to d$

Figure 18.5: Completion modulo A without extensions

- P_∞ *is empty,*

then (R_∞, A) *is Church-Rosser modulo* A *and* R_∞/A *is terminating.*

Proof: see [Bac87] □

18.4.3 Completion modulo A without extensions

A completion procedure modulo A with computation of A-critical pairs of rules on axioms is presented below in Figure 18.6.

```
PROCEDURE COMPLETION-MOD-CP (P,R,>)
IF P is not empty
THEN choose a pair (p,q) in P ; p':= R,A-normal form(p);
                                q':= R,A-normal form(q);
     CASE p' <-*->A q'  THEN  R := COMPLETION-MOD-CP(P-{(p,q)},R,>)
          p' > q' THEN  l:=p'; r:=q';
                        (P,R) := SIMPLIFICATION(P-{(p,q)},R,l -> r);
                        R := COMPLETION-MOD-CP(P,R U {l -> r},>)
          q' > p' THEN l:=q'; r:=p';
                       (P,R) := SIMPLIFICATION(P-{(p,q)},R,l -> r);
                       R := COMPLETION-MOD-CP(P,R U {l -> r},>)
          ELSE STOP with FAILURE
     END CASE;
ELSE IF all rules in R are marked
     THEN RETURN R; STOP with SUCCESS
     ELSE Choose an unmarked rule l -> r fairly;
          (P,R) := A-CRITICAL-PAIRS (l -> r, R, A);
          Mark the rule l -> r in R;
          R := COMPLETION-MOD-CP(P,R,>)
     END IF
END IF
END COMPLETION-MOD-CP
```

Figure 18.6: Completion modulo A without extensions

The procedure `A-CRITICAL-PAIRS` compute the A-critical pairs of the given rule $l \to r$ with other rules in R and add them to P. It also computes the A-critical pairs of the rule $l \to r$ with axioms in A and add them to R. It thus modifies both P and R.

Note that A-critical pairs of rules into axioms are never reduced before orientation. This may result in less simplifications, because the transition rules on simplification of rules are more restrictive than the transition rules on simplification of pairs. An alternative proposed in [JK86c] allows reduction of such A-critical pairs, but may need additional protections: more precisely, if (p,q) comes from an A-overlapping of a rule $l \rightarrow r$ on an axiom $g \rightarrow d$ and if the right-hand side of a critical pair (p,q) is R,A-reducible to q' at the outermost position (and not only R-reducible) using a rule $l' \rightarrow r'$, then the rule $l' \rightarrow r'$ must be *protected for coherence of* $l \rightarrow r$. This is a generalization of the notion of extension. Then the pair (p,q') is added to the set of pairs P.

18.5 General completion modulo A

Rewriting and completion modulo A based on equational matching and unification are very often very expensive. The idea is to use axioms in A as less as possible and to take advantage of linearity of rules. Another advantage of this idea is to build a more general method for rewriting and completion and to get both Huet's method and Peterson and Stickel's method as instances of this general method proposed in [JK86c].

Definition 18.8 Let $R = L \cup N$ with L a set of left-linear rules. A term t rewrites modulo A to t' with the relation $R_A = L \cup N, A$, denoted $t \rightarrow_{R_A} t'$, iff there exist

- either $t \rightarrow_{N,A} t'$, i.e. there exist a rule $l \rightarrow r \in N$, a position ω in t and a substitution σ such that $t_{|\omega} \overset{*}{\longleftrightarrow}_A \sigma(l)$ and $t' = t[\omega \hookleftarrow \sigma(r)]$,
- or $t \rightarrow_L t'$, i.e. there exist a rule $l \rightarrow r \in L$, a position ω in t and a substitution σ such that $t_{|\omega} = \sigma(l)$ and $t' = t[\omega \hookleftarrow \sigma(r)]$

Note that $\rightarrow_R \;\subseteq\; \rightarrow_{R_A} \;\subseteq\; \rightarrow_{R/A}$.

Definition 18.9 A proof of $(t = t')$ is a *rewrite proof modulo A* for R_A iff $\exists t_1, t_1', t \overset{*}{\longrightarrow}_{R_A} t_1,\ t' \overset{*}{\longrightarrow}_{R_A} t_1'$ *and* $t_1 \overset{*}{\longleftrightarrow}_A t_1'$.

A *peak* is a proof $t_1 \leftarrow_L t \rightarrow_R t_2$ or $t_1 \leftarrow_{N,A} t \rightarrow_R t_2$ or the inverse of such proof. A *cliff* is a proof $t_1 \leftarrow_L t \longleftrightarrow_A t_2$ or $t_1 \leftarrow_{N,A} t \longleftrightarrow_A t_2$ or the inverse of such proof.

Elimination of peaks and cliffs draw on the techniques already outlined. In essence, it requires computation of critical pairs of rules in L on rules in R and axioms in $\vec{A}$ and of axioms in $\vec{A}$ on rules in L. It also requires computation of A-critical pairs of N on R and either extensions of rules in N or computation of A-critical pairs of N on $\vec{A}$.

18.5.1 Transition rules for completion modulo A

In formulating transition rules for a general equational completion, the set L of rewrite rules used for standard rewriting must be distinguished from the set N of other rules used for rewriting modulo A. Also since extensions as well as A-critical pairs computation are allowed, a set of protected rules EX is necessary. For short, R will denoted $L \cup N \cup EX$ and $\rightarrow_{R_A}$ will denote $\rightarrow_L \cup \rightarrow_{(N \cup EX),A}$.

Several sets of critical pairs have to be distinguished:
$CP(A,L)$, the set of critical pairs of A on L,
$CP(L,A)$, the set of critical pairs of L on A,
$CP(L,R)$, the set of critical pairs of L on $L \cup N \cup EX$,
$CP_A(N,R)$, the set of A-critical pairs of N on $L \cup N \cup EX$,
$CP_A(N,A)$, the set of A-critical pairs of N on A.
The A-*completion procedure* is expressed by the set of transition rules $\mathcal{MODC}$ given in Figure 18.7.

Theorem 18.5 *[JK86c] Let A be a set of axioms with a finite complete unification algorithm, such that the proper subterm ordering modulo A is well-founded. Let L_0, N_0, EX_0 be empty sets and P_0 be a set of equalities. If a derivation $(P_0, L_0, N_0, EX_0) \mapsto\!\!\!\!\rightarrow (P_1, L_1, N_1, EX_1) \mapsto\!\!\!\!\rightarrow \dots$ satisfies*

- $CP(L_\infty, A) \cup CP(A, L_\infty)$ *are subsets of* R_*
- $CP(L_\infty, R_\infty) \cup CP_A(N_\infty, R_\infty)$ *are subsets of* P_*
- *For each rule $l \rightarrow r$ in N_∞ and each equality $g \rightarrow d$ in $\vec{A}$ such that l A-overlaps g at some position ω, either the extended rule $g[l] \rightarrow g[r]$ is in EX^*, or all A-critical pairs of $l \rightarrow r$ with $g \rightarrow d$ at position ω are in R^*.*

- $P_\infty = \emptyset$

then $\rightarrow_{L_\infty \cup (N_\infty \cup EX_\infty),A}$ *is Church-Rosser modulo* A *and* $\rightarrow_{R_\infty/A}$ *is terminating.*

18.5.2 General completion procedure modulo A

A completion procedure modulo A with distinction between left-linear and non-left-linear rules and with a minimal computation of A-critical pairs of rules on axioms is presented in Figure 18.8.

The procedure `A-CRITICAL-PAIRS`, applied to the rule $l \rightarrow r$, computes

- if l is linear, the critical pairs of the rule $l \rightarrow r$ on axioms and on axioms on $l \rightarrow r$, added to R, and critical pairs of $l \rightarrow r$ on all other rules in $(L \cup N \cup EX)$ added to P.

- if l is not linear, the A-critical pairs of the rule $l \rightarrow r$ on any other rule in $(L \cup N \cup EX)$, added to P. Extended rules are added for $l \rightarrow r$, when the rule is added to R.

This procedure thus may modify both P and R. An alternative could be to compute the A-critical pairs of the rule with axioms in A and add them to R.

Example 18.9 In Example 7.9, the theory of boolean rings, where the conjunction $\otimes$ and the exclusive-or $\oplus$ are associative and commutative, is given by the following sets of rewrite rules and equalities, denoted BR/AC:

$$\begin{array}{rcl}
x \oplus 0 & \rightarrow & x \\
x \oplus x & \rightarrow & 0 \\
x \otimes 0 & \rightarrow & 0 \\
x \otimes 1 & \rightarrow & x \\
x \otimes x & \rightarrow & x \\
-x & \rightarrow & x \\
x \otimes (y \oplus z) & \rightarrow & (x \otimes y) \oplus (x \otimes z)
\end{array}$$

$$\begin{array}{rcl}
x \otimes y & = & y \otimes x \\
(x \otimes y) \otimes z & = & x \otimes (y \otimes z) \\
x \oplus y & = & y \oplus x \\
(x \oplus y) \oplus z & = & x \oplus (y \oplus z)
\end{array}$$

This class rewrite system is generated by a completion algorithm modulo associativity and commutativity from the initial set of axioms:

$$\begin{array}{rcl}
x \oplus 0 & = & x \\
x \oplus (-x) & = & 0 \\
x \otimes 1 & = & x \\
x \otimes x & = & x \\
x \otimes (y \oplus z) & = & (x \otimes y) \oplus (x \otimes z) \\
x \oplus x & = & 0
\end{array}$$

$$\begin{array}{rcl}
x \otimes y & = & y \otimes x \\
(x \otimes y) \otimes z & = & x \otimes (y \otimes z) \\
x \oplus y & = & y \oplus x \\
(x \oplus y) \oplus z & = & x \oplus (y \oplus z)
\end{array}$$

18.5.3 Reduced systems

Contrary to the standard completion procedure in which inter-reduction of rules is a conserved property, some interreduction are dropped during a general completion procedure modulo A. However a reduced complete system may be obtained a posteriori.

Definition 18.10 A set of rules R is *reduced* if for any rule $l \to r$ in R, the term l is not R_A-reducible by any other rule and r is not R_A-reducible.

The following theorem is given for simplicity for the case where R_A is R, A.

Theorem 18.6 *[JK86c] Let R be a finite set of rewrite rules and A be a set of axioms, such that the proper subterm ordering modulo A is well-founded. Assume that R_A is Church-Rosser modulo A and R/A is terminating. Let R' be the set of rules obtained from R by deleting any rule $l \to r$ such that there exists another rule $l' \to r'$ such that $l \overset{*}{\longleftrightarrow}_A \sigma(l')$ for some substitution σ. Then R'_A is Church-Rosser modulo A and R'/A is terminating modulo A.*

18.6 Comparison between different completion methods

Faced to these different methods, the question arises to determine which is the good one to chose. Even when two methods are available, one of them can be preferred for efficiency or termination reasons.

Example 18.10 For the classical example of abelian groups, distinguishing between left-linear and non-left-linear rules is much more efficient than computing all AC-critical pairs. Starting from

$$\begin{array}{rcl} x+y &=& y+x \\ (x+y)+z &=& x+(y+z) \end{array}$$

$$\begin{array}{rcl} x+0 &\to& x \\ x+(-x) &\to& 0 \end{array}$$

the following set of left-linear rewrite rules is generated

$$\begin{array}{rcl} x+0 &\to& x \\ 0+x &\to& x \\ --x &\to& x \\ -0 &\to& 0 \\ -(x+y) &\to& (-x)+(-y) \end{array}$$

and the set on non-left linear rules is

$$\begin{array}{rcl} x+(-x) &\to& 0 \\ x+((-x)+y) &\to& y. \end{array}$$

Example 18.11 Starting from

$$\begin{array}{rcl} 0+x &=& x \\ a+b &=& 0 \\ (x+a)+b &=& x \end{array}$$

with $+$ being associative and commutative, completion using extensions and rewriting modulo AC gives the following set of rewrite rules:

$$\begin{array}{rcl} 0+x &\to& x \\ z+(0+x) &\to& z+x \\ a+b &\to& 0 \\ z+(a+b) &\to& z+0 \\ (x+a)+b &\to& x \\ z+((x+a)+b) &\to& z+x \end{array}$$

However using Huet's method, the completion procedure with left-linear rules diverges and R_∞ contains the following left-linear rules:

$$\begin{array}{rcl}
0+x & \rightarrow & x \\
x+0 & \rightarrow & x \\
a+b & \rightarrow & 0 \\
b+a & \rightarrow & 0 \\
(x+a)+b & \rightarrow & x \\
a+(b+z) & \rightarrow & z \\
b+(a+z) & \rightarrow & z \\
(x+b)+a & \rightarrow & x \\
b+(x+a) & \rightarrow & x \\
(a+x)+b & \rightarrow & x \\
(b+z)+a & \rightarrow & z \\
a+(y+b) & \rightarrow & y \\
(x+a)+(b+z) & \rightarrow & x+z \\
((x+(z+a))+b) & \rightarrow & x+z \\
z+((x+a)+b) & \rightarrow & z+x \\
 & \ldots &
\end{array}$$

It is clear that in that case, the first completion method should be preferred.

Some other comparisons may be found in [KK86].

As a conclusion, let us summarize which simple criteria may be used in order to choose the completion method:

- For efficiency it is important to get a minimal amount of computation of A-critical pairs: this leads to separate left-linear rules and to systematically add extensions.
- For theories with infinite sets of A-unifiers, and where it can be proved that extensions can be finitely computed, adjunction of extensions is preferrable.
- If a maximal interreduction of rules is favored, computation of A-critical pairs with possibly setting protections must be chosen.
- Theories with axioms $g = x$ prohibit the method of computing A-critical pairs of rules on axioms.
- Last but not least, remind that A-completion methods need the termination of R/A.

18.7 Ground associative commutative theories

The problem considered now is the decision of the word problem in the ground term algebra built on symbols $\mathcal{F}$ where some function symbols are associative and commutative.

The first result needed to handle this specific case is the existence of a total AC-simplification ordering. The case of only one AC-symbol was known from a long time (Malcev 1958), but the general case has been solved only in [NR91a].

In [BL81], Lankford and Ballantyne designed a completion algorithm for finitely presented commutative semi-groups, thus handling the case of one binary AC-function and a finite number of constants. The general case of ground AC-completion is handled in [Mar91, NR91a]. From their results follows the fact that any ground AC-theory has a finite canonical system.

In addition, provided a fair strategy ensuring that all possible simplifications are done before deductions, the AC-completion procedure presented in [Mar91] always terminate. We do not develop this procedure here but rather prefer to give an example of application. This example also illustrates a technique useful in AC theories which consists of flattening terms, i.e. replacing a term by $f(f(x,y),z)$ by $f(x,y,z)$. Then rewriting on flattened terms using R and its extensions simulates class rewriting with the original R. Completion modulo AC can be done on flattened terms, as in the example below.

Example 18.12 [1]. The chameleons problem can be stated as follows:

When a red chameleon meets a green one, both become blue.

When a red chameleon meets a blue one, both become green.

When a green chameleon meets a blue one, both become red.

These statements are translated into the set R of three rewrite rules:

$$\begin{array}{ccccccc} r & + & g & \rightarrow & b & + & b \\ r & + & b & \rightarrow & g & + & g \\ g & + & b & \rightarrow & r & + & r \end{array}$$

where r, g, b are constants and the $+$ operator is AC.

Given 42 chameleons, 15 red, 14 green and 13 blue, can chameleons become all of the same colour at some time?

Solving this problem amounts to prove that the initial state $i = 15r + 14g + 13b$ cannot be rewritten to either $42r$, $42g$ or $42b$. The problem here is that R does not terminates, as shown by the looping rewriting sequence:

$$r + g + g \rightarrow b + b + g \rightarrow b + r + r \rightarrow g + g + r$$

In order to prove that $i \overset{*}{\longrightarrow}_R 42x$ is false, the method consists in using ground completion to build from R another set R' satisfying:

$$i =_R 42x \text{ iff } i \overset{*}{\longrightarrow}_{R'} v \overset{*}{\longleftarrow}_{R'} 42x$$

We need a reduction ordering $>$ on the term algebra built on the vocabulary $\{+, r, g, b\}$ which is total ($\forall s, t,\ s \geq t$ or $t \geq s$), AC-compatible and satisfies $s =_{AC} t$ iff $s \geq t$ and $t \geq s$.

We can choose a recursive path ordering on flattened terms where $+$ has multiset status, and is minimal in the precedence, for example $r > g > b > +$. Critical pairs are defined directly on flattened terms. For example:

$$g + g + g \longleftarrow_{r+b \rightarrow g+g} b + r + g \longleftarrow_{r+g \rightarrow b+b} b + b + b$$

Ground completion modulo AC works as follows:

$$\begin{array}{rcc} & (1) & r + g \rightarrow b + b \\ & (2) & r + b \rightarrow g + g \\ & (3) & r + r \rightarrow b + g \\ \text{Deduce}(1,2) & (4) & g + g + g \rightarrow b + b + b \\ \text{Deduce}(1,3) & (5) & r + b + b \rightarrow g + b + g \\ \text{Simplify}(5,2) & & (5) \text{ deleted} \\ \text{Deduce}(2,3) & (6) & r + g + g \rightarrow b + b + g \\ \text{Simplify}(6,1) & & (6) \text{ deleted} \\ \text{Deduce}(1,4) & (7) & r + b + b + b \rightarrow g + g + b + b \\ \text{Simplify}(7,2) & & (7) \text{ deleted} \end{array}$$

We eventually obtain the system R':

$$\begin{array}{rcl} r + g & \rightarrow & b + b \\ r + b & \rightarrow & g + g \\ r + r & \rightarrow & b + g \\ g + g + g & \rightarrow & b + b + b \end{array}$$

Using R', normal forms can be computed for the following terms:

- $15r + 14g + 13b \overset{*}{\longrightarrow}_{R'} g + 41b$
- $42r \overset{*}{\longrightarrow}_{R'} 42b$
- $42g \overset{*}{\longrightarrow}_{R'} 42b$
- $42b \overset{*}{\longrightarrow}_{R'} 42b$

This proves that chameleons can never be all of the same colour.

[1]This example was presented by C. Marché at the RTA Conference in 1991, to illustrate his result [Mar91]

18.8 Conclusion

Even with very small sets of equalities, it is quickly irrealistic to attempt to do a completion modulo "by hand". This is really a place where the computer must help. The difficulty is often due to the computation of complete sets of unifiers, but also to the indispensable use of extensions to achieve completeness.

The main problems encountered in implementing and using completion modulo A come from several difficulties: the orientation and termination proof requires the ordering to be compatible with the equivalence classes and Section 7.3 of Chapter 7 provides few results on the automation of this point. A complete unification procedure must be available for the theory A of axioms. The most successful example of application is associative-commutative theories. But even in this case there may be a huge amount of critical pairs for one superposition. Criteria for eliminating useless ones become then crucial and researches in this area is described in Section 19.3 of Chapter 19. All these difficulties give rise to yet active research.

The notion of rewriting modulo an equational theory has been generalized in [Mar93], to handle a larger class of axioms including identity, idempotency, Abelian group theory and commutative ring theory. In this approach the theory A must be given as a convergent set (possibly modulo AC) of rewrite rules S and a S-normalized rewrite relation is defined, in which any term is first S-normalized before being rewritten. An adadpted completion procedure is defined which generalizes the algorithms for computing standard bases of polynomial ideals of [Win89].

Orient1 $P \cup \{p = q\}, L, N, EX$
$\mapstochar\twoheadrightarrow$
$P, L \cup \{p \rightarrow q\}, N, EX$
if $p > q$ &p *linear*

Orient2 $P \cup \{p = q\}, L, N, EX$
$\mapstochar\twoheadrightarrow$
$P, L, N \cup \{p \rightarrow q\}, EX$
if $p > q$ &p $non - linear$

Deduce R P, L, N, EX
$\mapstochar\twoheadrightarrow$
$P \cup \{p = q\}, L, N, EX$
if $(p, q) \in CP_A(N, R) \cup CP(L, R)$

Deduce1 R-A P, L, N, EX
$\mapstochar\twoheadrightarrow$
$P, L \cup \{l \rightarrow r\}, N, EX$
if $(l, r) \in CP_A(N, A) \cup CP(A, L) \cup CP(L, A)$ &l *linear*

Deduce2 R-A P, L, N, EX
$\mapstochar\twoheadrightarrow$
$P, L, N \cup \{l \rightarrow r\}, EX$
if $(l, r) \in CP_A(N, A) \cup CP(A, L) \cup CP(L, A)$ &l $non - linear$

Extend P, L, N, EX
$\mapstochar\twoheadrightarrow$
$P, L, N, EX \cup \{l \rightarrow r\}$
if $(l \rightarrow r) \in EXT_A(N \cup EX)$

Simplify $P \cup \{p = q\}, L, N, EX$
$\mapstochar\twoheadrightarrow$
$P \cup \{p' = q\}, L, N, EX$
if $p \rightarrow_{R_A} p'$

Delete $P \cup \{p = q\}, L, N, EX$
$\mapstochar\twoheadrightarrow$
P, L, N, EX
if $p \overset{*}{\longleftrightarrow}_A q$

Compose1 $P, L \cup \{l \rightarrow r\}, N, EX$
$\mapstochar\twoheadrightarrow$
$P, L \cup \{l \rightarrow r'\}, N, EX$
if $r \rightarrow_{R_A} r'$

Compose2 $P, L, N \cup \{l \rightarrow r\}, EX$
$\mapstochar\twoheadrightarrow$
$P, L, N \cup \{l \rightarrow r'\}, EX$
if $r \rightarrow_{R_A} r'$

Compose3 $P, L, N, EX \cup \{l \rightarrow r\}$
$\mapstochar\twoheadrightarrow$
$P, L, N, EX \cup \{l \rightarrow r'\}$
if $r \rightarrow_{R_A} r'$

Collapse1 $P, L \cup \{l \rightarrow r\}, N, EX$
$\mapstochar\twoheadrightarrow$
$P \cup \{l' = r\}, L, N, EX$
if $l \rightarrow_{R_A}^{g \rightarrow d} l'$ & $l \sqsubseteq_A g$

Collapse2 $P, L, N \cup \{l \rightarrow r\}, EX$
$\mapstochar\twoheadrightarrow$
$P \cup \{l' = r\}, L, N, EX$
if $l \rightarrow_{R_A}^{g \rightarrow d} l'$ & $l \sqsubseteq_A g$

Figure 18.7: Completion modulo A

```
PROCEDURE COMPLETION-MOD-A (P,L,N,EX,>)
IF P is not empty
THEN choose a pair (p,q) in P ; p':= R_A-normal form(p);
                                q':= R_A-normal form(q);
     CASE p' <-*->A q'  THEN  R := COMPLETION-MOD-A(P-{(p,q)},L,N,EX,>)
          p' > q' THEN  l:=p'; r:=q';
                        (P,R) := SIMPLIFICATION(P-{(p,q)},L,N,EX,l -> r);
                        IF l is linear THEN
                        R := COMPLETION-MOD-A(P,L U {l -> r},N,EX,>)
                        ELSE
                        R := COMPLETION-MOD-A(P,L,N U {l -> r},
                                              EX U EXT(l -> r),>)
                        END IF
          q' > p' THEN  l:=q'; r:=p';
                        (P,R) := SIMPLIFICATION(P-{(p,q)},L,N,EX,l -> r);
                        IF l is linear THEN
                        R := COMPLETION-MOD-A(P,L U {l -> r},N,EX,>)
                        ELSE
                        R := COMPLETION-MOD-A(P,L,N U {l -> r},
                                              EX U EXT(l -> r),>)
                        END IF
          ELSE STOP with FAILURE
     END CASE;
ELSE IF all rules in R are marked
     THEN RETURN R; STOP with SUCCESS
     ELSE Choose an unmarked rule l -> r fairly;
          (P,R) := A-CRITICAL-PAIRS (l -> r, L, N, A);
          Mark the rule l -> r in R;
          R := COMPLETION-MOD-A(P,L,N,EX,>)
     END IF
END IF
END COMPLETION-MOD-CP
```

Figure 18.8: General completion modulo A

Chapter 19

Ordered completion modulo a set of equalities

19.1 Introduction

For proving difficult mathematical theorems automatically in first-order logic with equality, associative-commutative theories are really necessary. But this is the kind of theories for which ordered completion often fails to terminates. This was the motivation for building these axioms in the deduction process. By combining techniques from ordered completion and completion in equivalence classes, a powerful completion process is obtained which allows building convergent class rewrite systems in some cases and always provides a refutationally complete theorem prover.

The most impressive results in automated deduction based on rewrite methods have been obtained with ordered completion modulo A and this will be illustrated in this chapter. But theorem provers such as SbReve and RRL also use powerful simplification mechanisms especially cancellation laws that considerably prune the search space of the theorem prover. In addition, in order to avoid useless computations, critical pair criteria are applied. All these refinements do contribute to the efficiency and realistic use of an equational prover.

19.2 Ordered completion modulo A

The ordered completion often fails to terminate in presence of associativity and commutativity equalities. So a natural idea is to combine ordered rewriting with class rewriting and to define an ordered completion procedure modulo a set of axioms A.

In the whole chapter, it is assumed that A is any set of axioms with decidable unification, matching and word problems and $\overset{*}{\longleftrightarrow}_A$ is the generated congruence relation on $\mathcal{T}(\Sigma, \mathcal{X})$. Also $>$ is a reduction ordering A-compatible ($\overset{*}{\longleftrightarrow}_A \circ > \circ \overset{*}{\longleftrightarrow}_A \subseteq >$), which is moreover (or can be extended into) a well-founded and total ordering on A-equivalence classes of ground terms (i.e. for all ground terms s and t, either $s \overset{*}{\longleftrightarrow}_A t$, or $s > t$ or $t > s$). Such orderings have been described for associative-commutative axioms by [NR91b, RN93].

19.2.1 Ordered critical pairs modulo A

Assume given an ordered class rewrite system $(E/A, >)$, defined by a set of axioms A, a set of equalities E and an A-compatible reduction ordering $>$. Remind that a term t $(E, A, >)$-rewrites to a term t', which is denoted by $t \rightarrow_{E,A,>} t'$ if there exist an equality $(l = r)$ of E, a position ω in t, a substitution σ, satisfying $t_{|\omega} \overset{*}{\longleftrightarrow}_A \sigma(l)$ and $\sigma(l) > \sigma(r)$, such that $t' = t[\omega \hookleftarrow \sigma(r)]$.

The relation $\rightarrow_{E,A,>}$ is ground Church-Rosser modulo A with respect to $>$ if on ground terms, $\overset{*}{\longleftrightarrow}_{E \cup A} \subseteq \overset{*}{\longrightarrow}_{E,A,>} \overset{*}{\longleftrightarrow}_A \overset{*}{\longleftarrow}_{E,A,>}$.

As for class rewriting, the ground Church-Rosser property modulo A expresses the existence of rewrite proof modulo A on ground terms.

In order to check local coherence and confluence properties of the relation $(E, A, >)$ on the sets E and A, or in a dual view, to obtain rewrite proofs by transforming peaks and cliffs, the usual notion of critical pairs extends to take into account A-unification and the reduction ordering.

Definition 19.1 Let $(g = d)$ and $(l = r)$ be two equalities in E with disjoint sets of variables. If there exists a position ω in g such that $g_{|\omega}$ is not a variable, $g_{|\omega}$ and l are A-unifiable with an A-unifier ψ in a complete set of A-unifiers, and if in addition $\psi(d) \not> \psi(g)$ and $\psi(r) \not> \psi(l)$, then $(\psi(g[r]_\omega) = \psi(d))$ is an *ordered critical pair modulo* A of $(l = r)$ into $(g = d)$.

Notation: The set of ordered A-critical pairs of E is denoted $OCP_A(E)$.

The extension of an equality is defined in a way similar to rewrite rules: Beyong critical pairs between equalities in E, for ensuring coherence, extended equalities are needed.

Definition 19.2 Let $(g = d)$ be an axiom in A and $(l = r)$ an equality in E with disjoint sets of variables. If there exists a position ω in g such that $g_{|\omega}$ is not a variable, $g_{|\omega}$ and l are A-unifiable with an A-unifier ψ in a complete set of A-unifiers, such that $\psi(r) \not> \psi(l)$, then $(g[l]_\omega = g[r]_\omega)$ is the *extended equality* of $(l = r)$ with respect to $(g = d)$.

In general extensions of extended equalities have also to be recursively computed. All these extended equalities are gathered in a set denoted $EXT_A(l = r)$. For a set E of equalities, $EXT_A(E)$ denotes the set of all extensions (recursively computed) of equalities in E with respect to A and E^{ext} denotes $E \cup EXT_A(E)$, i.e. the saturation of E under adjunction of extended equalities.

In general these extended equalities have to be recursively extended. However in the special case of A being only associativity and commutativity (AC) axioms, only a first level of extended equalities is needed.

19.2.2 Transition rules for ordered completion modulo A

As in completion modulo A, extensions are protected, that is an extension is never simplified and can only disappear if the initial rule itself disappears. In practice, a new set of protected equalities EX is introduced that contains extensions. Let P be a set of equalities and $>$ an A-compatible reduction ordering. The ordered completion procedure nodulo A is expressed by the set of transition rules set $\mathcal{MU}$ in Figure 19.1. For intereduction of equalities an ordering on equalities is needed. Let $>>$ be defined by $(p = q) >> (g = d)$ if $p \sqsupset g$ or $p \equiv g$ and $q > d$ in the given A-compatible reduction ordering.

Deduce P, EX
$\Vdash\!\!\twoheadrightarrow$
$P \cup \{p = q\}, EX$
if $(p, q) \in OCP_A(P)$

Extend P, EX
$\Vdash\!\!\twoheadrightarrow$
$P, EX \cup \{p = q\}$
if $(p, q) \in EXT_A(P)$

Delete $P \cup \{p = q\}, EX \cup EXT_A(p = q)$
$\Vdash\!\!\twoheadrightarrow$
P, EX
if $p \overset{*}{\longleftrightarrow}_A q$

Collapse $P \cup \{p = q\}, EX \cup EXT_A(p = q)$
$\Vdash\!\!\twoheadrightarrow$
$P \cup \{p' = q\}, EX \cup EXT_A(p' = q)$
if $(p \rightarrow^{g=d}_{P,A,>} p'$ & $p = q >> g = d$

Figure 19.1: Ordered completion modulo A

Let P_*, P_∞ be defined as previously and let EX_* be the set of all generated protected equalities, EX_∞ be the set of persisting protected equalities:

$$EX_\infty = \bigcup_i \bigcap_{j>i} EX_j.$$

This transition system is evidently sound, since the class of provable theorems is unchanged by any of these transitions.

Deduce-Eq-Eq $E \cup \{g = d,\ l = r\}$
$\mapstochar\twoheadrightarrow$
$E \cup \{g = d,\ l = r,\ \psi(g[r]_\omega) = \psi(d)\}$
if $\psi(g_{|\omega}) \overset{*}{\longleftrightarrow}_{AC} \psi(l), \psi(d) \not> \psi(g), \psi(r) \not> \psi(l)$

Deduce-Ext-Eq $E \cup \{g = d,\ l = r\}$
$\mapstochar\twoheadrightarrow$
$E \cup \{g = d,\ l = r,\ \psi(g[f(r, z)]_\omega) = \psi(d)\}$
if $\psi(g_{|\omega}) \overset{*}{\longleftrightarrow}_{AC} \psi(f(l, z)), \psi(d) \not> \psi(g), \psi(r) \not> \psi(l)$

Deduce-Ext-Ext $E \cup \{g = d,\ l = r\}$
$\mapstochar\twoheadrightarrow$
$E \cup \{g = d,\ l = r,\ \psi(f(d, z)) = \psi(f(r, z'))\}$
if $\psi(f(g, z)) \overset{*}{\longleftrightarrow}_{AC} \psi(f(l, z')), \psi(d) \not> \psi(g), \psi(r) \not> \psi(l)$

Delete $E \cup \{p = q\}$
$\mapstochar\twoheadrightarrow$
E
if $p \overset{*}{\longleftrightarrow}_{AC} q$

Simplify $E \cup \{p = q\}$
$\mapstochar\twoheadrightarrow$
$E \cup \{p' = q\}$
if $p \rightarrow^{(g=d),\sigma}_{E,AC,>} p'$ and $p > \sigma(g)$ or $q > \sigma(d)$

Figure 19.2: OCM: Ordered Completion Modulo AC

Theorem 19.1 *Let A be a set of axioms with a finite complete unification algorithm. Let $EX_0 = \emptyset$, P_0 be a set of equalities, and $>$ be a reduction ordering compatible with A that can be extended to a ground-total reduction ordering $\gg$. If $(P_0, EX_0) \mapstochar\twoheadrightarrow (P_1, EX_1) \mapstochar\twoheadrightarrow \ldots.$ is a derivation such that*

- *$OCP_A(P_\infty)$ is a subset of P_*,*
- *$EXT_A(P_\infty)$ is a subset of EX_*,*

then $(P_\infty \cup EX_\infty, A, >)$ is Church-Rosser modulo A w.r.t. $\gg$ on ground terms.

Proof: The proof is a generalization of the proofs of ordered completion and completion modulo A with extensions. □

19.2.3 The special case of associativity and commutativity

Let us focus on the special case where A is composed of associativity and commutativity axioms only. Then extensions can be systematically built: for each $(l = r) \in E$, such that the top symbol f in l is associative and commutative, the extended equality is $(f(l, z) = f(r, z))$ where z is a new variable (distinct from variables in l and r). From now on, an equality $(f(l, z) = f(r, z))$ implicitly denotes an extended equality of $(l = r)$ where the top symbol of l is an AC-operator f. For such l, it is sufficient to compute critical pairs at positions ω that are not imediately below another f. The rest of this section restricts to the case where $A = AC$ and describes a completion process for these specific axioms, in which the computation of extended equalities is incorporated in the rules.

Given the set OCM of rules for Ordered Completion Modulo given in Figure 19.2, a derivation is *fair* if any equality obtained from E_∞ by applying any rule **Deduce-Eq-Eq**, **Deduce-Ext-Eq**, **Deduce-Ext-Ext**, is included in $\bigcup_{i \geq 0} E_i$.

Theorem 19.2 *Let E_0 be a set of equalities, and $>$ be a well-founded reduction ordering total on AC-equivalence classes of ground terms. If $E_0 \mapstochar\twoheadrightarrow E_1 \mapstochar\twoheadrightarrow \ldots.$ is a fair derivation, then $(E_\infty^{ext}, AC, >)$ is Church-Rosser modulo AC on ground terms.*

19.2.4 Refutational proofs

Ordered completion modulo A can also be adapted to act as a refutational theorem prover.

Let E be a set of equalities and $(t = t')$ an equational theorem to be proved in the theory described by E, with $t, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Assume that t_0 and t'_0 are the skolemized versions of respectively t and t', that is terms whose variables are now considered as a set $\mathcal{H}$ of new constants disjoint from $\mathcal{F}$. Instead of introducing new operators T, F and eq, we rather introduce here disequalities and more specifically the disequality $(t_0 \neq t'0)$. We then extend the notion of ordered critical pairs between two equalities to the notion of ordered superposition of an equality into a disequality, producing then a new disequality.

Definition 19.3 Let $(g \neq d)$ a disequality in D and $(l = r)$ an equality in E with disjoint sets of variables. If there exists a position ω in g such that $g_{|\omega}$ is not a variable, $g_{|\omega}$ and l are A-unifiable with an A-unifier ψ in a complete set of A-unifiers, and if in addition $\psi(d) \not> \psi(g)$ and $\psi(r) \not> \psi(l)$, then $(\psi(g[\omega \hookleftarrow r]) \neq \psi(d))$ is an *ordered critical disequality modulo* A of $(l = r)$ into $(g \neq d)$.

The previous set OCM is modified by adding a set of disequalities D which is not modified by the previous rules but is transformed by additional rules given in Figure 19.3.

Deduce-Eq-Eq $E \cup \{g = d,\ l = r\}, D$
$\mapsto\!\!\!\twoheadrightarrow$
$E \cup \{g = d,\ l = r,\ \psi(g[r]_\omega) = \psi(d)\}, D$
if $\psi(g_{|\omega}) \xleftrightarrow{*}_{AC} \psi(l), \psi(d) \not> \psi(g), \psi(r) \not> \psi(l)$

Deduce-Ext-Eq $E \cup \{g = d,\ l = r\}, D$
$\mapsto\!\!\!\twoheadrightarrow$
$E \cup \{g = d,\ l = r,\ \psi(g[f(r, z)]_\omega) = \psi(d)\}, D$
if $\psi(g_{|\omega}) \xleftrightarrow{*}_{AC} \psi(f(l, z)), \psi(d) \not> \psi(g), \psi(r) \not> \psi(l)$

Deduce-Ext-Ext $E \cup \{g = d,\ l = r\}, D$
$\mapsto\!\!\!\twoheadrightarrow$
$E \cup \{g = d,\ l = r,\ \psi(f(d, z)) = \psi(f(r, z'))\}, D$
if $\psi(f(g, z)) \xleftrightarrow{*}_{AC} \psi(f(l, z')), \psi(d) \not> \psi(g), \psi(r) \not> \psi(l)$

Deduce-Eq-DEq $(E \cup \{l = r\}, D \cup \{g \neq d\})$
$\mapsto\!\!\!\twoheadrightarrow$
$(E \cup \{l = r\}, D \cup \{g \neq d,\ \psi(g[r]_\omega) \neq \psi(d)\})$
if $\psi(g_{|\omega}) \xleftrightarrow{*}_{AC} \psi(l), \psi(d) \not> \psi(g), \psi(r) \not> \psi(l)$

Deduce-Ext-DEq $(E \cup \{l = r\}, D \cup \{g \neq d\})$
$\mapsto\!\!\!\twoheadrightarrow$
$(E \cup \{l = r\}, D \cup \{g \neq d,\ \psi(g[f(r, z)]_\omega) \neq \psi(d)\})$
if $\psi(g_{|\omega}) \xleftrightarrow{*}_{AC} \psi(f(l, z)), \psi(d) \not> \psi(g), \psi(r) \not> \psi(l)$

Delete $E \cup \{p = q\}, D$
$\mapsto\!\!\!\twoheadrightarrow$
E, D
if $p \xleftrightarrow{*}_{AC} q$

Simplify $E \cup \{p = q\}, D$
$\mapsto\!\!\!\twoheadrightarrow$
$E \cup \{p' = q\}, D$
if $p \rightarrow^{(g=d),\sigma}_{E,AC,>} p'$ and $p > \sigma(g)$ or $q > \sigma(d)$

Figure 19.3: Refutational Ordered Completion Modulo AC

Let $ROCM$ denote the whole set of rules. A derivation $(E_0, D_0) \mapsto\!\!\!\twoheadrightarrow (E_1, D_1) \mapsto\!\!\!\twoheadrightarrow \dots$ with $ROCM$, is *fair* if any equality and disequality obtained from persisting equalities and disequalities (E_∞, D_∞) by applying any rule **Deduce-Eq-Eq**, **Deduce-Ext-Eq**, **Deduce-Ext-Ext**, **Deduce-Eq-DEq**, **Deduce-Ext-DEq**, is included in $\bigcup_i (E_i \cup D_i)$.

Definition 19.4 A *refutation* is a fair derivation $(E_0, D_0) \mapsto\!\!\!\twoheadrightarrow (E_1, D_1) \mapsto\!\!\!\twoheadrightarrow \dots$ for which $\bigcup_i D_i$ contains a disequality $(u \neq v)$ with $u \xleftrightarrow{*}_{AC} v$.

Refutational completeness of $ROCM$ is expressed by the next result, which is a consequence of Theorem 1 in [BG93].

Theorem 19.3 *Let E be a set of equalities, and $>$ be a reduction ordering compatible with AC total on AC-equivalence classes of ground terms. Then the equality $(t = t')$ is valid in $E \cup AC$ (i.e. $t \overset{*}{\longleftrightarrow}_{E \cup AC} t'$) iff ROCM generates a refutation from $E_0 = E \cup \{t_0 \neq t'_0\}$, where $(t_0 \neq t'_0)$ is the skolemized negation of $(t = t')$.*

In computational experiments as those described in the next section, the efficiency of the completion process can be greatly improved by using cancellation laws for some function symbols. A simple use of these laws is the application of right regularity of a constructor symbol c to deduce from $c(t_1, t) = c(t_2, t)$ the equality $t_1 = t_2$.

Definition 19.5 A function f is *right-cancellable* if it satisfies the right-cancellation law:
$\forall x, y, z,\ f(x, y) = f(z, y) \Rightarrow x = z.$
A function f is *left-cancellable* if it satisfies the left-cancellation law:
$\forall x, y, z,\ f(x, y) = f(x, z) \Rightarrow y = z.$

The right-cancellation laws can be implemented as transition rules [HRS87], as described in Figure 19.4.

Cancellation1
$P \cup \{f(s_1, s_2) = f(t_1, t_2)\}$
$\mapsto\!\!\!\rightarrow$
$P \cup \{f(s_1, s_2) = f(t_1, t_2), \sigma(s_1) = \sigma(t_1)\}$
if $\sigma(s_2) = \sigma(t_2)$
Cancellation2
$P \cup \{f(s_1, s_2) = y\}$
$\mapsto\!\!\!\rightarrow$
$P \cup \{f(s_1, s_2) = y, \sigma(s_1) = x\}$
if $\sigma(y) = f(x, s_2)$ & $y \in V(s_1) - V(s_2)$
Cancellation3
$P \cup \{f(s_1, s_2) = s, f(t_1, t_2) = t\}$
$\mapsto\!\!\!\rightarrow$
$P \cup \{f(s_1, s_2) = s, f(t_1, t_2) = t, \sigma(s_1) = \sigma(t_1)\}$
if $\sigma(s_2) = \sigma(t_2)$ & $\sigma(s) = \sigma(t)$
Cancellation4
$P \cup \{f(u, s) = f(u, t)\}$
$\mapsto\!\!\!\rightarrow$
$P \cup \{s = t\}$

Figure 19.4: Cancellation Laws

19.2.5 Experiments

To illustrate the ordered completion modulo A technique, we present two significant examples.

Example 19.1 Moufang's Identities in Alternative Rings:
Alternative rings are non-associative rings that satisfy the following set of axioms.

$$
\begin{array}{rcll}
0 + x & = & x & (19.1)\\
0 * x & = & 0 & (19.2)\\
x * 0 & = & 0 & (19.3)\\
i(x) + x & = & 0 & (19.4)\\
i(x + y) & = & i(x) + i(y) & (19.5)\\
i(i(x)) & = & x & (19.6)\\
x * (y + z) & = & (x * y) + (x * z) & (19.7)\\
(x + y) * z & = & (x * z) + (y * z) & (19.8)\\
(x * y) * y & = & x * (y * y) & (19.9)
\end{array}
$$

$$\begin{aligned}
(x*x)*y &= x*(x*y) &(19.10)\\
i(x)*y &= i(x*y) &(19.11)\\
x*i(y) &= i(x*y) &(19.12)\\
i(0) &= 0 &(19.13)\\
(x+y)+z &= x+(y+z) &(19.14)\\
x+y &= y+x &(19.15)
\end{aligned}$$

The identities below have been proved by R. Moufang, in 1933.

$$\begin{aligned}
(x*y)*x &= x*(y*x)\\
x*((y*z)*x) &= (x*(y*z))*x\\
x*(y*(x*z)) &= ((x*y)*x)*z\\
((z*x)*y)*x &= z*(x*(y*x))\\
(x*y)*(z*x) &= (x*(y*z))*x
\end{aligned}$$

A first computer proof has been reported in 1990 by S.Anantharaman and J.Hsiang in [AH90] using the system SBREVE. Let us briefly look at the proof of the first identity $\forall x, y, z,\ (y*x)*y = y*(x*y)$.

It is first negated then skolemized, so the following equality is added to the set of equalities defining alternative rings: $((cy*cx)*cy) \neq (cy*(cx*cy))$.

Among others a critical pair between 7 and 9 is computed:
$x*((y+z)*(y+z)) = ((x*(y+z))*y) + ((x*(y+z))*z)$

It is simplified by 7,8,9 to produce:
$x*(y*y) + x*(z*z) + x*(y*z) + x*(z*y) = x*(y*y) + x*(z*z) + (x*y)*z + (x*z)*y$

Regularity of + is used thanks to the application of a cancellation transition rule built-in in the system. This produces an ordered equality
$((x*y)*z) + ((x*z)*y) \to (x*(y*z)) + (x*(z*y))$

Then a new critical pair is computed between this last rule and the extended rule $i(x)+x+u \to u$:
$(x*(y*z)) + (x*(z*y)) + i((x*z)*y) = (x*y)*z$

The goal equality $((cy*cx)*cy) \neq (cy*(cx*cy))$ can now be simplified into:
$cy*(cx*cy) \neq cy*(cx*cy)$
using the orientable instance
$(cy*cx)*cy \to (cy*(cx*cy)) + (cy*(cy*cx)) + i((cy*cy)*cx)$.

To perform this proof, the system has computed 67 critical pairs and 15 ordered equalities.

Example 19.2 Ring Commutativity Problems
An associative ring is defined by the following set of axioms $RING$:

$$\begin{aligned}
x+y &= y+x\\
(x+y)+z &= x+(y+z)\\
x+0 &= x\\
x+i(x) &= 0\\
i(x+y) &= i(x)+i(y)\\
i(0) &= 0\\
i(i(x)) &= x\\
(x*y)*y &= x*(y*y)\\
x*(y+z) &= (x*y)+(x*z)\\
(x+y)*z &= (x*z)+(y*z)\\
0*x &= 0\\
x*0 &= 0\\
i(x)*y &= i(x*y)\\
x*i(y) &= i(x*y)
\end{aligned}$$

In 1945 the mathematician Jacobson proved that the next theorem holds in associative rings:

$$(\forall x, \exists n > 1, x^n = x) \Rightarrow \forall x, y, (x*y = y*x)$$

As such this theorem has not been proved automatically up to now, not even checked, but weaker versions have been proved using rewrite techniques. These are instances for specific n of the next theorem:

$$(\forall x, x^n = x) \Rightarrow \forall x, y, (x * y = y * x)$$

The case $n = 2$ is rather easy: Adding to the set of axioms $RING$ the equality $x * x = x$ and running completion modulo associativity and commutativity results in a rewrite system for boolean rings and produces the equality $x * y = y * x$.

The case $n = 3$ was given as a challenge problem in his book [Wos88]. The first mechanical proof was done in 1981 by Veroff [Ver81] using the Argonne National Laboratory theorem prover based on resolution and paramodulation. In 1984, Stickel produced a proof using exclusively rewrite techniques [Sti84]. Adding to the set of axioms $RING$ the equality $x * x * x = x$ and running completion modulo associativity and commutativity produced the equality $x * y = y * x$ as well as a complete set of rewrite rules. The proof was reproduced and optimized in RRL using a careful selection of critical pairs [KZ91]. RRL proceeeds by refutation, that is in addition to the $RING$ equalities and to the hypothesis $x * x * x = x$, the skolemized negation of the is added: $a * b \neq b * a$.

In addition RRL has been able to prove the commutativity of associative rings in which every element satisfies $x^n = n$ for a large class of even numbers $n < 2^{50000}$. For instance for $n = 6$, the system first generates the equality $x * x = x$ and thus reduces the problem to the previous case $n = 2$.

19.3 Critical pairs criteria

The efficiency of completion depends primarily on the number of critical pairs generated. The problem is even more crucial with completion of class rewrite systems. Very often critical pairs may be many and several attempts have been made to avoid unnecessary computations. This is the motivation of critical pairs criteria.

A very general definition of redundancy is that an equality e is redundant if there exist equalities $e_1, \dots, e_n$ such that $\{e_1, \dots, e_n\} \models e$ and $e \succ e_1, \dots, e_n$ for some adapted well-founded ordering $\succ$.

Redundancy of a critical pair can often be checked by considering the structure of the associated peak. Assume given a reduction ordering $>$ and a set of rewrite rules R contained in $>$.

Definition 19.6 A *critical pair criterion CPC* is a mapping from pairs (P, R) to subsets of critical pairs $CP(R)$, that is $CPC(P, R) \subseteq CP(R)$ and $CPC(P, R)$ are redundant critical pairs.

A critical pair criterion is *sound* if the check of convergence can be restricted to critical pairs in $CP(R) - CPC(P, R)$. In other words, a sound criterion provides a characterization of the Church-Rosser property.

Definition 19.7 A critical pair criteria CPC is *sound* w.r.t. a reduction ordering $>$ if whenever R is included into $>$ and is convergent for $CP(R) - CPC(R)$, R is Church-Rosser.

A lot of critical pair criteria have been proposed in the literature [BD86b, Küc85, Win83, KMN85, ZK89] following pioneer work of Buchberger [Buc79]. For instance, the *connectedness criterion* given in [Win83] allows ignoring any critical pair (p, q) derived from an overlap $t' \leftarrow_R t \rightarrow_R t''$ such that there exists a proof $t' \xleftrightarrow{*}_R t''$, each term of which is derivable from t using $\xrightarrow{+}_R$.

The main problem in designing a critical pair criterion is that it has to be compatible with the simplification mechanism employed by completion. The notion of fairness must be modified accordingly:

Definition 19.8 A derivation $(P_0, R_0) \mapsto\!\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\!\rightarrow \dots$ is *fair with respect to a critical pair criterion CPC* if whenever $\mathcal{P}$ is a critical overlap associated with a critical pair (p, q) in $CP(R^\infty) - \bigcup_i CPC(P_i, R_i)$, then there is a proof $\mathcal{P}'$ of $(p = q)$ in $\bigcup_i (P_i \cup R_i)$ such that $\mathcal{P} \overset{+}{\Longrightarrow} \mathcal{P}'$.

Definition 19.9 A critical pair criteria CPC is *correct* if every non-failing derivation that is fair with respect to CPC is also a fair derivation.

Every correct criterion is sound [BD88] but the converse does not hold.

Example 19.3 [BD89b] In [ZK89], the following criterion is proposed. $CPC(R)$ consists of all critical pairs of a rule $l \rightarrow r$ on a rule $g \rightarrow d$ at position ωv such that $\omega, v \neq \Lambda$ and the overlapped term $\sigma(g)$ is reducible at position ω. This criterion can be proved sound but is not correct. For instance the derivation

$$\begin{array}{rcl} (P_0, R_0) & = & (\emptyset, \{-0 \to 0, sq(-x+x) \to 0, -0+0 \to 0\}) \\ \mapsto\!\!\!\twoheadrightarrow & & \\ (P_1, R_1) & = & (\{0+0=0\}, \{-0 \to 0, sq(-x+x) \to 0\}) \\ \mapsto\!\!\!\twoheadrightarrow & & \\ (P_2, R_2) & & (\emptyset, \{-0 \to 0, sq(-x+x) \to 0, 0+0 \to 0\}) \end{array}$$

is fair w.r.t. this criterion, but the final rewrite system is not Church-Rosser, since the term $sq(-0+0)$ has two distinct normal forms 0 and $sq(0)$. The reason is that some critical pair, for instance $0 = sq(0+0)$ obtained by overlapping $-0 \to 0$ at position 1 in $sq(-x+x) \to 0$ is redundant w.r.t. the given criterion in (P_0, R_0) but not in (P_1, R_1) nor (P_2, R_2).

In [BD89b], a criterion is given, that subsumes all correct criteria. It is an abstract criterion that directly relies on the well-founded ordering on proofs, denoted by $\succ_c$, associated to a complexity measure c.

Definition 19.10 A peak $(t' \leftarrow_R t \to_R t'')$ is *composite* in P and R if there are proofs $\mathcal{P}_1, \ldots, \mathcal{P}_n$ in $P \cup R$ where $\mathcal{P}_i$ is a proof of $(u_{i-1} = u_i)$, such that $t' = u_0$, $t'' = u_{n+1}$, $t > u_i$ and $(t' \leftarrow_R t \to_R t'') \succ_c \mathcal{P}_i$ for all $i = 1, \ldots, n$.

A critical pair (p, q) is composite if its corresponding critical overlap is. $CCP(P, R)$ is the set of all composite critical pairs of R.

Proposition 19.1 [BD89b] The composite criterion is correct.

The compositeness criterion can be used to prove another result very useful to eliminate unnecessary critical pairs.

Definition 19.11 A critical pair $(\psi(g[\omega \hookleftarrow r]), \psi(d))$ of a rule $l \to r$ on $g \to d$ at position ω is *subsumed* by another critical pair $(\psi'(g[\omega' \hookleftarrow r']), \psi'(d))$ of a rule $l' \to r'$ on $g \to d$ at position ω' if $\omega' \neq \epsilon$ and there exists a substitution τ such that $\psi(x) = \tau(\psi'(x))$ for $x \in \mathcal{V}ar(g)$.

A set S of critical pairs is *complete* if each critical pair in $CP(R)$ is subsumed by a critical pair in S.

Proposition 19.2 [BD89b] Let R be a terminating rewrite system and S a complete subset of $CP(R)$. If all critical pairs in S are convergent, then R is Church-Rosser.

This result is especially interesting for rewrite rules that have multiple occurrence of a same subterm in the left-hand side. For instance if R contains a rule of the form $f(t, t) \to s$, for each critical pair obtained by superposing on one occurrence of t, there is a critical pair obtained by superposing on the other occurrence. As all these critical pairs subsume each other, it is sufficient to compute only one of them.

In the context of ring commutativity problems, effective criteria have been proposed [KZ91]. Let us consider two of them: restriction to prime (and blocked) superpositions and to general and symmetric superpositions.

- In the restriction to prime (and blocked) superpositions:
  the idea of these checks is to avoid the joinability test for critical pairs which are reducible.

  Consider for instance the two following rules

  $$\begin{array}{rcl} x+x+x+x+x+x & \to & 0 \\ (x'+y')*z' & \to & (x'*y')+(x'*z') \end{array}$$

  where $x+x+x+x+x+x$ and $x'+y'$ unify modulo AC with a complete set of unifiers composed of 125 unifiers. Among them, the unifer

  $$\begin{array}{rcl} x' & = & u+v+v+v+v+v+v \\ y' & = & u+u+u+u+u \\ x & = & u+v \end{array}$$

  is reducible by the first rule. It is useless to consider the critical pair corresponding to it. For this example, 94 out of 125 unifiers are eliminated with this criterion.

  However, all the unifiers have to be computed before checking their reducibility. It is of prime interest to be able to detect useless superpositions before the computation of unifiers.

- The second criterion corresponding to this last requirement takes advantage of symmetries in rules and subterms. For instance, consider the rule
$$(x*y*z)+(x*z*y)+(y*x*z)+(y*z*x)+(z*x*y)+(z*y*x) \rightarrow 0$$
that superposes with the rule $x*x*x \rightarrow x$. There is a superposition corresponding to each of the six product subterms of the first rule. It can easily be seen that variables x, y, z are symmetric in this rule, because of the associativity and commutativity of $+$. So it is enough to consider the superpositions due to unifying only one of the product terms with the left-hand side $x*x*x$ of the second rule.

In order to justify these examples and to give more precise results, some formalization is needed. A superposition in characterized by a 4-tuple $\langle \psi, g \rightarrow d, \omega, l \rightarrow r\rangle$, where ψ is a unifier (the most general one or a more general one) of the left-hand side of $l \rightarrow r$ with the non-variable subterm $g_{|\omega}$ of $g \rightarrow d$. This superposition is said joinable (for short) if the corresponding critical pair is joinable.

Definition 19.12 A superposition $\langle \psi, g \rightarrow d, \omega, l \rightarrow r\rangle$ of a rewrite system R is *composite* if there exists $v \neq \Lambda$ such that $\psi(g)_{|v}$ is R-reducible. A superposition is *prime* if it is not composite.

Example 19.4 Assume that R contains rewrite rules
$$\begin{array}{rcl} i(i(x)*y) & \rightarrow & i(y)*i(i(x)) \\ x*i(x) & \rightarrow & e \\ i(i(x)) & \rightarrow & x \end{array}$$
The critical overlap between the first two rules
$$i(i(i(x)))*i(i(x)) \leftarrow_R i(i(x)*i(i(x))) \rightarrow_R i(e)$$
is composite because the strict subterm $i(i(x))$ of $i(i(x)*i(i(x)))$ is R-reducible.

A special case of composite substitutions are blocked ones, first studied by [LB77a].

Definition 19.13 A superposition $\langle \psi, g \rightarrow d, \omega, l \rightarrow r\rangle$ of a rewrite system R is *blocked* if ψ is R-irreducible. A superposition is *unblocked* if it is not blocked.

Any composite superposition can be factored into two prime superpositions and only these ones are needed for establishing the Church-Rosser property of a terminating rewrite system:

Theorem 19.4 *[KMN88] A terminating rewrite system R is Church-Rosser iff every prime superposition of R is joinable.*

Definition 19.14 A superposition $\langle \alpha, g \rightarrow d, \omega_2, l_2 \rightarrow r_2\rangle$, is *more general than* a superposition $\langle \psi, g \rightarrow d, \omega_1, l_1 \rightarrow r_1\rangle$, using the same first rule, if α is more general than ψ (i.e. $\psi = \rho\alpha$).

Theorem 19.5 *A terminating rewrite system R is Church-Rosser iff for any superposition of R $\langle \psi, g \rightarrow d, \omega_1, l_1 \rightarrow r_1\rangle$, either it is joinable, or there exists a more general superposition $\langle \alpha, g \rightarrow d, \omega_2, l_2 \rightarrow r_2\rangle$, with ω_1 and ω_2 being disjoint positions in g.*

A symmetric renaming θ of an equality e is a permutation of variables in e such that $\theta(e) \overset{*}{\longleftrightarrow}_{AC} e$. Two subterms of e are symmetric if there exists a symmetric renaming θ of e such that $\theta(t_1) \overset{*}{\longleftrightarrow}_{AC} t_2$.

As a corollary of this theorem, if a left-hand side of a rule has identical of symmetric subterms at disjoint positions, it is sufficient to superpose other rules at one of these subterm positions.

These criteria are correct with respect to completion, which means that if a superposition has been identified as unnecessary at some step of a completion process, it will remain so in further steps. Moreover they can be combined inside a same completion procedure.

Further hints for efficient implementations can be found in [KZ91].

19.4 Conclusion

We got at this point a very sophisticated notion of rewriting, using built-in A-equalities and a reduction ordering. We are able to combine rewrite rules (modulo A) with ordered equality, provided an adequate notion of ordering. Completion for this kind of rewrite relation uses a combination of previously introduced concepts. However in practical implementations, this expressive power is balanced by crucial problems of efficiency. Faced to this question, current research is oriented now towards the concept of rewriting and completion with constraints. This formalism is powerful enough to take into account orderings and equations modulo A and to provide a solution to efficiency improvement as argued in Chapter 21.

Chapter 20

Conditional completion

20.1 Introduction

In Section 7.5 of Chapter 7, different notions for conditional rewriting have been studied, and the notion of *reductive* conditional rewrite systems was proposed, where conditions are smaller than left-hand sides, so that recursively evaluating conditions always terminate. The rewrite relation for reductive systems is terminating and decidable, when there are only a finite number of rules.

Once termination is obtained, the problem of confluence arises, as well as how to test it using an adequate notion of critical pairs. Here again the situation sensibly differs from the unconditional case, because local confluence is not in general implied by the joinability of critical pairs. Several works attacked this problem [Kap84, DOS87, Kap87, JW86, DP88, DO90].

As in the equational case, the Church-Rosser property is equivalent to confluence of the recursive conditional rewriting relation, and confluence is equivalent to local confluence. This last property is in turn equivalent to convergence of conditional critical pairs, provided syntactic restrictions of decreasingness. In [Kap87], the critical pairs lemma is proved for simplifying systems: a simplifying system is locally confluent iff there is a rewrite proof for each instance σ of a conditional critical pair $s = t$ if Γ such that $\sigma(\Gamma)$ holds. Other systems for which the critical pairs lemma holds are considered in [DO90]. Nevertheless confluence is only semi-decidable, on account of the semi-decidability of the satisfiability of conditions. Moreover, the problem of determining whether critical pairs always converge still remains.

Another problem is the design of a completion procedure for conditional rewrite systems.

In [Kap87], a process based on computation of conditional critical pairs, conditional simplification, narrowing for checking the satisfiability of conditions, is proposed but its implementation appeared as highly inefficient.

An approach based on contextual rewriting is implemented in Reveur-4 [BR87], a version of REVE for conditional rewriting. The idea is to use case splitting on the condition in order to partition the set of critical pairs into different subsets. It is then checked whether the contexts are unsatisfiable, or whether the terms are convergent without additional hypotheses [KR89b]. In this approach, the class of models is limited to the subclass of algebras where case reasoning is valid. In these models, the Boolean part is isomorphic to the classical two-element Boolean algebra [Bou90c, NO87]. However, contextual rewriting considers only restrictive conditional rules: the conditions need to be Boolean equations whose right-hand sides belong to the set $\{true, false\}$. Moreover, several additional hypotheses are required to force the models to be consistent extensions of the two-element Boolean algebra. Techniques for checking ground confluence by using case analysis are developed in [Bou90c].

In the case of recursive rewriting, the main problem is the treatment of non-reductive equalities in which the condition is more complex than the conclusion. An interesting approach to overcome this problem is to superpose rules on the condition in order to enumerate its solutions [KR87]. This process translates the non-reductive equality into a set of reductive rules. Unfortunately this set is infinite in general and additional techniques must be used to ensure termination of completion. In the system CEC, a procedure based on these ideas is implemented and described in [Gan91].

A quite different approach of the problem arises from the community of resolution theorem provers. A conditional equality or rewrite rule may be understood as an equational Horn clause built with the only equality predicate. Starting from a set of equational Horn clauses, the completion process is understood as a saturation process on these formulas, using a refutationally complete set of transition rules for first-order logic with equality. When it stops, the resulting set of formulas can be used to solve the word problem in the Horn theory, that is to decide whether $t = t'$ holds in the equality Herbrand models of the given set of equational Horn clauses. A conditional equality is used as a conditional rewrite rule when it is orientable and its conditions are smaller than the equality for some well-founded ordering, which ensures that conditions can be recursively checked. This approach generalizes ordered completion. Among several propositions built from this point of view, let us cite [Der90, KR87, BG91b]. This seems to be, at the time being, the most advance technique for completion of conditional rewrite systems.

20.2 Conditional critical pairs and local confluence

For terminating unconditional rewrite systems, the critical pairs lemma provides a test for confluence. For such systems with a finite number of rules, the joinability relation ($p \downarrow q$) is decidable. With conditional rewrite systems, the critical pair test does not guarantee confluence for terminating join systems.

Definition 20.1 Let $g \to d$ if Δ and $l \to r$ if Γ be two conditional rewrite rules with disjoint sets of variables, such that l and g overlap at position ω of $\mathcal{G}rd(g)$ with the most general unifier ψ. The *overlapped term* $\psi(g)$ produces the *conditional critical pair* $\psi(g[\omega \hookleftarrow r]) = \psi(d)$ if $\psi(\Delta \wedge \Gamma)$.

This conditional critical pair is *feasible* if $\psi(\Delta \wedge \Gamma)$ is R-unifiable.

A conditional critical pair $s = t$ if Γ is *trivial* if s and t are the same term.

A conditional critical pair $s = t$ if Γ is *joinable* if for any substitution σ such that $\sigma(s_i) \downarrow \sigma(t_i)$ for any $(s_i = t_i) \in \Gamma$, $\sigma(s) \downarrow \sigma(t)$.

Notation: Let $CCP(R)$ be the set of conditional critical pairs obtained from any two rules in R.

Example 20.1 [BK86] Consider the join conditional rewriting relation generated by the conditional rewrite system:

$$\begin{array}{rrcl} & f(a) & \to & a \\ x = f(x) \quad \text{if} & f(x) & \to & c \end{array}$$

This system has no critical pair, but is not locally confluent: Since $a \downarrow f(a)$, $f(a) \to c$ and $f(f(a)) \to f(c)$. Also $f(f(a)) \to c$. But c and $f(c)$ are irreducible.

Of course the difficulty relies of checking joinability since this property requires finding all substitutions that are solutions of the condition. Indeed trivial critical pairs are (trivially) joinable. If the condition can be proved unsatisfiable, the critical pair is also trivially joinable.

Example 20.2 [Kap87] Consider the finite conditional rewrite system R

$$\begin{array}{rrcl} (k(x) = c) \quad \text{if} & f(x) & \to & g(x) \\ (k(x) = c) \quad \text{if} & f(x) & \to & h(x) \\ & k(a) & \to & c \\ & k(b) & \to & c \\ & g(a) & \to & d \\ & g(b) & \to & e \\ & h(a) & \to & d \\ & h(b) & \to & e \end{array}$$

The system has a feasible conditional critical pair $g(x) = h(x)$ if $(k(x) = c)$, since $(x \mapsto a)$ and $(x \mapsto b)$ are two R-unifiers of $k(x)$ and c. This critical pair is joinable.

Coming back to the terminology of Section 7.5 of Chapter 7, let us review different critical pairs lemma that can be proved.

For the most general notion of natural conditional rewriting, the result was proved by Dershowitz and Plaisted:

Theorem 20.1 *[DP88] A terminating natural conditional rewriting relation is confluent iff every critical pair is joinable.*

This is ensured for instance if all conditional critical pairs are nonfeasible or trivial, as in the following example:

Example 20.3 [Kap87] This example is a specification of natural numbers with the "less-than" predicate $\leq$, of lists with constructors *nil* (the empty list) and *cons*, and of the membership predicate $\in$ for lists and the function *ins* that inserts an element e in a list l before the first element in l that is greater than e.

$$\begin{array}{rlcl}
 & 0 \leq 0 & \rightarrow & true \\
 & s(x) \leq 0 & \rightarrow & false \\
 & s(x) \leq s(y) & \rightarrow & x \leq y \\
 & x \leq x & \rightarrow & true \\
 & x \in nil & \rightarrow & false \\
 & x \in cons(x,s) & \rightarrow & true \\
(x \leq y) = false \ \ \mathsf{if} & x \in cons(y,s) & \rightarrow & x \in s \\
(y \leq x) = false \ \ \mathsf{if} & x \in cons(y,s) & \rightarrow & x \in s \\
 & ins(x,nil) & \rightarrow & cons(x,nil) \\
(x \leq y) = true \ \ \mathsf{if} & ins(x,cons(y,l) & \rightarrow & cons(x,cons(y,l)) \\
(x \leq y) = false \ \ \mathsf{if} & ins(x,cons(y,l) & \rightarrow & cons(y,cons(x,l))
\end{array}$$

But join conditional rewriting requires an additional hypothesis.

Example 20.4 [DOS87] There exists a noetherian, non-locally confluent join conditional rewriting relation associated to a system all of whose critical pairs are joinable:

$$\begin{array}{rlcl}
 & c & \rightarrow & k(f(a)) \\
 & c & \rightarrow & k(g(b)) \\
 & a & \rightarrow & b \\
 & h(x) & \rightarrow & k(x) \\
 & h(f(a)) & \rightarrow & c \\
h(f(x)) = k(g(b)) \ \ \mathsf{if} & f(x) & \rightarrow & g(x)
\end{array}$$

Though all four critical pairs are joinable, the term $f(a)$ has two normal forms $f(b)$ and $g(b)$.

Exercice 62 — Consider the system:

$$\begin{array}{rlcl}
x = f(x) \ \ \mathsf{if} & f(x) & \rightarrow & a \\
 & f(b) & \rightarrow & b
\end{array}$$

Prove that the join rewrite relation associated to this system is not locally confluent although it has no critical pair.
Answer: Its join rewriting relation is not locally confluent since

$$f(a) \leftarrow f(f(b)) \rightarrow a$$

but $f(a) \downarrow a$ does not hold, because this would require $f(a) \rightarrow a$ and thus $f(a) \downarrow a$. However, the system has no critical pair. [Klo90b]

In [Kap87] first, the critical pair lemma was extended to simplifying conditional rewrite systems. With a similar proof, the result holds for decreasing rewriting.

Theorem 20.2 *[DO90] Let R be a conditional rewrite system such that R^{join} is decreasing. Then R^{join} is confluent if every critical pair of R is joinable.*

This result may be extended to decreasing join rewriting relations with built-in predicates (see [DO90]). If we do not want to prove decreasingness, an alternative is possible.

Definition 20.2 An *overlay* conditional rewrite system is such that no left-hand side unifies with a non-variable subterm of any other left-hand side.

Here is an example of an overlay system.

Example 20.5 Consider the conditional rewrite system defining the "less-than" predicate on natural numbers:

$$\begin{array}{rlcl}
 & 0 \leq 0 & \rightarrow & true \\
 & s(x) \leq 0 & \rightarrow & false \\
 & s(x) \leq s(y) & \rightarrow & x \leq y \\
(x \leq y) = true \ \ \mathsf{if} & x \leq s(y) & \rightarrow & true
\end{array}$$

The only conditional critical pair is between the two last rules and both left-hand sides unify on top. The conditional critical pair is:

$$x \leq y = true \ \mathsf{if} \ s(x) \leq y \downarrow true.$$

Theorem 20.3 *[DOS87] Let R be an overlay conditional rewrite system such that R^{join} is terminating. Then R^{join} is confluent if every critical pair is joinable.*

It is important to note that interpreting Horn clauses as conditional rewrite rules with right-hand side *true*, leads to an overlay system since predicate symbols are never nested in the head of a clause. Furthermore all critical pairs are joinable since all right-hand sides are the same. This theorem also applies to pattern-directed functional languages in which defined functions may not be nested on left-hand sides.

The remaining difficulty is then to prove that the rewriting relation terminates. Adding a left-linearity hypothesis allows dropping the termination requirement. This result generalizes the confluence property of orthogonal rewrite systems.

Theorem 20.4 *[BK86] For any left-linear conditional rewrite system R without critical pairs, R^{norm} is confluent.*

As a conclusion of this section, let us consider an interesting application of conditional rewrite systems to combinatory logic.

Example 20.6 Combinatory logic can be enriched by adding new constants D and E as well as a new rule that can be seen as a test for syntactic identity. We get the following set of rules $CL - e$:

$$\begin{array}{rcl} Sxyz & \rightarrow & xz(yz) \\ Kxy & \rightarrow & x \\ Ix & \rightarrow & x \\ Dxx & \rightarrow & E. \end{array}$$

This system has been proved non-confluent in [Klo80]. However it has the unique normal form property, as proved in [KdV90], using the linearized system $CL - e^*$:

$$\begin{array}{rlcl} & Sxyz & \rightarrow & xz(yz) \\ & Kxy & \rightarrow & x \\ & Ix & \rightarrow & x \\ x = y \;\; \mathsf{if} & Dxy & \rightarrow & E \end{array}$$

The result stated in the previous example for $CL - e$ and its linearization $CL - e^*$ can be generalised:

Proposition 20.1 [KdV90] Let R a rewrite system and R^L its linearization. Assume that R^L is confluent. Then R has unique normal forms.

Another interesting property that can be deduced from these results is the next one:

Let us call a rewrite system R *strongly non-ambiguous* if after replacing each non-left-linear rewrite rule by its linearization, the resulting system has no critical pair.

Theorem 20.5 *[KdV90] Any strongly non-ambiguous rewrite system has unique normal forms.*

This property that may seem surprising comes from the fact that when R has no critical pair, then R^L is orthogonal and orthogonal conditional rewrite systems are confluent as stated in Theorem 20.4 [BK86].

20.3 Saturated sets of conditional equalities

We adopt in this section another point of view, by considering now conditional equalities as equational Horn clauses. So it is assumed that "=" is the only predicate which occurs within the encountered clauses. However the techniques developed in this section can be applied to general first-order clauses (see [BG91b].

In a theorem prover for equational logic, such as ordered completion, the expansion rule is the superposition of ordered equalities. In equational Horn clause logic, more expansion rules have to be considered: superposition, narrowing and reflection. Similar to the critical pairs lemma for ordered completion, lifting lemmas have to be stated for each of these expansion rules.

20.3.1 Superposition, narrowing and reflection

We describe in this section how to produce new conditional equalities by three inference rules called superposition, narrowing and reflection.

Let us consider the inference rules presented in Figure 20.1.

In order to limit the set of useful inferences, some ordering restrictions are imposed in the conditions of these rules. They are expressed thanks to an ordering on conditional equalities.

Superposition	$g = d$ if $\Delta,\ l = r$ if Γ
	$\mapsto\!\!\!\!\twoheadrightarrow$
	$\psi(g[\omega \hookleftarrow r]) = \psi(d)$ if $\psi(\Gamma \wedge \Delta)$
	if ψ $mgu(g_{\|\omega}, l)$
Narrowing	$g = d$ if $(\Delta \wedge s = t),\ l = r$ if Γ
	$\mapsto\!\!\!\!\twoheadrightarrow$
	$\psi(g) = \psi(d)$ if $\psi(\Gamma \wedge \Delta \wedge s[r]_\omega = t)$
	if ψ $mgu(s_{\|\omega}, l)$
Reflection	$g = d$ if $(\Delta \wedge s = t)$
	$\mapsto\!\!\!\!\twoheadrightarrow$
	$\psi(g) = \psi(d)$ if $\psi(\Delta)$
	if ψ $mgu(s, t)$

Figure 20.1: Inference rules for equational Horn logic

20.3.2 Ordering on conditional equalities

Let us recall the ordering introduced to define ordered instance of conditional equalities in Section 7.5 of Chapter 7.

Let $>$ be a reduction ordering on terms contained in some ordering total on ground terms, and $\geq$ be defined by $s \geq t$ if $s > t$ or $s = t$. The multiset extension $>^{mult}$ is an ordering on equalities denoted by $>_{\mathcal{E}}$.

This ordering extends to conditional equalities considered as multisets of equalities as follows: First add $\perp$ as a new symbol that satisfies for every term t, $t > \perp$. Then associate to each occurrence of the equality $s = t$, its complexity $c(s = t)$ which is

- the multiset $\{\{s\}, \{t\}\}$ if $s = t$ is the conclusion of C,

- the multiset $\{\{s, \perp\}, \{t, \perp\}\}$ if $s = t$ occurs in the condition of C, These complexities are multisets of multisets of terms that may be compared using $(>^{mult})^{mult}$.

The complexity of a conditional equality C, denoted by $c(C)$ is the multiset of complexities $c(s = t)$ for any $s = t \in C$. In other words, each conditional equality

$$C = (l = r \text{ if } \bigwedge_{i=1,\dots,n} s_i = t_i)$$

has the complexity

$$c(C) = \{\{\{s_1, \perp\}, \{t_1, \perp\}\}, ..., \{\{s_n, \perp\}, \{t_n, \perp\}\}, \{\{l\}, \{r\}\}\}.$$

Conditional equalities are now compared with the ordering $>_{\mathcal{C}}$, defined by

$$C >_{\mathcal{C}} C' \text{ if } c(C)((>^{mult})^{mult})^{mult} c(C').$$

$\geq_{\mathcal{C}}$ is defined by $C \geq_{\mathcal{C}} C'$ if $C >_{\mathcal{C}} C'$ or $C = C'$.

Example 20.7 Let $s > t > u$. Then $(s = t) >_{\mathcal{E}} (s = u)$, and $(s = u \Rightarrow a = b) >_{\mathcal{C}} (\ \Rightarrow s = t)$, since $\{s, \perp\}$ is greater w.r.t. $>^{mult}$ than both $\{s\}$ and $\{t\}$.

20.3.3 Critical pairs, narrowing and resolvent

Each of the previous inference rule can be applied on a set of conditional equalities with variables and schematizes application of inference rules on ground instances of these conditional equalities. This is expressed in lifting lemmas, similar to the critical pairs lemma.

Definition 20.3 Let $g = d$ if Δ and $l = r$ if Γ be two conditional equalities with disjoint sets of variables, such that l and g overlap at position ω of $\mathcal{G}rd(g)$ with the most general unifier ψ, $\psi(d) \not\geq \psi(g)$, $\psi(r) \not\geq \psi(l)$, $\forall(u_i = v_i) \in \Delta, \psi(u_i = v_i) \not\geq \psi(g = d)$, $\forall(s_i = t_i) \in \Gamma, \psi(s_i = t_i) \not\geq \psi(l = r)$. The *overlapped term* $\psi(g)$ produces the *conditional ordered critical pair*

$$\psi(g[\omega \hookleftarrow r]) = \psi(d) \text{ if } \psi(\Delta \wedge \Gamma).$$

Let $SUP(P)$ be the set of conditional ordered critical pairs obtained from any two equalities in P.

Lemma 20.1 [BG91b] Let $D = (g = d \text{ if } \Delta)$ and $C = (l = r \text{ if } \Gamma)$ be two conditional equalities with disjoint sets of variables, and let $\sigma(D) = (\sigma(g) = \sigma(d) \text{ if } \sigma(\Delta))$ and $\sigma(C) = (\sigma(l) = \sigma(r) \text{ if } \sigma(\Gamma))$ be ground instances such that $\sigma(D) >_{\mathcal{C}} \sigma(C)$, $\sigma(C)$ ordered and $\forall x \in \mathcal{V}ar(D), \sigma(x)$ is irreducible by $l = r$. Then any superposition inference from $\sigma(C)$ on $\sigma(D)$ produces a new formula which is a ground instance of a conditional ordered critical pair in $SUP(P)$.

Proof: The formula produced by the superposition inference rule from $\sigma(C)$ and $\sigma(D)$ is

$$\psi(\sigma(g)[\omega \hookleftarrow \sigma(r)]) = \psi(\sigma(d)) \text{ if } \psi(\sigma(\Delta) \wedge \sigma(\Gamma))$$

where ψ is the most general unifier of the two terms $\sigma(g)_{|\omega}$ and $\sigma(l)$. Since these terms are ground, ψ is the identity and $\sigma(g)_{|\omega} = \sigma(l)$. Moreover, since $\forall x \in \mathcal{V}ar(D), \sigma(x)$ is irreducible by $l = r$, ω is a position in $\mathcal{G}rd(g)$ and $\sigma(g_{|\omega}) = \sigma(l)$. This means that σ is a unifier of $g_{|\omega}$ and l, greater than their most general unifier, say α. The conditions $\alpha(d) \not\geq \alpha(g)$, $\alpha(r) \not\geq \alpha(l)$, $\forall(u_i = v_i) \in \Delta, \alpha(u_i = v_i) \not\geq \alpha(g = d)$, $\forall(s_i = t_i) \in \Gamma, \alpha(s_i = t_i) \not\geq \alpha(l = r)$ all hold otherwise this would contradict one of the facts that $\sigma(C) = \beta\alpha(C)$ or $\sigma(D) = \beta\alpha(D)$ is ordered. So there exists a conditional ordered critical pair

$$\alpha(g[\omega \hookleftarrow r]) = \alpha(d) \text{ if } \alpha(\Delta \wedge \Gamma)$$

which has as ground instance by β the initial formula. □

In addition to compute ordered conditional critical pairs, we also need to partially solve conditions of equalities when they are more complex than the conclusion. This motivates the introduction of the set of *conditional narrowings* computed from P, in which the left-hand side l of a conditional equality is unified with a non-variable subterm u occurring in the condition of another one.

Definition 20.4 Let $g = d \text{ if } (\Delta \wedge s = t)$ and $l = r \text{ if } \Gamma$ be two conditional equalities with disjoint sets of variables, such that l and s unify at position ω of $\mathcal{G}rd(s)$ with the most general unifier ψ, $\psi(t) \not\geq \psi(s)$, $\psi(r) \not\geq \psi(l)$, $\forall(u_i = v_i) \in \Delta \cup \{g = d\}, \psi(u_i = v_i) \not\geq \psi(s = t)$, $\forall(s_i = t_i) \in \Gamma, \psi(s_i = t_i) \not\geq \psi(l = r)$. Then the *conditional ordered narrowing* obtained from these two equalities is the conditional equality

$$\psi(g) = \psi(d) \text{ if } \psi(\Gamma \wedge \Delta \wedge s[r]_\omega = t).$$

Let $NAR(P)$ be the set of conditional ordered narrowings obtained from any two equalities in P.

Lemma 20.2 [BG91b] Let $D = (g = d \text{ if } (\Delta \wedge s = t))$ and $C = (l = r \text{ if } \Gamma)$ be two conditional equalities with disjoint sets of variables, and let $\sigma(D) = (\sigma(g) = \sigma(d) \text{ if } \sigma(\Delta \wedge s = t))$ and $\sigma(C) = (\sigma(l) = \sigma(r) \text{ if } \sigma(\Gamma))$ be ground instances such that $\sigma(D) >_{\mathcal{C}} \sigma(C)$, $\sigma(C)$ ordered and $\forall x \in \mathcal{V}ar(D), \sigma(x)$ is irreducible by $l = r$. Then any narrowing inference from $\sigma(C)$ on $\sigma(D)$ produces a new formula which is a ground instance of a conditional ordered narrowing in $NAR(P)$.

Proof: The formula produced by the narrowing inference rule from $\sigma(C)$ and $\sigma(D)$ is

$$\psi(\sigma(g)) = \psi(\sigma(d)) \text{ if } \psi(\sigma(\Gamma) \wedge \sigma(\Delta) \wedge \sigma(s)[\sigma(r)]_\omega = \sigma(t))$$

where ψ is the most general unifier of the two terms $\sigma(s)_{|\omega}$ and $\sigma(l)$. Since these terms are ground, ψ is the identity and $\sigma(g)_{|\omega} = \sigma(l)$. Moreover, since $\forall x \in \mathcal{V}ar(D), \sigma(x)$ is irreducible by $l = r$, ω is a position in $\mathcal{G}rd(g)$ and $\sigma(s_{|\omega}) = \sigma(l)$. This means that σ is a unifier of $s_{|\omega}$ and l, greater than their most general unifier, say α. The conditions $\alpha(t) \not\geq \alpha(s)$, $\alpha(r) \not\geq \alpha(l)$, $\forall(u_i = v_i) \in \Delta \cup \{g = d\}, \alpha(u_i = v_i) \not\geq \alpha(s = t)$, $\forall(s_i = t_i) \in \Gamma, \alpha(s_i = t_i) \not\geq \alpha(l = r)$ all hold otherwise this would contradict one of the facts that $\sigma(C) = \beta\alpha(C)$ is ordered or $\sigma(D) = \beta\alpha(D)$ is greater than $\sigma(C)$. So there exists a conditional ordered narrowing

$$\alpha(g) = \alpha(d) \text{ if } \alpha(\Gamma \wedge \Delta \wedge s[r]_\omega = t)$$

which has as ground instance by β the initial formula. □

In addition to conditional critical pairs and conditional narrowings, the notion of resolvent is also needed. When one of the conditions of a rule can be immediately solved by syntactic unification, then we can build a *reflexive resolvent*.

Definition 20.5 Let $g = d \text{ if } (\Delta \wedge s = t)$ be a conditional equality, such that s and t unify with the most general unifier ψ, $\psi(\Delta) \not>_{\mathcal{C}} \psi(s = t), \psi(g = d) \not>_{\mathcal{C}} \psi(s = t)$. Then the *reflexive resolvent* obtained from this equality is the conditional equality

$$\psi(g) = \psi(d) \text{ if } \psi(\Delta).$$

Let $REF(P)$ be the set of reflexive resolvent obtained from all equalities in P.

Lemma 20.3 [BG91b] Let $D = (g = d \text{ if } \Delta \wedge s = t)$ be a conditional equality and let $\sigma(D) = (\sigma(g) = \sigma(d) \text{ if } \sigma(\Delta) \wedge \sigma(s) = \sigma(t))$ a ground instance such that $\sigma(s) = \sigma(t)$ is maximal in $\sigma(D)$. Then a reflection inference applied on the litteral $\sigma(s) = \sigma(t)$ of $\sigma(D)$ produces a new formula which is a ground instance of a reflexive resolvent of P.

Proof: The formula produced by the reflection inference rule from $\sigma(D) = \sigma(g) = \sigma(d) \text{ if } (\sigma(\Delta) \wedge \sigma(s) = \sigma(t))$ is

$$\psi(\sigma(g)) = \psi(\sigma(d)) \text{ if } \psi(\sigma(\Delta))$$

where ψ is the most general unifier of the two terms $\sigma(s)$ and $\sigma(t)$. Since these terms are ground, ψ is the identity and $\sigma(s) = \sigma(t)$. This means that σ is a unifier of s and t, greater than their most general unifier, say α. The conditions $\alpha(\Delta) \not>_{\mathcal{C}} \alpha(s = t), \alpha(g = d) \not>_{\mathcal{C}} \alpha(s = t)$ all hold otherwise this would contradict the fact that $\sigma(s) = \sigma(t)$ is maximal in $\sigma(D)$. So there exists a reflexive resolvent

$$\alpha(g) = \alpha(d) \text{ if } \alpha(\Delta)$$

which has as ground instance by β the initial formula. □

An essential property of the above newly generated formulas is that they are smaller for the ordering $>_{\mathcal{C}}$ than one of the initial formulas.

20.3.4 Saturated sets

In the pure equational case, given a set of axioms E, a confluent rewrite system R provides a congruence equivalent to $=_E$, the smallest congruence that contains the set of equalities E. In a similar way, we now build, from a set of conditional equalities P, a congruence on ground terms, and prove that the quotient set is a model of P, provided P is saturated under application of inference rules for equational Horn logic described in Figure 20.1.

Using induction on the ordering $>_{\mathcal{C}}$, we define for any ground instance C of a conditional equality in P, the sets of equalities E_C, R_C and I_C as follows:

$$R_C = \bigcup_{C >_{\mathcal{C}} C'} E_{C'} \quad \text{and} \quad I_C = \quad =_{R_C}$$

Then

$$E_C = \{s = t\}$$

if $C = (s = t \text{ if } \Gamma)$ such that
(i) C is ordered,
(ii) s is irreducible by R_C,
(iii) $\Gamma \subseteq I_C$.
Otherwise $E_C = \emptyset$

Then define I to be the quotient set of ground terms by $=_R$ where $R = \bigcup_C E_C$ is the set of all equalities produced by ground instances of conditional equalities in P. Note that by construction, R_C and R are left-reduced rewrite system. Thus they are convergent and equality in I may be decided using R: $s = t$ is true in I iff $s\downarrow_R = t\downarrow_R$.

The importance of this construction relies on the fact that I is a model of P, provided P is saturated. The notion of saturation itself relies on the concept of redundancy.

Intuitively a conditional equality is redundant if it does not contribute to the construction of I or in other words, does not modify the congruence relation on ground terms.

Definition 20.6 A ground instance C of a conditional equality in P is *redundant* if it is true in I_C. A conditional equality in P is redundant if all its ground instances are redundant.

A saturated set is a set to which no non-redundant consequence can be added.

Definition 20.7 A set of conditional equalities P is *saturated* if all ground instances of inferences from P are redundant.

With these definitions, we can state the result:

Theorem 20.6 *A saturated set of conditional equalities P has a model if and only if it does not contain the empty conditional equality* if.

Proof: The proof is an instance of the proof of the similar theorem stated in [BG91b] for first-order logic. We just sketch the proof below: if P contains the empty conditional equality if, then it has no model. On the other hand, if P is saturated and does not contain the empty conditional equality if, every ground instance C of P is true in the quotient of ground terms by $=_{R_C \cup E_C}$ and hence true in I. So I is a model of P. □

Example 20.8 Given constants 0 and *true*, function symbols s and P, the following set of conditional equalities is saturated:

$$\begin{array}{rrcl} & P(0) & = & true \\ P(x) = true \ \text{ if } & P(s(x)) & = & true \end{array}$$

Note that there is no superposition between the two conditional equalities and no narrowing because $P(x) < P(s(x))$ in any simplification ordering. Applying deduction rules without ordering restriction would lead to the generation of an infinite set of conditional equalities of the form

$$P(s^k(0)) = true, \text{ for } k = 1, 2, \ldots$$

When P is not saturated, a completion process has to be applied to saturate it.

20.4 Completion

The transition rules for completion of conditional equalities are deduced from the approach followed in the previous section. Here again a rewrite rule where the condition Γ is a conjunction of positive literals is viewed as an equational Horn clause. This approach of completion based on a set of transition rules which is refutationally complete for Horn clause logic with equality has been proposed by [KR87]. These rules are direct extensions of unfailing completion. There the idea was always to apply term replacement only within the larger member of an equality and never replace a term by a larger one. Similarly for equational Horn conditional equalities, any term replacement is performed only into the largest literals in a conditional equality using the largest equality literal of a conditional equality.

According to the previous section, the notion of completion is extended to a notion of saturation of a set of formulas (equalities or conditional equalities): all non-redundant inferences are computed. From inference rules given in Figure 20.1, three expansion rules of the completion process, namely **Superpose**, **Narrow** and **Reflect**, can be deduced immediately. Applying them until saturation is refutationally complete but highly inefficient. Then the key notion of redundancy allows dealing with simplification techniques, such as deletion of tautologies or subsumption. Redundancy has already been abstractly defined and these simplification techniques provide more effective criteria to eliminate redundancies.

20.4.1 Transition rules

Let P be a set of Horn conditional equalities. By $C[t]$ we mean that the term t occurs as a subterm in the conditional equality C. A conditional equality is viewed as a set of literals, and the set-containment relation on conditional equalities is denoted by $\subseteq$.

The transition rules of the unfailing completion procedure can be extended to deal with conditional equalities. The three first rules **Superpose**, **Narrow** and **Reflect** are expansion rules, while the four last ones are contraction rules.

Let $\mathcal{UCOND}$ be the set of transition rules for unfailing conditional completion given in Figure 20.2.

Superpose adds in P the set of conditional ordered critical pairs obtained by superposition of conditional equalities in P and **Narrow** adds in P the conditional ordered narrowings between two elements. **Reflect** removes a maximal premise of a conditional rule if it is solved. **Delete** and **Trivial** are rules for getting rid of tautologies. **Subsume** allows elimination of redundancies; a rule less general than another one is non-essential and can be deleted. **Simplify** reduces conjectures using conditional rules whose conditions are valid in the underlying theory. This rule is deliberately given in a very general format. Remind that $P \cup \{C[\sigma(t)]\} \models \sigma(s = t)$ means that $P \cup \{C[\sigma(t)]\}$ implies $\sigma(s = t)$, that is every model of $P \cup \{C[\sigma(t)]\}$ satisfies $\sigma(s = t)$. For implementation, the search of a proof of the conditions must be bounded in some way.

The transition rules are iterated on the initial set of conditional equalities. When this process stops, the resulting set of conditional ordered equalities is warranted to have the Church-Rosser property on ground terms.

Superpose	P	$\mapsto\!\!\!\!\rightarrow$	$P \cup \{(p = q \text{ if } \Gamma)\}$ if $(p = q \text{ if } \Gamma) \in SUP(P)$
Narrow	P	$\mapsto\!\!\!\!\rightarrow$	$P \cup \{(p = q \text{ if } \Gamma)\}$ if $(p = q \text{ if } \Gamma) \in NAR(P)$
Reflect	P	$\mapsto\!\!\!\!\rightarrow$	$P \cup \{(p = q \text{ if } \Gamma)\}$ if $(p = q \text{ if } \Gamma) \in REF(P)$
Delete	$P \cup \{(p = p \text{ if } \Gamma)\}$	$\mapsto\!\!\!\!\rightarrow$	P
Trivial	$P \cup \{(s = t \text{ if } \Gamma \wedge s = t)\}$	$\mapsto\!\!\!\!\rightarrow$	P
Subsume	$P \cup \{C, D\}$	$\mapsto\!\!\!\!\rightarrow$	$P \cup \{C\}$ if $\sigma(C) \subseteq D$
Simplify	$P \cup \{C[\sigma(s)]\}$	$\mapsto\!\!\!\!\rightarrow$	$P \cup \{C[\sigma(t)]\}$ if $C[\sigma(s)] >_{\mathcal{C}} C[\sigma(t)]$ and $C[\sigma(s)] >_{\mathcal{C}} \sigma(s = t)$ and $P \cup \{C[\sigma(t)]\} \models \sigma(s = t)$

Figure 20.2: The set $\mathcal{UCOND}$ for unfailing conditional completion

For a practical application of simplification, a conditional equality can be used to rewrite a term only if it is reductive. Otherwise, it is only considered for completion to generate new consequences but it can be deleted once completion is achieved, since it will never be applied for normalizing terms. The next examples illustrates these notions and the completion process.

Example 20.9 The following set of conditional equalities in which $\leq$ defines the predicate *less or equal* on integers has the Church-Rosser property on ground terms:

$$\begin{array}{rclcl}
 & & succ(pred(x)) & \rightarrow & x \\
 & & pred(succ(x)) & \rightarrow & x \\
 & & (0 \leq 0) & \rightarrow & true \\
 & & (0 \leq pred(0)) & \rightarrow & false \\
 & & (succ(x) \leq y) & \rightarrow & (x \leq pred(y)) \\
 & & (pred(x) \leq y) & \rightarrow & (x \leq succ(y)) \\
(0 \leq x) = true & \text{if} & (0 \leq succ(x)) & \rightarrow & true \\
(0 \leq x) = false & \text{if} & (0 \leq pred(x)) & \rightarrow & false \\
(0 \leq pred(x)) = true & \text{if} & (0 \leq x) & = & true \\
(0 \leq succ(x)) = false & \text{if} & (0 \leq x) & = & false.
\end{array}$$

The last two conditional equalities have been generated during the completion of the other ones. Note that none of their instances are reductive; when they are discarded, the remaining system still has the Church-Rosser property.

Example 20.10 This example is a completion performed by the system CEC and reported in [Gan91]. Consider the example of natural numbers with $\leq$. Transitivity and totality axioms are added to the definition of $\leq$. These axioms are non-reductive conditional equalities.

$$\begin{array}{rclcl}
 & & 0 \leq x & = & true \\
 & & succ(x) \leq 0 & = & false \\
 & & succ(x) \leq succ(y) & = & x \leq y \\
 & & x \leq x & = & true \\
 & & x \leq succ(x) & = & true \\
(x \leq y) = true & \text{if} & x \leq succ(y) & = & true \\
(x \leq y) = false & \text{if} & y \leq x & = & true \\
(x \leq y = true \wedge y \leq z = true) & \text{if} & x \leq z & = & true
\end{array}$$

Starting with this specification, CEC terminates with the following set of rewrite rules:

$$\begin{array}{rclcl}
 & & 0 \leq x & \rightarrow & true \\
 & & succ(x) \leq 0 & \rightarrow & false \\
 & & succ(x) \leq succ(y) & \rightarrow & x \leq y \\
 & & x \leq x & \rightarrow & true \\
(x \leq y) = true & \text{if} & x \leq succ(y) & \rightarrow & true
\end{array}$$

and the following set of conditional equalities:

$$\begin{array}{rlll} (x \leq y) = false & \text{if} & y \leq x & = & true \\ (x \leq y = true \wedge y \leq z = true) & \text{if} & x \leq z & = & true \\ s(x) \leq y = true & \text{if} & x \leq y & = & true \\ (succ(y) \leq z = true \wedge x \leq y = true) & \text{if} & x \leq z & = & true \\ (succ(y) \leq z = true \wedge x \leq y = true) & \text{if} & succ(x) \leq z & = & true \end{array}$$

For a precise description of the computations performed by CEC and their justifications, see [Gan91].

20.4.2 Refutational completeness proof

Let us consider $P_0 = P$ and apply the transition rules in $\mathcal{UCOND}$ to it. We get a derivation $P_0 \mapsto\!\!\!\!\rightarrow_{\mathcal{UCOND}} P_1 \mapsto\!\!\!\!\rightarrow_{\mathcal{UCOND}} P_2$... and call its limit P_∞. Let also $P_* = \bigcup_i P_i$ denote the union of generated sets of conditional equalities. According to the previous theorem 20.6, if P_∞ is saturated and does not contain the empty conditional equality, P_∞ is consistent since we know how to build a model. We introduce a notion of fairness of the derivation that ensures both that P_∞ is saturated and that the model built for it is also a model for P_0. We thus prove that the set of transition rules $\mathcal{UCOND}$ is refutationally complete: a contradiction (the empty conditional equality) can be derived from any inconsistent set of conditional equalities.

The fairness assumption ensures that no crucial transition rule will be postponed forever. Intuitively, a derivation is fair if all conditional equalities which can be generated from persisting conditional equalities are either generated or made redundant by the generation of other conditional equalities.

But the definition of redundancy previously given is too general and we now need sufficient conditions for redundancy that also cover simplification and deletion techniques. The concept of composite conditional equality is introduced for this purpose. Intuitively, every composite ground instance of some conditional equality in a saturated set is redundant. This notion of compositeness must be related to the definition of composite critical pairs given in Section 19.3 of Chapter 19.

Definition 20.8 A ground conditional equality C is *composite* w.r.t. P if there exist ground instances $C_1, ..., C_k$ of conditional equalities in P such that $C_1, ..., C_k \models C$ and $C >_{\mathcal{C}} C_j$ for all $1 \leq j \leq k$. A non-ground conditional equality C is composite w.r.t. P if all its ground instances are composite.

The next lemma shows that compositeness w.r.t. a set P is preserved if conditional equalities are added or if composite conditional equalities are deleted. In the completion process, a conditional equality that is composite w.r.t. P_i at some step will remain composite w.r.t. P_j, $j > i$, since any expansion rule adds new conditional equalities and any contraction rule only deletes composite conditional equalities.

Lemma 20.4 [BG91b] If $P \subseteq P'$, then any conditional equality composite w.r.t. P is also composite w.r.t. P'.

If $P \subseteq P'$ and all conditional equalities in $P' - P$ are composite w.r.t. P, then any conditional equality composite w.r.t. P' is also composite w.r.t. P.

This lemma is applied to $P_\infty \subseteq P_*$, to prove that if every formula deduced from P_∞ is composite w.r.t. P_*, then the model I associated to P_∞ is also a model for P_*.

We now consider contraction rules and prove their correctness by showing that they eliminate redundant formulas. **Delete**, **Trivial**, **Subsume** and **Simplify**.

Lemma 20.5 If C subsumes D (i.e. $D = \sigma(C)$ for some substitution σ), then D is composite w.r.t. C.

Proof: $\sigma(C) \models D$ and $D >_{\mathcal{C}} C$. □

Lemma 20.6 A conditional equality ($p = p$ if Γ) or ($s = t$ if $\Gamma \wedge s = t$) is composite.

Proof: Since $\models D$, one can choose an empty set of ground instances $C_1, \ldots, C_k$ in Definition 20.8. □

Lemma 20.7 In the **Simplify** rule, $C[\sigma(s)]$ is composite w.r.t. $P \cup \{C[\sigma(t)]\}$.

Proof: First $C[\sigma(s)]$ is composite w.r.t. $P \cup \{C[\sigma(t)], \sigma(s = t)\}$ since $C[\sigma(s)] >_{\mathcal{C}} C[\sigma(t)]$, $C[\sigma(s)] >_{\mathcal{C}} \sigma(s = t)$, and $P \cup \{C[\sigma(t)], \sigma(s = t)\} \models C[\sigma(s)]$. Then it is also composite w.r.t. $P \cup \{C[\sigma(t)]\}$ since $P \cup \{C[\sigma(t)]\} \models \sigma(s = t)$. □

Let us also say some words about the frequently used technique of case analysis. The first step in a case analysis consists of splitting a conditional equality $C = \Gamma \Rightarrow l = r$ of the set of conditional equalities P into n conditional equalities $C_i = A_i \wedge \Gamma \Rightarrow l = r$ where all A_i are consequences of the set of ground instances C' of C such that $C >_{\mathcal{C}} C'$. Suppose in addition that C_i is simplified into D_i. Then C becomes composite in $P \cup \{D_1, \ldots, D_n\}$. So case analysis can be justified with the same technique. More details are given in [BG91b].

With this notion of composite formula, the property of fairness can be formulated:

Definition 20.9 A derivation $P_0 \mapsto\!\!\!\!\rightarrow_{\mathcal{UCOND}} P_1 \mapsto\!\!\!\!\rightarrow_{\mathcal{UCOND}} P_2 \ldots$ is called *fair* if every inference from P_∞ is composite w.r.t. P_*.

Definition 20.10 A set of conditional equalities P is *complete* if all inferences from P are composite w.r.t. P.

Fairness, completeness and saturation are related in the following way:

Lemma 20.8 If a derivation $P_0 \mapsto\!\!\!\!\rightarrow_{\mathcal{UCOND}} P_1 \mapsto\!\!\!\!\rightarrow_{\mathcal{UCOND}} P_2 ...$ is fair then P_∞ is complete and every conditional equality in $P_* - P_\infty$ is composite w.r.t. P_∞.

Lemma 20.9 Any complete set of conditional equalities that does not contain the empty conditional equality is saturated.

Using these two lemmas, we get:

Theorem 20.7 *Let $P_0 \mapsto\!\!\!\!\rightarrow_{\mathcal{UCOND}} P_1 \mapsto\!\!\!\!\rightarrow_{\mathcal{UCOND}} P_2 ...$ be a fair derivation. If P_* does not contain the empty conditional equality, then P_∞ is saturated and P_0 is consistent.*

Proof: By fairness, P_∞ is complete. Using Lemma 20.9, it is saturated. Indeed, if P_* does not contain the empty conditional equality, nor does P_∞. Using Lemma 20.8, the interpretation I constructed from P_∞ is a model of P_*. $\Box$

20.5 Conclusion

The saturation process presented in this chapter is more flexible than previous conditional completion procedures proposed for instance by [JW86, KR89b] since it does not fail in the presence of non-reductive rules or non-orientable equalities. Moreover, this technique also applies to conditional equalities with disequations in their conditions and the method can be used to refute goals in the logic programming sense. The correspondance with logic programs with negation is explained in [BG91a].

The results described here are proved using induction methods based on powerful orderings but no more on a proof reduction process. Actually attemps have been made both by Bachmair in [Bac91] and by Dershowitz in [Der90] to apply the proof ordering technique to Horn clauses. They both proved that a unit strategy with simplification is complete for reducing any proof of an equality theorem to some normal form. But the unit strategy limits the application of superposition to (non-conditional) equalities and allows narrowing in the conditions only using (non-conditional) equalities.

Chapter 21

Completion with constraints

21.1 Introduction

A first motivation for introducing constraints in completion processes arises when considering the problem of completion modulo a set of axioms A [BD89a, JK86c]. A main drawback of this class of completion procedures is an inherent inefficiency, due to the computation of matches and unifiers modulo A. For instance, let us consider the two rewrite rules

$$\begin{aligned} x * x * x * x &\to x \\ u * v * w * a &\to a. \end{aligned}$$

where $*$ is supposed to be associative and commutative (AC for short). Note that a complete set of AC-solutions for the equation $x*x*x*x =^?_{AC} u*v*w*a$ where x, u, v, w are variables and a a constant, contains several millions of substitutions [Dom92]. The idea is here to use constraints to record unification problems in the theory AC and to avoid solving them immediately. By constrained superposition at top occurrence, the constrained critical pair

$$(x = a \parallel (x * x * x * x =^?_{AC} u * v * w * a))$$

is computed. It schematizes for instance the trivial critical pair $(a = a)$ obtained with the solution $\{(x \mapsto a)(u \mapsto a)(v \mapsto a)(w \mapsto a)\}$; it also schematizes the critical pair $(a * x' = a)$, by considering the solution $\{(x \mapsto a * x')(u \mapsto a * x' * x')(v \mapsto a * x')(w \mapsto a * x')\}$.

Another example is the Ring Commutativity Problems in Example 19.2 of Chapter 19. In this example for the case n, the equation $w^n = w$ is added to the set $RING$ of equations. The first interesting critical pair is between the extended equality of distibutivity and the equation $w^n = w$. Following [Dom92], it can be computed that an equation of the form $v * w^n =^?_{AC} u * x * (y + z)$ has a set of minimal AC-solutions whose cardinal is $(6n + 8)2^n - 12$. Instead of computing this huge set of corresponding critical pairs, the following constrained critical pair $[u * ((x * y) + (x * z)) = v * w \parallel v * w^n =^?_{AC} u * x * (y + z)]$ is built and the deduction process goes on and adds other constraints which further narrow the set of solutions.

A third example of the use of constraints in completion processes is provided by theories involving commutativity, associativity and identity axioms ($ACIdentity$ for short). The idea to extend the technique of ordered completion modulo A to this kind of theories emerges from the remark that it is in general easier to solve equations modulo $ACIdentity$ than modulo AC, in the sense that complete sets of unifiers have less elements. So for theories like commutative group, commutative ring with unit, Boolean algebra, group or ring homomorphism, distributive lattice, it is interesting to deal with the commutativity, associativity and identity axioms through the matching and unification processes. Unfortunately a termination problem arises since the rule $-(x + y) \to (-x) + (-y)$ leads to an infinite derivation for any term t: $(-t) \xleftrightarrow{*}_{ACIdentity} -(t+0) \to (-t)+(-0) \to \ldots$. The idea is then to prevent this infinite chain by imposing the constraint that both x and y must be non-equivalent to 0 modulo $ACIdentity$ to apply the rule $-(x + y) \to (-x) + (-y)$. So when generated, the equality $(-(x + y) = (-x) + (-y))$ is split into:

a rewrite rule $-(x + y) \to (-x) + (-y) \parallel (x \neq^?_{ACIdentity} 0 \wedge y \neq^?_{ACIdentity} 0)$,

and an equality $-(x + y) = (-x) + (-y) \parallel (x =^?_{ACIdentity} 0 \vee y =^?_{ACIdentity} 0)$,

itself equivalent to two equalities

$-(x + y) = (-x) + (-y) \parallel (x =^?_{ACIdentity} 0)$ and $-(x + y) = (-x) + (-y) \parallel (y =^?_{ACIdentity} 0)$,

further reduced to $-(y) = (-0) + (-y)$ and $-(x) = (-x) + (-0)$, then to trivial equalities. Further developments may be found in [BPW89, JM90] and other examples of theories are handled in [BPW89]: commutative ring with unit, Boolean algebra, group or ring homomorphism, distributive lattice.

So the notions of rewriting and completion with constraints take their interest from the fact that checking satisfiability of a constraint is in general much simpler than finding a complete set of solutions or a solved form, especially in equational theories. For instance, a criterion to check the satisfiability of a system of equations modulo associativity-commutativity is given in [Dom91a]. Finding complete sets of AC-solutions of equation systems is much more difficult and less efficient. Originally a completion procedure with AC-constraints has been proposed in [KK89] and a general framework for deduction with constraints developped in [KKR90]. Ordering and equality constraints were proposed for several deduction processes in first-order logic with equality. Then the same idea was used in implementations of ordered completion described in [MN90, Pet90]. and completion modulo associativity, commutativity and identity [BPW89, JM90]. Completion with membership constraints is studied in [Com91a, Com92]. Later on, the relation between constrained superposition and basic superposition has been enlighted in [BGLS92, NR92a, NR92b]. Completeness results for deduction systems based on constrained superposition were obtained.

21.2 Constrained rewriting

A very general and powerful notion of rewriting with constraints was introduced in Section 7.6 of Chapter 7. The definition is slightly restricted here to cases that are useful in the context of a theorem proving process. Since we are mainly interested in deduction processes like completion which involves both ordering conditions and equational problems, we assume from now on, that the constraint language $L_{\mathcal{K}}[\Sigma, \mathcal{X}]$ is specialized to conjunctions of equations and inequations. The interpretation $\mathcal{K}$ is chosen as $\mathcal{T}(\Sigma, \mathcal{X})/\overset{*}{\longleftrightarrow}_A$ and we do not introduce new function symbols in the signature to build our constraint formulas. The equality predicate is interpreted in $\mathcal{K}$ by equality modulo A and the ordering predicate by an A-compatible reduction ordering. Let $L_>$ denote this instance of $L_{\mathcal{K}}$.

A variant of Definition 7.36 is necessary:

Definition 21.1 Let CE be a set of constrained equalities and $>$ an A-compatible reduction ordering on $\mathcal{T}(\Sigma, \mathcal{X})$. The relation $(CE, A, L_>)$ is defined on $\mathcal{T}(\Sigma, \mathcal{X})$ by : $t \rightarrow_{CE,A,>} t'$ if $\exists (l = r \parallel c) \in CE$, $\exists \sigma$ s.t. $t_{|\omega} \overset{*}{\longleftrightarrow}_A \sigma(l)$, $\sigma \in SS_{\mathcal{K}}(c \wedge (l >^?_A r))$, $t' = t[\sigma(r)]_\omega$.

When there is a common orientation for the equality instances, we get the notio of constrained rewrite rules.

Definition 21.2 Let CR be a set of constrained rewrite rules. The relation $(CR, A, L_>)$ is defined on $\mathcal{T}(\Sigma, \mathcal{X})$ by : $t \rightarrow_{CR,A,L_>} t'$ if $\exists (l \rightarrow r \parallel c) \in CR$, $\exists \sigma$ s.t. $t_{|\omega} \overset{*}{\longleftrightarrow}_A \sigma(l)$, $\sigma \in SS_{\mathcal{K}}(c)$, $t' = t[\sigma(r)]_\omega$.

Note that the matching problem is solved in the constraint language $L_>$. Assuming that there is a unification procedure in $L_>$ to solve equational constraints is enough. Then the substitution σ is a solution modulo A of a restricted unification problem.

21.3 Constrained superposition

Let us first concentrate on the case where A is an empty set of axioms.

Definition 21.3 A *constrained critical pair* of $(g = d \parallel c')$ and $(l = r \parallel c)$ is the constrained equality $(g[r]_\omega = d \parallel c \wedge c' \wedge (g_{|\omega} =^?_\emptyset l) \wedge (g >^?_\emptyset d) \wedge (l >^?_\emptyset r))$ if $c \wedge c' \wedge (g_{|\omega} =^?_\emptyset l) \wedge (g >^?_\emptyset d) \wedge (l >^?_\emptyset r)$ is satisfiable.

A constrained critical pair of the form

$$(g[\omega \hookleftarrow r] = d \parallel (c \wedge c' \wedge (g_{|\omega} =^?_\emptyset l) \wedge (g >^?_\emptyset d) \wedge (l >^?_\emptyset r)))$$

schematizes the following set of formulas:

$$\mathcal{CP} = \{\sigma(g[\omega \hookleftarrow r]) = \sigma(d) | \sigma \in SS_\emptyset(c \wedge c' \wedge (g_{|\omega} =^?_\emptyset l) \wedge (g >^?_\emptyset d) \wedge (l >^?_\emptyset r))\}.$$

The superposition rule for constrained equalities with equality and inequality constraints is formalized as follows:

Basic-Superpose $(l = r \parallel c), (g = d \parallel c')$

$\mapsto\!\!\!\!\twoheadrightarrow$

$(g[\omega \hookleftarrow r] = d \parallel c \wedge c' \wedge (g_{|\omega} =^?_\emptyset l) \wedge (l >^?_\emptyset r) \wedge (g >^?_\emptyset d))$

if $c \wedge c' \wedge (g_{|\omega} =^?_\emptyset l) \wedge (l >^?_\emptyset r) \wedge (g >^?_\emptyset d)$ is satisfiable

Figure 21.1: Basic Superposition Rule with equality and inequality constraints

Example 21.1 Given $\mathcal{F} = \{+, 0\}$ and the precedence $0 \prec +$, let us consider the following equalities:

$$\begin{aligned} x + 0 &= 0 + x \\ (x + y) + z &= x + (y + z) \end{aligned}$$

that can be converted into constrained rules:

$$(x + 0 \rightarrow 0 + x \parallel x >^?_\emptyset 0)$$
$$((x + y) + z \rightarrow x + (y + z) \parallel \mathbf{T})$$

There are two constrained superpositions:

$$((0 + x) + z = x + (0 + z) \parallel x >^?_\emptyset 0)$$
$$(0 + (x + y) = x + (y + 0) \parallel (x + y) >^?_\emptyset 0)$$

Note that the constraint of the second rule is always satified so may be simplified to $\mathbf{T}$:

$$(0 + (x + z) = x + (0 + z) \parallel x >^?_\emptyset 0)$$
$$(0 + (x + y) = x + (y + 0) \parallel \mathbf{T})$$

The difficulty that arises in trying to prove confluence is that the critical pairs lemma does not hold anymore in the context of constrained rewrite rules.

Example 21.2 Let us consider $A = \emptyset$, $\mathcal{F} = \{a, b, f, g\}$ and a simplification ordering $>$ induced by the precedence $f, g > a > b$ [DJ90a]. Let CE be the set of two constrained equalities: $(g(a) = b \parallel \mathbf{T})$ and $(f(x) = a \parallel x =^?_\emptyset g(y))$. There is no constrained critical pair, nevertheless there exists a peak $a \leftarrow_{CE,\emptyset,L_>} f(g(a)) \rightarrow_{CE,\emptyset,L_>} f(b)$ which is not convergent since neither a nor $f(b)$ is reducible.

Example 21.3 The same problem occurs with ordering constraints. Let $\mathcal{F} = \{a, b, c, f\}$ with $f > a > b > c$.

$$a = c$$
$$f(x) = c \parallel x >^?_\emptyset b$$

There is no constrained critical pair.

Nevertheless there exists a peak

$$c \leftarrow f(a) \rightarrow f(c)$$

not convergent since neither c nor $f(c)$ is reducible.

In the previous examples, the problem comes from the fact that there is a superposition in the constraint part which is not taken into account by the computation of constrained critical pairs. A first idea is to look for cases where superposition into constraints is useless. Using a hierarchical approach in which constraints are restricted to a subsignature is enough to recover the critical pair lemma. This is done with so-called built-in constraints in [KR93] for instance. When the initial set of equalities is unconstrained, the basic superposition deduction rule (the rule **Basic-Superpose** given in Figure 21.1) is the basis of a saturation process in [NR92b, BGLS92].

Theorem 21.1 *[NR92a, NR92b, BGLS92] Let $A = \emptyset$ and E_0 be a set of equalities without constraints and let E be the closure of E_0 under the superposition rule* **Superpose***. Then E is confluent on ground terms.*

Another alternative initially proposed in [KKR90] is to use propagation. Constraints are weakened by partially solving them and propagating the instantiations in the equality. This formalized by the rule **Propagate** given in Figure 21.2 for a general term-generated interpretation domain, where $\hat{\theta}$ is the equational form of the substitution θ defined on the domain $\mathcal{D}om(\theta)$.

$$\begin{array}{ll}\textbf{Propagate} & CE \cup \{(g = d \parallel c)\} \\ & \Vdash\!\!\twoheadrightarrow \\ & CE \cup \{(\theta(g) = \theta(d) \parallel c')\} \\ & \text{if } c \equiv_{\mathcal{K}} c' \wedge \hat{\theta} \text{ and } \mathcal{D}om(\theta) \cap \mathcal{V}ar(c') = \emptyset \end{array}$$

Figure 21.2: Propagation Rule

Example 21.4 Coming back to the example 21.2, the rule **Propagate** applies straightforwardly and yields the (unconstrained) equality $(f(g(y)) = a)$. Now **Superpose** applies and generates with $(g(a) = b)$ the critical pair $(f(b) = a)$. The ordered rewrite system given by $>$ and the equalities

$$g(a) = b \quad f(g(y)) = a \quad f(b) = a$$

is Church-Rosser and generates the same congruence as the initial set of constrained equalities.

Note however that applying the rule **Propagate** assumes that constraints have solved forms similar to substitutions. This not the case indeed when membership constraints or disequations are considered for instance, since we should allow $x \neq a$ or $x \in A$ as solved constraints.

Example 21.5 Consider for instance the specification defined by a set of sorts $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$, a set of function symbols $\mathcal{F} = \{a, f, g, h\}$, a set of function declarations

$$\begin{array}{rrcl} a: & & \to & s_0 \\ f: & s_0 & \to & s_0 \\ & s_1 & \to & s_2 \\ & s_2 & \to & s_2 \\ & s_3 & \to & s_3 \\ g: & s_0 & \to & s_1 \\ & s_3 & \to & s_3 \\ h: & s_3, s_3 & \to & s_3 \end{array}$$

a set of subsort declarations

$$\begin{array}{rcl} s_0 & \leq & s_3 \\ s_1 & \leq & s_3 \\ s_2 & \leq & s_3 \end{array}$$

and a set of rewrite rules with membership constraints:

$$\begin{array}{lcl} f(x) \to a & \parallel & x \in s_2 \\ g(x) \to h(x, x) & \parallel & x \in s_3 \end{array}$$

Again **Deduce** does not apply, although we have a peak:

$$f(f(h(a, a)) \leftarrow f(f(g(a))) \to a$$

Now applying the rule **Propagate** is also problematic, since we do not know how to propagate solutions of $x \in s_2$. This problem is considered in [Com92]. Using the correspondance between order-sorted signature and regular tree automaton, the constraint $x \in s_2$ is first converted into a membership relation $(x \in^? f^+(g(f^*(a))))$ with a regular expression for s_2. Then from

$$\begin{array}{lcl} f(x) \to a & \parallel & x \in^? f^+(g(f^*(a))) \\ g(x) \to h(x, x) & & \end{array}$$

two constrained critical equalities are deduced:

$$\begin{array}{rcl} a = f(X(h(x, x))) & \parallel & (x \in^? f^*(a) \wedge X \in^? f^+) \\ a = Y(a) & \parallel & (Y \in^? f^+) \end{array}$$

The computation mechanism of these critical pairs involves second order unification and the critical pairs contain second order variables X and Y.

21.4 Constrained superposition modulo A

Let us now come back to the more general case where A is not empty. In that case the completion process requires the computation of constrained extensions as well as constrained critical pairs modulo A.

Definition 21.4 A *constrained critical pair modulo A* of $(g = d \parallel c')$ and $(l = r \parallel c)$ is the constrained equality $(g[r]_\omega = d \parallel c \wedge c' \wedge (g_{|\omega} =^?_A l) \wedge (g >^?_A d) \wedge (l >^?_A r))$ if $c \wedge c' \wedge (g_{|\omega} =^?_A l) \wedge (g >^?_A d) \wedge (l >^?_A r)$ is satisfiable.

A constrained critical pair of the form

$$(g[\omega \hookleftarrow r] = d \parallel (c \wedge c' \wedge (g_{|\omega} =^?_A l) \wedge (g >^?_A d) \wedge (l >^?_A r)))$$

schematizes the following set of formulas:

$$\mathcal{CP} = \{\sigma(g[\omega \hookleftarrow r]) = \sigma(d) | \sigma \in SS_A(c \wedge c' \wedge (g_{|\omega} =^?_A l) \wedge (g >^?_A d) \wedge (l >^?_A r))\}.$$

Definition 21.5 A *constrained extended equality* of $(l = r \parallel c)$ w.r.t. $(g = d) \in A$ is the constrained equality $(g[l]_\omega = g[r]_\omega \parallel c \wedge (g_{|\omega} =^?_A l) \wedge (l >^?_A r))$ if $c \wedge (g_{|\omega} =^?_A l) \wedge (l >^?_A r)$ is satisfiable.

For a set CE of constrained equalities, CE^{ext} denotes the saturation of CE under adjunction of constrained extended equalities.

As for ordered completion, more optimized rules may be given when A is composed of associativity and commutativity axioms, which we assume from now on in this section. Let us call CCM the set of rules for Constrained Completion Modulo given in Figure 21.3. The next result is a reformulation of Theorem 6.1 in [NR94].

Theorem 21.2 *Let E_0 be a set of equalities without constraints, $>$ an AC-compatible simplification ordering total on A-equivalence classes of ground terms, and let CE be the closure of E_0 under CCM. Then $(CE^{ext}, AC, L_>)$ is Church-Rosser modulo AC.*

A refutationally complete theorem prover based on constrained paramodulation and using a different proof technique is proposed in [Vig93].

An additional difficulty in the completeness proof, is to prove that completeness is preserved when simplification and deletion are incorparated into constrained completion. A simple sufficient condition is expressed in the condition of rule **Simplify** in Figure 21.3, but more powerful conditions can be found in [BGLS92].

A restricted form of simplification is applied in this process and the next section is devoted to a more powerful notion of simplification using constrained rewrite rules.

21.4.1 Constrained simplification

Defining a very general notion of simplification for constrained formulas is more complex than constrained deduction. This is because when simplifying, the starting formula is *replaced* by the simplified one and lost, while otherwise, when a deduction step is applied, a new formula is *added*. In the simplification process, it must be taken care of not loosing schematized instances of the initial formula. This is why the definition of simplification given below involves two parts. The first one deals with instances which are really simplified by an instance of the constrained rewrite rule. The second part deals with the non-simplifiable instances and records the failure in the constraint. To simplify a constrained formula $(F_1 \parallel c_1)$, where F_1 may be in particular a rewrite rule or an equality, using a constrained rewrite rule $(l \rightarrow r \parallel c)$, it is assumed that the two constrained formulas are variable disjoint. This condition can always be satisfied by renaming variables of the constrained rewrite rule. It is also assumed that $(l \rightarrow r \parallel c)$ satisfies $\mathcal{V}ar(c) \subseteq \mathcal{V}ar(l) \cup \mathcal{V}ar(r)$.

Definition 21.6 A constrained formula $(F_1 \parallel c_1)$ *simplifies* to $(F_2 \parallel c_2)$, with the rest $(F_1 \parallel c'_2)$, at the non-variable position ω of F_1, with the constrained rewrite rule $(l \rightarrow r \parallel c)$, if the constraint $c_2 = c_1 \wedge (c \wedge (l =^?_K F_{1|\omega}))$ is satisfiable. Then $F_2 = F_1[r]_\omega$ and $c'_2 = (c_1 \wedge [\forall \mathcal{V}ar(c) \cup \mathcal{V}ar(l), \quad \neg c \vee \neg(l =^?_K F_{1|\omega})])$.

The previous definition given for simplifying a constrained formula $(F \parallel c)$ takes into account all the schematized formulas. Some of these instances that are not reducible are recorded in the rest. It should be noted that solving constraints with negation is needed in this definition. This general form of simplification is used in [LS93]. When the constraint associated to the rest is unsatisfiable, this means that all instances of the formula are reducible. This case is sometimes called a total simplification.

Deduce-Eq-Eq
$CE \cup \{(g = d \parallel c), (l = r \parallel c')\}$
$\mapsto\!\!\!\twoheadrightarrow$
$CE \cup \{ \begin{array}{l} (g = d \parallel c), (l = r \parallel c'), \\ (g[r]_\omega = d \parallel c \wedge c' \wedge (g_{|\omega} =^?_{AC} l) \wedge (g >^?_{AC} d) \wedge (l >^?_{AC} r)) \end{array} \}$
if $c \wedge c' \wedge (g_{|\omega} =^?_{AC} l) \wedge (g >^?_{AC} d) \wedge (l >^?_{AC} r)$ satisfiable

Deduce-Ext-Eq
$CE \cup \{(g = d \parallel c), (l = r \parallel c')\}$
$\mapsto\!\!\!\twoheadrightarrow$
$CE \cup \{ \begin{array}{l} (g = d \parallel c), (l = r \parallel c'), \\ (g[f(r, z)]_\omega = d \parallel c \wedge c' \wedge (g_{|\omega} =^?_{AC} f(l, z)) \wedge (g >^?_{AC} d) \wedge (l >^?_{AC} r)) \end{array} \}$
if $c \wedge c' \wedge (g_{|\omega} =^?_{AC} f(l, z)) \wedge (g >^?_{AC} d) \wedge (l >^?_{AC} r)$ satisfiable

Deduce-Ext-Ext
$CE \cup \{(g = d \parallel c), (l = r \parallel c')\}$
$\mapsto\!\!\!\twoheadrightarrow$
$CE \cup \{ \begin{array}{l} (g = d \parallel c), (l = r \parallel c'), \\ (f(d, z) = f(r, z') \parallel c \wedge c' \wedge (f(g, z) =^?_{AC} f(l, z')) \wedge (g >^?_{AC} d) \wedge (l >^?_{AC} r)) \end{array} \}$
if $c \wedge c' \wedge (f(g, z) =^?_{AC} f(l, z')) \wedge (g >^?_{AC} d) \wedge (l >^?_{AC} r)$ satisfiable

Delete
$CE \cup \{p = q \parallel c\}$
$\mapsto\!\!\!\twoheadrightarrow$
CE
if $p \overset{*}{\longleftrightarrow}_{AC} q$

Simplify
$CE \cup \{p = q \parallel c\}$
$\mapsto\!\!\!\twoheadrightarrow$
$CE \cup \{p' = q \parallel c\}$
if $p \rightarrow^{(g=d \parallel c'),\sigma}_{CE,AC,L_>} p'$ with $\mathcal{V}ar(c') \subseteq \mathcal{V}ar(g) \cup \mathcal{V}ar(d)$ and $\mathcal{R}an(\sigma) \subseteq \mathcal{R}an(\theta), \forall\theta \in CSS_{AC}(c)$

Figure 21.3: CCM: Constrained Completion Modulo AC

Example 21.6 Consider the constrained rewrite rule,

$$x * x * x * x \rightarrow x \parallel (x \neq^?_\emptyset a)$$

where $*$ is denoted in infix notation and the expressions are supposed to be left parenthesized, whenever parentheses are implicit. The constrained equality $((b * b * b * b) * (y * y * y * y) = a \parallel \mathbf{T})$ with the trivial constraint $\mathbf{T}$ simplifies to another constrained equality

$$(b * b * b * b * x = a \parallel (x * x * x * x =^?_\emptyset y * y * y * y) \wedge (x \neq^?_\emptyset a))$$

with the rest

$$(b * b * b * b) * (y * y * y * y) = a \parallel \forall x, \ \neg(x * x * x * x =^?_\emptyset y * y * y * y) \vee \neg(x \neq^?_\emptyset a).$$

Example 21.7 Let us consider the simplification of $(a * y = z \parallel y * z =^?_{AC} d * b)$ using $(a * b = b \parallel \mathbf{T})$ and the substitution $\sigma = \{y \mapsto b, \ z \mapsto d\}$. Then the simplification produces two new constrained equalities: $(b = z \parallel y =^?_{AC} b \wedge z =^?_{AC} d)$ and $(a * y = z \parallel (y * z =^?_{AC} d * b) \wedge ((y \neq^?_{AC} b) \vee (z \neq^?_{AC} d)))$.

Beyond correctness, the definition of such a simplification relation rises two other problems: termination and relationship with other definitions of simplification.

Constrained simplification needs to check first that the list of constraints c_2 is satisfiable. Otherwise obviously termination problems arise. But this is not enough to ensure termination of the process, as shown in the next example. The growth of the constraints must be controlled too and this must be taken into account by a reduction strategy.

Example 21.8 Let us consider $A = \emptyset$, the constrained rewrite rule $(f(x) \to f(y) \parallel (x >^?_\emptyset y))$ and the equality $(f(a) = b \parallel \mathbf{T})$. This equality is simplified into: $(f(y) = b \parallel (x =^?_\emptyset a) \wedge (x >^?_\emptyset y))$ and $(f(a) = b \parallel \forall x, y, \;\; (x \neq^?_\emptyset a) \vee \neg(x >^?_\emptyset y))$. The first constrained equality $(f(y) = b \parallel (x =^?_\emptyset a) \wedge (x >^?_\emptyset y))$ can be simplified again; for instance, we obtain after n steps:

$$(f(y_n) = b \parallel a >^?_\emptyset y >^?_\emptyset \cdots >^?_\emptyset y_n).$$

Let us consider now restrictions of Definition 21.6 that are used in practice.

• If the formula F_1 is reducible by a constrained rewrite rule $(l \to r \parallel c)$, there exists a match σ that satisfies also c. All formulas schematized by $(F_1 \parallel c_1)$ are reducible by this rule and there is no rest in such a simplification.

Example 21.9 Let us consider the simplification of $(a * b = z \parallel a * z =^?_{AC} a * b)$ by constrained rewriting with $(x * a \to y \parallel x * y =^?_{AC} a * b)$ and the substitution $\sigma = \{x \mapsto b, \; y \mapsto a\}$ which is both a match and a solution of the constraint. Then the simplification produces the constrained equality: $(a = z \parallel (a * z =^?_{AC} a * b))$.

• If simplification is performed by matching l to a subterm of F_1 with a substitution σ, variables from the formula F_1 are not instantiated. Then we can define $F_2 = F_1[\sigma(d)]_\omega$, $c_2 = c_1 \wedge \sigma(c)$ and $c'_2 = c_1 \wedge \forall \mathcal{V}ar(c) - \mathcal{V}ar(l), \; \neg\sigma(c)$. In the case where the simplifying rule satisfies in addition $\mathcal{V}ar(c) \subseteq \mathcal{V}ar(l)$, we get $c_2 = c_1 \wedge \sigma(c)$ and $c'_2 = c_1 \wedge \neg\sigma(c)$.

Example 21.10 Let us consider the simplification of $(a * b = z \parallel a * z =^?_{AC} a * b)$ by matching $(x * a \to y \parallel x * y =^?_{AC} a * b)$ with the substitution $\sigma = \{x \mapsto b\}$. Then the simplification produces the constrained equality: $(y = z \parallel (a * z =^?_{AC} a * b) \wedge (b * y =^?_{AC} a * b))$ and the rest $(a * b = z \parallel (a * z =^?_{AC} a * b) \wedge (\forall y, \; b * y \neq^?_{AC} a * b))$.

• Restricting further to the case where c is an inequality constraint $(u >^?_\emptyset v)$ and $\mathcal{V}ar(c) \subseteq \mathcal{V}ar(l)$, $c'_2 = c_1 \wedge \neg\sigma(u >^?_\emptyset v)$. Assuming that $>$ is interpreted as a total ordering on terms, this negation can be transformed to a disjunction: $c'_2 = c_1 \wedge (\sigma(v >^?_\emptyset u) \vee \sigma(u =^?_\emptyset v))$.

Example 21.11 Let us consider the equality $(0 + (x + y) = x + (y + 0) \parallel \mathbf{T})$. It is simplified by $(x + 0 \to 0 + x \parallel x >^?_\emptyset 0)$ into: $(0 + (x + y) = x + (0 + y) \parallel y >^?_\emptyset 0)$ and $(0 + (x + y) = x + (y + 0) \parallel 0 >^?_\emptyset y \vee y =^?_\emptyset 0)$.

This last notion of simplification is used in the joinability test proposed in [Pet90]. A constrained critical pair $(p = q \parallel c)$ is joinable if for every instance σ satisfying c, $\sigma(p) \xrightarrow{*}_{CR,A,L_>} \circ \xleftrightarrow{*}_A \xleftarrow{*}_{CR,A,L_>} \sigma(q)$.

Joinability is ensured if sufficient conditions are satisfied [Pet90].

Proposition 21.1 A constrained equation $(p = q \parallel c)$ is joinable if one of the following cases occurs: either $p \xleftrightarrow{*}_A q$; or c is equivalent to $\mathbf{F}$; or c is equivalent to $c' \wedge \theta$ and $(\theta(p) = \theta(q) \parallel c')$ is joinable ; or $(p = q \parallel c)$ can be simplified to a set of constrained equations, each of which being joinable.

This joinability test is applied to constrained critical pairs and additional critical equations $(l = r \parallel \neg c)$ systematically added for each $(l = r \parallel c)$. This leads to the following result.

Theorem 21.3 *[Pet90] $(CR, A, L_>)$ is Church-Rosser modulo A on ground terms iff every critical equation is joinable.*

Example 21.12 Using the method described in [Pet90], Church-Rosser sets of constrained rules for semi-lattices, boolean algebras and ternary boolean algebras. For instance, for a semi-lattice, the set of constrained rewrite rules below is

$$\begin{array}{llll}
(x * y) * z & \to & x * (y * z) & \parallel \;\; \mathbf{T} \\
x * y & \to & y * x & \parallel \;\; x >^?_\emptyset y \\
x * (y * z) & \to & y * (x * z) & \parallel \;\; x >^?_\emptyset y \\
x * x & \to & x & \parallel \;\; \mathbf{T} \\
x * (x * y) & \to & x * y & \parallel \;\; \mathbf{T}
\end{array}$$

has two constrained critical pairs between the two first rules

$$\begin{array}{llll}
x * (y * z) & = & (y * x) * z & \parallel \;\; x >^?_\emptyset y \\
x * (y * z) & = & z * (x * y) & \parallel \;\; x * y >^?_\emptyset z
\end{array}$$

that are proved joinable.

The technique of constrained completion modulo A has been successfully applied to theories involving commutativity, associativity and identity axioms (*ACIdentity* for short).

Example 21.13 A set of constrained rules with the Church-Rosser modulo *ACIdentity* property, for a commutative group is computed with this method from the only initial equality $(x + (-x) = 0 \parallel \mathbf{T})$. Axioms are

$$\begin{array}{rcl} x + y & = & y + x \\ (x + y) + z & = & x + (y + z) \\ x + 0 & = & x \end{array}$$

The produced set of constrained rewrite rules is

$$\begin{array}{lcll} x + (-y) + y & \to & x & \parallel \ \mathbf{T} \\ -(-x) & \to & x & \parallel \ \mathbf{T} \\ -(x + y) & \to & (-x) + (-y) & \parallel \ x \neq^?_{ACIdentity} 0 \wedge y \neq^?_{ACIdentity} 0 \end{array}$$

Other examples of theories are handled in [BPW89]: commutative ring with unit, Boolean algebra, group or ring homomorphism, distributive lattice. Comparisons of step size of derivations, run times, number and symmetry of critical pairs are also considered.

21.5 Conclusions

We have shown how constraints help to describe completion procedures in a precise way and to improve their efficiency. Although we focussed here on rewriting and completion with constraints, the formalism of deduction with constraints goes much beyond the pure equational case [KKR90]. Several examples have been developed in first-order logic. The notion of constrained resolution has already been studied for equational constraints in the empty theory. Caferra and Zabel [CZ90] use it in a procedure which is able to refute or to generate some kind of models when they exist. Buntine and Bürckert [BB89] also noticed that constrained resolution improve the efficiency of theorem provers, for instance by preventing from the generation of tautologies, an idea further developed in [Bür90]. Several other examples have been developed and implemented in equational or first-order logic, using in particular the systems Datac [Vig93] and Saturate [NR94]. The remaining important problems are to incorporate a powerful simplification mechanism and to study its interaction with constraint propagation. A special investigation effort has to be put on the design of strategies for managing deduction rules in an efficient way.

Chapter 22

Proofs by induction

22.1 Introduction

Another class of equational theorems is considered in this chapter, namely those theorems that hold in the initial algebra of an equational theory. An additional difficulty encountered in this framework is that the proof of such theorems needs an induction principle. A surprising feature of the completion procedure is that it is powerful enough to be used as an inductive theorem prover by applying the *proof by consistency method* [KM87]. The methods presented here do not require the explicit expression of an induction principle. This is why they are called *implicit induction methods*.

The general principle is as follows: Let R be a ground convergent rewrite system that presents the theory E. An equational theorem $(t = t')$ holds in the initial algebra of E iff the completion of $R \cup \{t = t'\}$ does not derive an inconsistency. Roughly speaking, the discovery of an inconsistency witnesses the fact that the two sets of equalities R and $R \cup \{(t = t')\}$ do not define the same initial algebra. According to different authors, the detection of an inconsistency may change. In 1980, in work of Musser [Mus80] and independently Goguen [Gog80], an inconsistency is the equality between boolean terms ($true = false$). This assumes an axiomatization of booleans and an equality predicate eq_s for each sort of data s. This also needs that any expression $eq_s(t, t')$ reduces either to $true$ or to $false$.

In 1982, Huet and Hullot showed how the notion of constructors (function symbols on which data are built) allows dropping the requirements about the equality predicate [HH82], provided there is no relation between constructors. An inconsistency is then just an equality between terms built only with constructors.

In 1986, Jouannaud and Kounalis, following previous ideas of Dershowitz [Der83b] and Rémy [Rém82], introduced the important concept of *ground reducibility* [JK86b], which allows handling relations between constructors. In their approach, an inconsistency is an equality between two irreducible ground terms.

In 1988, Bachmair adapted an unfailing completion procedure to produce a procedure for proof by consistency that is refutationally complete [Bac88]. His method avoids two drawbacks of the previous ones: first, it does not fail with a non-orientable equality, which was a case for which nothing could be concluded in other methods. Second, inductive proofs are not concerned with the production of a (ground) Church-Rosser set of rules, which was obtained as a side effect in previous approaches. Following Fribourg [Fri86], Bachmair proposed a linear proof method that considerably reduces the number of equalities to be deduced.

A lot of work has been devoted to decreasing the number of necessary superpositions and avoiding divergence of the above mentionned procedures, including [Küc89, BK89, Fri86, Gra89, HK88]. A comparison between completion techniques and direct induction as performed in the Boyer-Moore or in the CLAM systems is done in [BBH92].

In 1990, another major step was performed by formalizing the rewriting induction method that does not require confluence nor ground confluence of the rewrite system [KR90, Red90]. The method gives a sufficient condition for proving inductive theorems. In case where confluence is retained, the method is refutationally complete.

This chapter gives an account of several approaches to automated inductive theorem proving.

22.2 Many-sorted specifications

We slightly generalise the algebraic framework of previous chapters by introducing the notion of sorts for classifying terms, as in Section 2.5 of Chapter 2.

Let us briefly remind the basic concepts. A *many-sorted signature* Σ is given by a (denumerable) set of *sorts* S and a (denumerable) set of function symbols $\mathcal{F}$ with ranks. A function f with *arity* $w = s_1 \dots s_n \in S^*$ and *co-arity* (or value sort) s is written $f : w \mapsto s$. Variables are also sorted and $x : s$ means that variable x has sort s. $\mathcal{X}_s$ denotes a denumerable set of variables of sort s and $\mathcal{X} = \bigcup_{s \in S} \mathcal{X}_s$ is the set of many-sorted variables. Many-sorted terms are built on many-sorted signatures and classified according to their sorts. The set of terms of sort s, denoted $\mathcal{T}(\Sigma, \mathcal{X})_s$ is the smallest set containing $\mathcal{X}_s$ and any constant $a : \epsilon \mapsto s$ such that $f(t_1, \dots, t_n)$ is in $\mathcal{T}(\Sigma, \mathcal{X})_s$ whenever $f : s_1 \dots s_n \mapsto s$ and $t_i \in \mathcal{T}(\Sigma, \mathcal{X})_{s_i}$ for $i \in [1..n]$. The set of *many-sorted terms* $\mathcal{T}(\Sigma, \mathcal{X})$ is the family $\{\mathcal{T}(\Sigma, \mathcal{X})_s | s \in S\}$.

Many-sorted algebras have carriers corresponding to each sort and operations with sorted arguments. Given a many-sorted signature Σ, a Σ-algebra $\mathcal{A}$ consists of a family $\{A_s | s \in S\}$ of subsets of $\mathcal{A}$, called the *carriers* of $\mathcal{A}$, and a family of operations $f_{\mathcal{A}} : A_{s_1} \times \dots \times A_{s_n} \mapsto A_s$, associated to each function $f \in \Sigma$ such that $f : s_1, \dots, s_n \mapsto s$.

Substitutions are defined as mappings σ from sorted variables to sorted terms such that if $x : s$ then $\sigma(x) \in \mathcal{T}(\Sigma, \mathcal{X})_s$. They induce Σ-homomorphisms on the Σ-algebra of many-sorted terms. Equalities are built from many-sorted terms.

A *specification*, denoted $SP = (\Sigma, E)$ is given by a many-sorted signature Σ, and a set E of universally quantified equalities $(\forall X, t = t')$ where $\mathcal{V}(t) \cup \mathcal{V}(t') \subseteq X$. (The quantification may be omitted when $X = \mathcal{V}(t) \cup \mathcal{V}(t')$).

A specification $SP = (\Sigma, E)$ actually describes a class of algebras, namely the class of Σ-algebras satisfying the equalities E, denoted $ALG(SP)$. $ALG(SP)$ with SP-homomorphisms is a category also denoted by $ALG(SP)$.

Deduction rules for equational deduction given in Figure 2.1 of Chapter 2 generalize to the many-sorted framework provided there is no *empty* sort. A precise analysis of the possible problems that may arise when this hypothesis is not satisfied can be found in [MG85]. So in the following, only models with non-empty sorts are considered. This means that for any model $\mathcal{A}$ and any $s \in S$, $\mathcal{A}_s$ is a non-empty set.

Let $\overset{*}{\longleftrightarrow}_E$ denote the replacement of equals by equals on $\mathcal{T}(\Sigma, \mathcal{X})$ which is correct and complete for deduction in $ALG(SP)$:

$$t \overset{*}{\longleftrightarrow}_E t' \text{ iff } \forall \mathcal{A} \in ALG(SP), \mathcal{A} \models (\forall X, t = t').$$

The class $ALG(SP)$ has an initial algebra denoted by $\mathcal{T}(\Sigma)/E$ or by $\mathcal{T}_{SP}$. $\mathcal{T}(\Sigma)/E$ is built as the quotient algebra of the ground term algebra $\mathcal{T}(\Sigma)$ by the congruence $\overset{*}{\longleftrightarrow}_E$ generated by E.

22.3 Inductive theorems and consistency

Inductive theorems are equalities that hold in the initial algebra, which means that for all assignments of variables by ground terms, we get equal ground terms.

Definition 22.1 An equality $(t = t')$ is an *inductive theorem of* E if for any ground substitution σ, $\sigma(t) \overset{*}{\longleftrightarrow}_E \sigma(t')$.

The *inductive theory* of E, denoted by $ITh(E)$, is the class of equalities $(t = t')$ valid in $\mathcal{T}(\Sigma)/E$.

Notation: This is denoted $(t = t') \in ITh(E)$ or $\mathcal{T}(\Sigma)/E \models (t = t')$ or $E \models_{ind} (t = t')$.

The inductive theory of E includes the equational theory of E, but this inclusion is in general strict. For instance, associativity and commutativity of $+$ on integers are not equational consequences of the theory E that defines $+$ on integers built with the constant 0 and the successor function s. Equational theories that coincide with their inductive theories are called *ω-complete* and are studied in [Hen77, Tar68, Tay79, Hee86, LLT90].

In what follows, we assume that E is presented by a *ground convergent* rewrite system R. The results also hold for an ordered rewrite system $(R, >)$ which is ground Church-Rosser with respect to the reduction ordering $>$.

Remember that this means that R satisfies the following properties:

1. the congruences $\overset{*}{\longleftrightarrow}_E$ and $\overset{*}{\longleftrightarrow}_R$ coincide on ground terms,

2. any rewriting sequence issued from a ground term t terminates,

3. for any ground terms t and t', $t \overset{*}{\longleftrightarrow}_R t'$ iff $t \overset{*}{\longrightarrow}_R t'' \overset{*}{\longleftarrow}_R t'$ for some ground term t''.

Then different characterizations of an inductive theorem can be given:

Proposition 22.1 Assume that E is presented by a ground convergent rewrite system R. The following statements are equivalent:

- $(t = t')$ is an inductive theorem of E.
- for any ground substitution σ, $\sigma(t) \overset{*}{\longleftrightarrow}_E \sigma(t')$.
- for any ground substitution σ, $\sigma(t) \overset{*}{\longleftrightarrow}_R \sigma(t')$.
- for any ground substitution σ, $\sigma(t) \downarrow_R = \sigma(t') \downarrow_R$.
- $\overset{*}{\longleftrightarrow}_R$ and $\overset{*}{\longleftrightarrow}_{R \cup \{t=t'\}}$ coincide on $\mathcal{T}(\Sigma)$.
- $\mathcal{T}(\Sigma)/ \overset{*}{\longleftrightarrow}_R$ and $\mathcal{T}(\Sigma)/ \overset{*}{\longleftrightarrow}_{R \cup \{t=t'\}}$ are isomorphic.

Any of these equivalent statements will be freely used in what follows. We shall use the word *conjecture* to denote a theorem to prove or to refute in the initial algebra.

Definition 22.2 A *conjecture* is an implicitely universally quantified equality. A set of conjectures C is *consistent* with R (or E) if all equalities of C are inductive theorems of R (or E). Otherwise C is said *inconsistent*.

Consistency in this sense is not decidable. In order to find an operational concept, let consider necessary conditions for being an inductive theorem:

Proposition 22.2 Let $>$ be a reduction ordering, R a ground convergent rewrite system included into $>$ and $(t = t')$ an inductive theorem of R. Then

1. if $t > t'$, then for any ground substitution σ, $\sigma(t)$ is R-reducible.
2. for any ground substitution σ, such that $\sigma(t) \neq \sigma(t')$, either $\sigma(t)$ or $\sigma(t')$ is R-reducible.

Proof:
1. Assume that $t > t'$ and $\sigma(t)$ is R-irreducible. Then for any ground substitution σ, $\sigma(t) \overset{*}{\longleftrightarrow}_R \sigma(t')$, so $\sigma(t) \overset{*}{\longleftarrow}_R \sigma(t')$ and $\sigma(t') \geq \sigma(t)$. This contradicts the fact that $t > t'$ implies $\sigma(t) > \sigma(t')$.
2. By definition, for any ground substitution σ, $\sigma(t) \downarrow_R = \sigma(t') \downarrow_R$. If there exists a ground substitution σ such that $\sigma(t)$ and $\sigma(t')$ are irreducible and $\sigma(t) \neq \sigma(t')$, this again yields a contradiction.

□

The concept of ground reducibility is deduced from this result.

22.4 Ground reducibility

Ground reducibility is first defined for a term. It amounts to check reducibility of all ground instances.

Definition 22.3 Given a ground convergent rewrite system R, a term t is *ground reducible* with R if all its ground instances are R-reducible.

The notion of ground reducibility then extends to equalities.

Definition 22.4 Given a ground convergent rewrite system R, an equality $(t = t')$ is *ground reducible* with R if for any ground substitution σ such that $\sigma(t) \neq \sigma(t')$, either $\sigma(t)$ or $\sigma(t')$ is R-reducible.

Example 22.1 Consider the following signature:

$$\begin{array}{llll} sort & Int & & \\ 0 : & & \mapsto & Int \\ succ : & Int & \mapsto & Int \end{array}$$

and the rewrite system

$$succ(succ(0)) \quad \rightarrow \quad 0.$$

The term $succ(succ(x))$ is ground reducible, but the term $succ(x)$ is not.

Proposition 22.3 [KNZ87, Pla85] Ground reducibility is decidable for finite rewrite systems.

However deciding ground reducibility is in exponential time even for left-linear rules. Algorithms for deciding ground reducibility in the case of left-linear rules have been proposed, for instance in [JK86b, KNZ87, Bun87, Kuc88, NW83]. The general case is considered in [Der85a] and in [Kou85]. The former defines a *test set* by instantiating the generic term $f(x_1, ..., x_n)$ by ground substitutions in a finite set. The number of substitutions to check is bounded by a number that depends on the maximal depth of left-hand sides. The latter constructs a smaller test set, computed by repeated unification of $f(x_1, ..., x_n)$ with the left-hand sides. Another decision method is obtained by reducing ground reducibility to the emptiness problem of the language describing the ground normal forms and produced by a conditional tree grammar [Com88a]. Ground reducibility for a class rewrite system R/E is undecidable when E is a set of associative and commutative axioms [KNZ87] but is decidable when R is left-linear [JK86a].

In order to illustrate a test for ground reducibility, we restrict to the simpler case of left-linear rules and describe the test given in [JK86b].

The goal is to exhibit a finite set of substitutions $\mathcal{S}$ such that a term is ground reducible iff all its instances by substitutions in $\mathcal{S}$ are reducible. In the case of left-linear rules, the idea to construct $\mathcal{S}$ is that left-hand sides of rules have a finite depth which bounds the number of substitutions to be tested.

Some preliminary notions are needed first. The length of a term t is the maximal size of positions ω in $\mathcal{D}om(t)$. Let $d = depth(R)$ be the maximal depth of left-hand sides of rules in R.

The top of a term t at depth i is a term defined by:

$top(t, i) = t$ if $depth(t) < i$

$top(f(t_1, ..., t_n), 0) = f(x_1, ..., x_n)$ where x_i are new variables

$top(f(t_1, ..., t_n), i) = f(top(t_1, i-1), ..., top(t_n, i-1))$

for any symbol $f \in \mathcal{F}$.

Example 22.2

$$\begin{array}{rcl}
top(f(g(a,b), h(h(c))), 0) & = & f(x_1, x_2) \\
top(f(g(a,b), h(h(c))), 1) & = & f(g(x_1, x_2), h(x_3)) \\
top(f(g(a,b), h(h(c))), 2) & = & f(g(a,b), h(h(x))) \\
top(f(g(a,b), h(h(c))), 3) & = & f(g(a,b), h(h(c)))
\end{array}$$

Given a set of rewrite rules R of depth d, let

$$\mathcal{S}(R) = \{\ top(t_0, d) \mid t_0 \text{ is an } R\text{−irreducible ground term }\}$$

For practical reasons, variables in terms of $\mathcal{S}(R)$ are assumed distinct, which is always possible by renaming them: $\forall t, t' \in \mathcal{S}(R)$, $\mathcal{V}(t) \cap \mathcal{V}(t') = \emptyset$.

Theorem 22.1 *[JK86b] A term t is ground reducible by a left-linear rewrite system R iff all its instances*

$$\{\sigma(t) \mid \forall \sigma : \mathcal{V}(t) \mapsto \mathcal{S}(R)\},$$

obtained by substitution of terms in $\mathcal{S}(R)$ to variables of t, are reducible by R.

Proof: [JK86a, JK89].

- Assume that all instances of t obtained by substitution of terms in $\mathcal{S}(R)$ to variables of t, are reducible by R. For any ground substitution σ', if σ' is not R-normalized, then $\sigma'(t)$ is R-reducible. Otherwise, let define for any variable x of t, $\sigma(x) = top(\sigma'(x), d)$. Then $\sigma \in \mathcal{S}(R)$ and $\sigma(t)$ is R-reducible by hypothesis. Since σ' is an instance of σ, $\sigma'(t)$ is also R-reducible.
- Assume now that t is ground reducible. Given $\sigma \in \mathcal{S}(R)$, let us define for any variable x_i of t, $t_i = \sigma(x_i)$. Then there exists t'_i an R-irreducible instance of t_i such that $t_i = top(t'_i, d)$. Finally let us define σ' such that $\sigma'(x_i) = t'_i$. Since t is ground reducible, $\sigma'(t)$ is R-reducible by a rewrite rule $l \rightarrow r$, at some non-variable position ω in t because σ' is normalized. But for any position v in l, the top symbols of $l_{|v}$, $\sigma(t)_{|\omega.v}$ and $\sigma'(t)_{|\omega.v}$ are the same, because of the definition of d and the fact that $t_i = top(t'_i, d)$. Now let W be the set of variable occurrences in l and for any variable y at occurrence $v \in W$ define $\sigma''(y) = \sigma(t)_{|\omega.v}$. Note that σ'' is well-defined because y has only one occurrence in l. So $\sigma(t)_{|\omega} = \sigma''(l)$ and so $\sigma(t)$ is R-reducible.

□

The set $\mathcal{S}(R)$ is computed from the limit of a stationary sequence of sets $\mathcal{S}_i$ defined as follows:

$$\mathcal{S}_i = \{ \ top(t_0, d) \mid \forall t_0 \ R-irreducible\ ground\ term\ s.t.\ depth(t_0) \leq i \ \}$$

Then $\mathcal{S}(R) = S_k$ as soon as $S_k = S_{k+1}$ for some k.

Example 22.3 Consider the following signature:

$$\begin{array}{llcl} sort & Int & & \\ 0: & & \mapsto & Int \\ succ: & Int & \mapsto & Int \\ +: & Int \ \ Int & \mapsto & Int \end{array}$$

Let R be the set of left-linear rules:

$$\begin{array}{rcl} \forall x: Int,\ x+0 & \rightarrow & x \\ \forall x,y: Int,\ x+succ(y) & \rightarrow & succ(x+y). \end{array}$$

The depth of R is $d = 2$.

$$\begin{array}{rcl} S_0 & = & \{0\} \\ S_1 & = & \{0, succ(0)\} \\ S_2 & = & \{0, succ(0), succ(succ(x_1))\} \\ S_3 & = & S_2 \end{array}$$

In order to check that the term $(x+y)+z$ is ground reducible, every ground substitution from $\{x, y, z\}$ to $\{0, succ(0), succ(succ(x_1))\}$ has to be applied to $(x+y)+z$ and it is easy to check that each ground term so obtained is reducible.

A more general, but more complex test for checking ground reducibility without the restriction of left-linearity on rules can be found in [Kou90].

22.5 Inductive completion

A first approach to use ground reducibility in the automatization of inductive theorem proofs is based on the idea that enriching R with rules whose left-hand sides are ground reducible does not change the normal forms for R.

Lemma 22.1 Let $(l \rightarrow r)$ be a rewrite rule such that l is ground reducible with R. Then a ground term t is in normal form for R iff t is in normal form for $R \cup \{l \rightarrow r\}$.

Proof: If t is a ground term in normal form for R but not for $R \cup \{l \rightarrow r\}$, t has a subterm that is a ground instance of l, so that is reducible by R, which is impossible.

The converse is obvious. □

It follows that R and $R \cup \{l \rightarrow r\}$ have the same inductive theory.

Theorem 22.2 *Let R be a ground convergent rewrite system and $(l \rightarrow r)$ be a rewrite rule such that l is ground reducible with R. If $R \cup \{l \rightarrow r\}$ is ground convergent, then $(l = r)$ is an inductive theorem of R.*

Proof: Let $R' = R \cup \{l \rightarrow r\}$. For any ground substitution σ, $\sigma(l)\downarrow_{R'} = \sigma(r)\downarrow_{R'}$. Since R is contained into R' and R' is confluent, $\sigma(l) \xrightarrow{*}_R \sigma(l)\downarrow_R \xrightarrow{*}_{R'} \sigma(l)\downarrow_{R'}$ and $\sigma(r) \xrightarrow{*}_R \sigma(r)\downarrow_R \xrightarrow{*}_{R'} \sigma(r)\downarrow_{R'}$. But, using Lemma 22.1, $\sigma(l)\downarrow_R = \sigma(l)\downarrow_{R'}$ and $\sigma(r)\downarrow_R = \sigma(r)\downarrow_{R'}$. So $\sigma(l) \xleftrightarrow{*}_R \sigma(r)$. □

In general there is no reason for $R \cup \{l \rightarrow r\}$ to be confluent. A completion procedure must be applied. The only difference with standard completion is to check that when a rule is added, its left-hand side has to be ground reducible with the initial convergent set of rewrite rules R_0.

$$\begin{array}{llll}
\textbf{Orient} & P \cup \{p = q\}, R & \mapsto\!\!\!\!\rightarrow & P, R \cup \{p \rightarrow q\} \\
 & & & \text{if } p > q \ \&\ p \text{ ground reducible with } R_0 \\
\textbf{Deduce} & P, R & \mapsto\!\!\!\!\rightarrow & P \cup \{p = q\}, R \\
 & & & \text{if } (p, q) \in CP(R) \\
\textbf{Simplify} & P \cup \{p = q\}, R & \mapsto\!\!\!\!\rightarrow & P \cup \{p' = q\}, R \\
 & & & \text{if } p \rightarrow_R p' \\
\textbf{Delete} & P \cup \{p = p\}, R & \mapsto\!\!\!\!\rightarrow & P, R \\
\textbf{Compose} & P, R \cup \{l \rightarrow r\} & \mapsto\!\!\!\!\rightarrow & P, R \cup \{l \rightarrow r'\} \\
 & & & \text{if } r \rightarrow_R r' \\
\textbf{Collapse} & P, R \cup \{l \rightarrow r\} & \mapsto\!\!\!\!\rightarrow & P \cup \{l' = r\}, R \\
 & & & \text{if } l \rightarrow_R^{g \rightarrow d} l' \ \&\ l \rightarrow r >> g \rightarrow d
\end{array}$$

Figure 22.1: Standard inductive completion

22.5.1 Transition rules for inductive completion

Let P be a set of equalities (quantified pairs of terms), R_0 the initial ground convergent rewrite system, R be the current rewrite system, and $>$ a reduction ordering that contains R_0. The inductive completion procedure is expressed using the set of transition rules $\mathcal{IC}$ presented in Figure 22.1.

Note that R_0 is used to check ground reducibility, but critical pairs are computed with R.

The correctness result presented below relies on the two following lemmas.

Lemma 22.2 If $(P_0, R_0) \mapsto\!\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\!\rightarrow \ldots.$ is a derivation such that

- $P_\infty = \emptyset$, R_∞ is reduced and
- $CP(R_\infty)$ is a subset of P_*

then for any ground terms t, t', $t \overset{*}{\longleftrightarrow}_{R_0 \cup P_0} t'$ implies $t \downarrow_{R_\infty} = t' \downarrow_{R_\infty}$.

Proof: As for the proof of the completion procedure, it is shown that any equational proof of $(t = t')$ is reduced to a rewrite proof using R_∞. □

The next lemma proves that ground terms in normal form for R_0 remain in normal form at each step of the inductive completion procedure. This justifies the restriction to R_0 in the test of ground reducibility of the left-hand side of a new rule.

Lemma 22.3 For any ground term t, t is R_∞-reducible iff t is R_0-reducible.

Proof: Assume first that t is R_0-reducible to t'. Then $t > t'$. Using Lemma 22.2, $t \downarrow_{R_\infty} = t' \downarrow_{R_\infty} = t''$. But $t \neq t''$, otherwise $t \leq t'$, which is impossible.

If t is R_∞-reducible, either t is reducible by a persisting rule of R_0, or t is reducible by a rule $(l \rightarrow r)$ added during the inductive completion procedure. But then l is ground reducible using R_0 and thus t contains a subterm R_0-reducible. □

If a successful derivation starting from (P_0, R_0) is found, R_0 and R_∞ define the same normal forms on ground terms; this implies that each equality of P_0 holds. Moreover if the process generates a rule whose left-hand side is not ground reducible by R_0, this gives the inconsistency.

Theorem 22.3 *Let R_0 be the initial ground convergent rewrite system presenting the theory E, P_0 be the set of conjectures to be proved, and $>$ be a reduction ordering that contains R_0. If $(P_0, R_0) \mapsto\!\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\!\rightarrow \ldots.$ is a derivation such that*

- *$P_\infty = \emptyset$, R_∞ is reduced and*
- *$CP(R_\infty)$ is a subset of P_**

then R_∞ is Church-Rosser on ground terms, R_∞ is terminating and P_0 is consistent with R_0. If $(P_0, R_0) \mapsto\!\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\!\rightarrow \ldots.(P_i, R_i)$ is a derivation such that P_i contains an equality $l = r$ with $l > r$ and l non ground reducible with R_0, then P_0 is inconsistent with R_0.

Proof: For any equality $(g = d)$ in P_0 and any ground substitution σ, $\sigma(g) \overset{*}{\longleftrightarrow}_{P_0} \sigma(d)$. Using Lemma 22.3, both $t = \sigma(g) \downarrow_{R_0}$ and $t' = \sigma(d) \downarrow_{R_0}$ are also in R_∞-normal form and using Lemma 22.2, $t \downarrow_{R_\infty} = t' \downarrow_{R_\infty}$. Thus $t = t'$ and $\sigma(g) \overset{*}{\longleftrightarrow}_{R_0} \sigma(d)$.

If an equality $(l = r)$, with $l > r$ and l non ground reducible with R_0, is generated, then there exists a ground substitution σ such that $\sigma(l)$ is R_0-irreducible. By Proposition 22.2, $(l = r)$ is not an inductive theorem. □

22.5.2 An inductive completion procedure

An inductive completion procedure can be derived from the standard completion procedure. An *inductive completion procedure* is a program that takes a finite set of equalities P_0, a convergent rewrite system R_0 and a reduction ordering $>$ that contains R_0, and that generates from (P_0, R_0) a derivation $(P_0, R_0) \mapsto\!\!\!\!\rightarrow (P_1, R_1) \mapsto\!\!\!\!\rightarrow ....$, using the transition rules in $\mathcal{IC}$.

An inductive completion procedure is presented in Figure 22.2.

```
PROCEDURE INDUCTIVE-COMP (P, R, >)
IF P is not empty
THEN choose a pair (p,q) in P ; P := P-{(p,q)};
     p':=R-normal form(p); q':=R-normal form(q);
     CASE p' = q'  THEN  R := INDUCTIVE-COMP(P, R, >)
          p' > q' IF p' ground reducible with R0
                  THEN  l:=p'; r:=q';
                        (P,R) := SIMPLIFICATION(P, R, l -> r);
                        R := INDUCTIVE-COMP(P, R U {l -> r}, >)
                  ELSE STOP with DISPROOF
                  END IF
          q' > p' IF q' ground reducible with R0
                  THEN l:=q'; r:=p';
                       (P,R) := SIMPLIFICATION(P, R, l -> r);
                       R := INDUCTIVE-COMP(P, R U {l -> r}, >)
                  ELSE STOP with DISPROOF
                  END IF
          ELSE STOP with FAILURE
     END CASE;
ELSE IF all rules in R are marked
     THEN RETURN R; STOP with SUCCESS
     ELSE Choose an unmarked rule l -> r fairly;
          P := CRITICAL-PAIRS (l -> r,R);
          Mark the rule l -> r in R;
          R := INDUCTIVE-COMP(P,R,>)
     END IF
END IF
END INDUCTIVE-COMP
```

Figure 22.2: A standard completion procedure

Example 22.4 Let consider the following specification of integers modulo 2 defined by the signature:

$$\begin{array}{llll} sort & Int & & \\ 0: & & \mapsto & Int \\ succ: & Int & \mapsto & Int \end{array}$$

and the rewrite system R_0:

$$succ(succ(0)) \rightarrow 0.$$

Assume that the equality to be proved is

$$\forall x : Int,\ succ(succ(x)) = x.$$

The inductive completion procedure acts as follows: The equality is oriented into a rule

$$succ(succ(x)) \rightarrow x$$

alfter checking that the term $succ(succ(x))$ is ground reducible. Since $\mathcal{S}(R_0) = \{0, succ(0)\}$, this is checked by verifying that the two terms $\{succ(succ(0)), succ(succ(succ(0)))\}$ are both reducible by R_0. Then the new rule is added, simplifies the rule in R_0 and the process terminates, because there is only one critical pair of the new rule on itself that is already convergent.

In order to improve efficiency of the inductive completion procedure and to avoid some cases of divergence, it is sometimes possible to take advantage of the existence of constructors, that is a subset $\mathcal{C}$ of Σ such that $\mathcal{T}(\mathcal{C})$ is exactly the set of normal forms of $\mathcal{T}(\Sigma)$. As a consequence, note that any term of $\mathcal{T}(\mathcal{C})$ is irreducible by R.

Definition 22.5 A function symbol $f \in \Sigma$ of arity n is a *constructor* if $f(x_1, ..., x_n)$ is not ground reducible by R.

A function symbol $f \in \Sigma$ of arity n is a *defined function* if $f(x_1, ..., x_n)$ is ground reducible by R.

Within this context, checking ground reducibility becomes trivial: a term t is inductively reducible iff it contains a defined function symbol.

Existence of constructors may be taken into account during completion by adding to the set of transition rules $\mathcal{IC}$ presented in Figure 22.1, the two transition rules in Figure 22.3.

Contradict
$P \cup \{f(s_1, ..., s_n) = g(t_1, ..t_m)\}, R$
$\Vdash\!\!\twoheadrightarrow$
$Disproof$
if $f \in \mathcal{C}$ & $f(s_1, ..., s_n) > g(t_1, ..t_m)$
Decompose
$P \cup \{f(s_1, ..., s_n) = f(t_1, ..t_n)\}, R$
$\Vdash\!\!\twoheadrightarrow$
$P \cup \{(s_i = t_i), i = 1, ..., n\}, R$
if $f \in \mathcal{C}$

Figure 22.3: Optimization rules

However, even with this kind of improvement, the inductive completion procedure suffers from the following drawbacks.

- Inductive theorems, such as commutativity (of addition on integers, for instance), for which no orientation is possible are not handled by this method. The inductive completion procedure terminates with failure and nothing can be concluded about validity of the conjectures. Note however that some solutions have been proposed in [Kir84b, JK86b] to handle anyway such non-orientable theorems, using class rewriting and completion modulo a set of equalities.

- When interested in proofs of inductive theorems, there is no need to achieve the Church-Rosser property on ground terms. This means that very often too many critical pairs are computed, much more than necessary. In some cases, inefficiency and even divergence can result from that overhead.

- Moreover the set of rules R_∞, convergent on ground terms, returned by the procedure is in general different from R_0. For repeated applications of inductive proofs, this may be disturbing.

22.6 Inductive proof by consistency

By contrast to the previous method, the proof by consistency procedure, presented hereafter, provides several advantages.

- It avoids failure, because it never attempts to orient equalities into rewrite rules. In this way, it is close from an unfailing completion method.

- It keeps the original set of rewrite rules unmodified.
- It avoids many unnecessary critical pairs computation, and in this respect can be called a *linear procedure*, such as in [Fri86].
- It gives the possibility to add some lemmas, for instance given by the user, or coming from another proof. Such adjunctions may help the termination of the process.

The method developed hereafter consists of transforming a set of initial conjectures and trying to generate a provable inconsistency. Intuitively, a provable inconsistency is a minimal proof which implies an inconsistency. The definition of a provable inconsistency comes from Proposition 22.2. This is an operational notion, defined from ground reducibility.

Definition 22.6 Let $>$ denote a reduction ordering that contains R. A set of equalities C is *provably inconsistent* if it contains an equality $(s = t)$ which satisfies either $s > t$ and s is not ground reducible, or $(s = t)$ is not ground reducible.

The property of being provably inconsistent is decidable because ground reducibility is decidable.

Clearly, if C is provably inconsistent, it is inconsistent with R: if C is provably inconsistent, it contains a provably inconsistent equality $(s = t)$. According to Definition 22.6, there exists a ground substitution σ, that can be assumed R-normalized without lost of generality, such that

- either $\sigma(s) \longleftrightarrow_C \sigma(t) \xrightarrow{*}_R t'$ with $\sigma(s)$ irreducible and $\sigma(s) \neq t'$ since $\sigma(s) > \sigma(t) \geq t'$,
- or $\sigma(s) \longleftrightarrow_C \sigma(t)$ with $\sigma(s)$ and $\sigma(t)$ irreducible and $\sigma(s) \neq \sigma(t)$.

In both cases, $s = t$ is not an inductive theorem of R.

If C is inconsistent, but not provably inconsistent, there exists an equality $(s = t) \in C$ and a R-normalized substitution σ such that

$$s' \xleftarrow{*}_R \sigma(s)[\sigma(r)] \leftarrow_R \sigma(s)[\sigma(l)] \longleftrightarrow_C \sigma(t) \xrightarrow{*}_R t'$$

with $s' \neq t'$. This proof can be transformed into a smaller proof (for some ordering defined later on)

$$s' \xleftarrow{*}_R \sigma(s)[\sigma(r)] \longleftrightarrow_{C'} \sigma(t) \xrightarrow{*}_R t'$$

with $s' \neq t'$, in which a reduction step has been eliminated and the set of conjectures has been enriched by an equality $(\tau(s[r]) = \tau(t))$, where τ is a most general unifier of l with some subterm of s. This transformation of C into C' is done via a critical pair computation of rules of R into conjectures of C.

Notation: Let $CP(R, C)$ denote the set of critical pairs obtained by superposition of a rule in R into an equality of C.

22.6.1 Transition rules for proof by consistency

Let C be a set of conjectures (equalities), R the implicit ground convergent rewriting system, L be a set of inductive lemmas, and $>$ a reduction ordering that contains R. The proof by consistency procedure is expressed by the set $\mathcal{CP}$ of transition rules presented in Figure 22.4.

Thanks to transition rule **Induce**, sets of axioms like commutativity or associativity and commutativity, can be put in L, once they are proved valid, either by this method or by another one. They can then be used in the simplification process, through application of the transition rule **Simplify**, without the need of equational matching or unification.

Note also that, once a lemma has been proved valid, it can be deleted from C using **Delete** and put in L using **Induce**. Then it may be used to simplify other conjectures.

As usual, let C_* denote the union of generated conjectures and C_∞ denote the set of persisting conjectures.

A first result states that consistency is preserved by applying each transition rule.

Proposition 22.4 Let $(L, C) \mapsto\!\!\!\!\rightarrow (L', C')$. Then C is consistent with R iff C' is consistent with R.

Proof: Let $(s = t)$ be an equality in the difference $C - C'$ or $C' - C$. We need to prove that for any ground substitution σ, $\sigma(s) \downarrow_R = \sigma(t) \downarrow_R$. Let us examine the different transition rules:

1. Deduce: $C' - C = \{(p = q) \in CP(R, C)\}$, so $\exists u,\ p \leftarrow_R u \rightarrow_C q$. Then for any ground substitution σ, $\sigma(p) \leftarrow_R \sigma(u) \rightarrow_C \sigma(q)$. So $\sigma(p) \downarrow_R = \sigma(u) \downarrow_R = \sigma(q) \downarrow_R$.

$$\begin{array}{llcl}
\textbf{Deduce} & L, C & \mapstochar\twoheadrightarrow & L, C \cup \{p = q\} \\
 & & & \text{if } (p, q) \in CP(R, C) \\
\textbf{Induce} & L, C & \mapstochar\twoheadrightarrow & L \cup \{p = q\}, C \\
 & & & \text{if } (p = q) \in ITh(R) \\
\textbf{Delete} & L, C \cup \{p = q\} & \mapstochar\twoheadrightarrow & L, C \\
 & & & \text{if } (p = q) \in ITh(R) \\
\textbf{Simplify} & L, C \cup \{p = q\} & \mapstochar\twoheadrightarrow & L, C \cup \{p' = q\} \\
 & & & \text{if } p > p' \ \&\ p \overset{+}{\longleftrightarrow}_{R \cup L} p' \\
\textbf{Compose} & L, C \cup \{p = q\} & \mapstochar\twoheadrightarrow & L, C \cup \{p' = q\} \\
 & & & \text{if } p \longleftrightarrow_{C}^{g=d} p' \ \&\ p \sqsupset g > d
\end{array}$$

Figure 22.4: Proof by consistency rules

2. Induce: Then $C' = C$.
3. Delete: $C - C' = \{(p = q) | (p = q) \in ITh(R)\}$. Then for any ground substitution σ, $\sigma(p) \downarrow_R = \sigma(q) \downarrow_R$ by definition of $ITh(R)$ and because R is ground convergent.
4. Simplify: $C - C' = \{(p = q)\}$ and $C' - C = \{(p' = q)\}$. Since $p \overset{+}{\longleftrightarrow}_{R \cup L} p'$, then $\forall \sigma$ ground, $\sigma(p) \overset{+}{\longleftrightarrow}_{R \cup L} \sigma(p')$, so $\sigma(p) \downarrow_R = \sigma(p') \downarrow_R$.

   Now if C' is consistent with R, $p \overset{+}{\longleftrightarrow}_{R \cup L} p' \longleftrightarrow_{C'} q$ implies $\forall \sigma$ ground, $\sigma(p) \overset{*}{\longleftrightarrow}_R \sigma(p') \overset{*}{\longleftrightarrow}_R \sigma(q)$.

   If C is consistent with R, $p' \overset{+}{\longleftrightarrow}_{R \cup L} p \longleftrightarrow_C q$ implies $\forall \sigma$ ground, $\sigma(p') \overset{+}{\longleftrightarrow}_R \sigma(p) \overset{*}{\longleftrightarrow}_R \sigma(q)$.
5. Compose: $C - C' = \{(p = q)\}$ and $C' - C = \{(p' = q)\}$. Since $p \longleftrightarrow_C p'$, using an equality in $C \cap C'$, then $\forall \sigma$ ground, $\sigma(p) \downarrow_R = \sigma(p') \downarrow_R$, because $\sigma(p)_{|\omega} = \alpha(g), \sigma(p')_{|\omega} = \alpha(d)$, $(g = d) \in C$ so $\alpha(g) \overset{*}{\longleftrightarrow}_R \alpha(d)$.

   Now if C' is consistent with R, $p \longleftrightarrow_C p' \longleftrightarrow_{C'} q$ implies $\forall \sigma$ ground, $\sigma(p) \overset{+}{\longleftrightarrow}_R \sigma(p') \overset{*}{\longleftrightarrow}_R \sigma(q)$.

   If C is consistent with R, $p' \longleftrightarrow_C p \longleftrightarrow_C q$ implies $\forall \sigma$ ground, $\sigma(p') \overset{+}{\longleftrightarrow}_R \sigma(p) \overset{*}{\longleftrightarrow}_R \sigma(q)$.

□

The functionality of a proof by consistency procedure is as follows:

- If the starting set of conjectures C_0 is inconsistent, then the procedure will generate a provably inconsistent set C_i.
- If the procedure reaches some step i where all equalities in C_i have been marked (which means that all critical pairs of R on all equalities have been computed) and no provably inconsistent C_j with $j \leq i$ has been detected, then all equalities in $\bigcup_{j \leq i} C_j$ are inductive theorems of R.
- The procedure may not terminate. In this case, adding new lemmas, for instance obtained by generalization as in Boyer and Moore's theorem prover, can make the process terminate.

The desired property for this procedure is actually refutation completeness: for any unsatisfiable conjecture, there exist a derivation using the transition rules applied with an adequate strategy that will provide a provable inconsistency.

Definition 22.7 A proof by consistency procedure is *refutationally complete* if from an inconsistent set of conjectures C, it generates a derivation in which some C_i is provably inconsistent.

Provable inconsistencies may be reflected at the level of proofs by the notion of inconsistency witness due to Gramlich [Gra89]. This leads to look at a proof by consistency procedure as a proof transformation process, but restricted to a specific kind of proofs. Let us make this notion more precise.

Indeed if C is inconsistent, there exists an equality $(s = t) \in C$ and a R-normalized substitution σ such that

$$s' \overset{*}{\longleftarrow}_R \sigma(s) \longleftrightarrow_C \sigma(t) \overset{*}{\longrightarrow}_R t'$$

with $s' \neq t'$. Such a proof witnesses inconsistency.

Definition 22.8 Let C be a set of conjectures. An *inconsistency witness* for C is any proof on ground terms of the form:

$$s' \overset{*}{\longleftarrow}_R \sigma(s) \longleftrightarrow_C \sigma(t) \overset{*}{\longrightarrow}_R t'$$

with $s' \neq t'$ and σ R-normalized.

More precisely, such proofs can be of two forms:

Definition 22.9 Let C be a set of conjectures.

An *indirect inconsistency witness* for C is any proof on ground terms of the form:

$$s' \overset{+}{\longleftarrow}_R \sigma(s) \longleftrightarrow_C \sigma(t) \overset{*}{\longrightarrow}_R t'$$

with $s' \neq t'$ and σ R-normalized.

A *direct inconsistency witness* for C is any proof on ground terms of the form:

- either $\sigma(s) \longleftrightarrow_C \sigma(t) \overset{*}{\longrightarrow}_R t'$ with $\sigma(s)$ irreducible and $\sigma(s) \neq t'$ since $\sigma(s) > \sigma(t) \geq t'$,
- or $\sigma(s) \longleftrightarrow_C \sigma(t)$ with $\sigma(s)$ and $\sigma(t)$ irreducible and $\sigma(s) \neq \sigma(t)$.

An ordering on proofs is now defined and tailored to the inconsistency witnesses transformation process.

An elementary proof step is here a proof $\sigma(p) \longleftrightarrow_C \sigma(q)$. The complexity measure of this elementary proof step is defined by:

$$\begin{array}{lcll} c(\sigma(p),\sigma(q)) & = & (\{\sigma(p)\}, p, \sigma(q)) & \text{if } p > q \\ c(\sigma(p),\sigma(q)) & = & (\{\sigma(q)\}, q, \sigma(p)) & \text{if } q > p \\ c(\sigma(p),\sigma(q)) & = & (\{\sigma(p),\sigma(q)\}) & \text{otherwise} \end{array}$$

Let $>'$ be a simplification ordering that contains R and $>$ (for instance the transitive closure of $(> \cup \rhd_{sub})$). Complexities of elementary proof steps are compared using the lexicographic combination, denoted $>_{ec}$, of the multiset extension $>'^{mult}$ of the simplification ordering for the first component, the strict encompassement ordering $\sqsupset$ for the second component and the simplification ordering $>'$ for the third one. Since both $>$ and $\sqsupset$ are well-founded, so is $>_{ec}$. The complexity of a non-elementary proof of the form

$$s' \overset{*}{\longleftarrow}_R \sigma(s) \longleftrightarrow_C \sigma(t) \overset{*}{\longrightarrow}_R t'$$

is defined as $c(\sigma(s), \sigma(t))$.

For any other proof $\mathcal{P}$, $c(\mathcal{P}) = max$ where max is a symbol taken to be maximal in $>_{ec}$.

Let define the well-founded ordering on proofs by:

$$\mathcal{P} >_c \mathcal{P}' \text{ if } c(\mathcal{P}) >_{ec} c(\mathcal{P}').$$

The following lemma states that any application of the transition rules does not increase the complexity of proofs.

Lemma 22.4 Whenever $(L, C) \mapsto\!\!\!\rightarrow (L', C')$ and $\mathcal{P}$ is a proof using $R \cup C$, then there is a proof using $R \cup C'$ such that $\mathcal{P} \geq_c \mathcal{P}'$.

Proof: Let $\mathcal{P}$ be a proof using $R \cup C$. If $c(\mathcal{P}) = max$, the conclusion is clearly satisfied. Otherwise, $\mathcal{P}$ is an inconsistency witness

$$s' \overset{*}{\longleftarrow}_R \sigma(p) \longleftrightarrow_C \sigma(q) \overset{*}{\longrightarrow}_R t'$$

with $s' \neq t'$ and s', t' both irreducible.

The proof $\mathcal{P}'$ is found by looking at the transformation induced by each transition rule.

1. Deduce: If **Deduce** applies on C, $\mathcal{P}$ is again a proof in $C' \cup R$, so $\mathcal{P}' = \mathcal{P}$.
2. Induce: If **Induce** applies, C is not modified, again $\mathcal{P}' = \mathcal{P}$.
3. Delete: If **Delete** modifies C, this means that $(p = q) \in ITh(R)$, so $s' = t'$, which is impossible.
4. Simplify: If **Simplify** modifies $(p = q)$, since $\sigma(p) \overset{+}{\longrightarrow}_{R \cup L} \sigma(p')$, $\sigma(p)\downarrow_R = \sigma(p')\downarrow_R = s'$. Let choose $\mathcal{P}'$ as

$$s' \overset{*}{\longleftarrow}_R \sigma(p') \longleftrightarrow_C \sigma(q) \overset{*}{\longrightarrow}_R t'.$$

Now $c(\mathcal{P})$ is

- if $p > q$, $c(\sigma(p), \sigma(q)) = (\{\sigma(p)\}, p, \sigma(q))$, which is strictly greater w.r.t. $>_{ec}$ than $c(\mathcal{P}')$ whose first component is either $\{\sigma(p')\}$, or $\{\sigma(q)\}$, or $\{\sigma(p'), \sigma(q)\}$, since $p > p'$.
- if $q > p$, $c(\sigma(p), \sigma(q)) = (\{\sigma(q)\}, q, \sigma(p))$, which is strictly greater w.r.t. $>_{ec}$ than $c(\mathcal{P}') = (\{\sigma(q)\}, q, \sigma(p'))$, by looking at the third component, since $q > p > p'$.
- otherwise, $c(\sigma(p), \sigma(q)) = (\{\sigma(p), \sigma(q)\})$ which is strictly greater w.r.t. $>_{ec}$ than $c(\mathcal{P}')$ whose first component is either $\{\sigma(p')\}$, or $\{\sigma(q)\}$, or $\{\sigma(p'), \sigma(q)\}$, since $p > p'$.

5. Compose: If **Compose** modifies $(p = q)$, let assume that $\sigma(p) \longleftrightarrow_C^{g=d} \sigma(p')$ and $\sigma(p') \downarrow_R = s''$. If $s'' \neq t'$, let choose $\mathcal{P}'$ as
$$s'' \overset{*}{\longleftarrow}_R \sigma(p') \longleftrightarrow_C \sigma(q) \overset{*}{\longrightarrow}_R t'.$$
Now $c(\mathcal{P}) >_{ec} c(\mathcal{P}')$ with the same proof as for **Simplify**, since again $p > p'$.
If $s'' = t'$, the previous proof is no more an inconsistency witness and another $\mathcal{P}'$ must be chosen. Since $p \longleftrightarrow_C^{g=d} p'$, there exists a subterm of $\sigma(p)$ at some position ω which is $\tau(g)$ and $\sigma(p') = \sigma(p)[\omega \hookleftarrow \tau(d)]$. Let $g' = \tau(g) \downarrow_R$ and $d' = \tau(d) \downarrow_R$. Since $s' \neq t'$, $s'' \neq s'$ so $g' \neq d'$. So let choose $\mathcal{P}'$ as
$$g' \overset{*}{\longleftarrow}_R \tau(g) \longleftrightarrow_C \tau(d) \overset{*}{\longrightarrow}_R d'.$$
Now $c(\mathcal{P}') = (\{\tau(g)\}, g, \tau(d))$ since $g > d$. Then $c(\mathcal{P})$ is
   - if $p > q$, $c(\sigma(p), \sigma(q)) = (\{\sigma(p)\}, p, \sigma(q))$, which strictly greater w.r.t. $>_{ec}$ than $c(\mathcal{P}')$ because either $\tau(g)$ is a strict subterm of $\sigma(p)$ and then $\sigma(p) >' \tau(g)$, or $\sigma(p) = \tau(g)$ and $p \sqsupset g$.
   - if $q > p$, $c(\sigma(p), \sigma(q)) = (\{\sigma(q)\}, q, \sigma(p))$, which strictly greater w.r.t. $>_{ec}$ than $c(\mathcal{P}')$, since $\sigma(q) > \sigma(p) \geq' \tau(g)$.
   - otherwise, $c(\sigma(p), \sigma(q)) = (\{\sigma(p), \sigma(q)\})$ which strictly greater w.r.t. $>_{ec}$ than $c(\mathcal{P}')$ because either $\sigma(p) >' \tau(g)$, or $\sigma(p) = \tau(g)$ and $p \sqsupset g$.

□

To assure transformation of inconsistency witnesses of a set C which is inconsistent but not provably inconsistent, the notion of covering set is introduced.

Definition 22.10 C' is a *covering set* for C with respect to R and $>$ if

- $C \subseteq ITh(R)$ iff $C \cup C' \subseteq ITh(R)$.
- for every indirect inconsistency witness of C, there is a smaller (w.r.t. $>_c$) inconsistency witness in $C \cup C'$.

The first condition in Definition 22.10 is motivated by the fact that it is not suitable to introduce arbitrary new conjectures in C' that have nothing to do with C.

Definition 22.11 A derivation $(L_0, C_0) \mapsto\!\!\!\!\rightarrow (L_1, C_1) \mapsto\!\!\!\!\rightarrow ....$ is *fair* if C_* is a covering set for C_∞.

A sufficient condition to satisfy the fairness hypothesis can be given:

Proposition 22.5 A derivation $(C_0, L_0) \mapsto\!\!\!\!\rightarrow (C_1, L_1) \mapsto\!\!\!\!\rightarrow ....$ is fair if $CP(R, C_\infty) \subseteq C_*$.

Proof: Assume that $CP(R, C_\infty) \subseteq C_*$.

- $C_\infty \subseteq ITh(R)$ if $C_* \cup C_\infty = C_* \subseteq ITh(R)$ is true since $C_\infty \subseteq C_*$.
  Conversely, if $C_\infty \subseteq ITh(R)$, this means that C_∞ is consistent with R, which is equivalent, according to Proposition 22.4, to C_0 consistent with R, and for any $i \geq 0$, C_i consistent with R. So C_* is consistent with R.
- Let $\mathcal{P}$ be an indirect inconsistency witness of C_∞
$$s' \overset{+}{\longleftarrow}_R \sigma(s) \longleftrightarrow_{C_\infty} \sigma(t) \overset{*}{\longrightarrow}_R t'$$
with σ R-normalized. Then there exists a critical pair $p = q$ of R into C_∞ which by hypothesis belongs to C_*, and a substitution τ such that $\sigma(s) \rightarrow_R \tau(p)$ and $\sigma(t) = \tau(q)$. So there exists an inconsistency witness $\mathcal{P}'$ in C_*
$$u' \overset{+}{\longleftarrow}_R \tau(p) \longleftrightarrow_{C_*} \tau(q) \overset{*}{\longrightarrow}_R v'$$
such that $\mathcal{P} >_c \mathcal{P}'$.

□

Proving that the proof consistency procedure acts as described relies on the following theorem and proposition.

Theorem 22.4 *Let R be the initial ground convergent rewrite system presenting the theory E, C_0 be the set of conjectures to be proved, and L_0 a set of inductive lemmas of R. If C_0 is inconsistent and if $(L_0, C_0) \mapsto\!\!\!\rightarrow (L_1, C_1) \mapsto\!\!\!\rightarrow ....$ is a fair derivation, then some set C_i is provably inconsistent.*

Proof: If C_0 is provably inconsistent, then the result is proved. Else there exists a proof that some conjecture $s = t$ is inconsistent with R:

$$\mathcal{P}_0 : s' \overset{*}{\longleftarrow}_R \sigma(s) \longleftrightarrow_C \sigma(t) \overset{*}{\longrightarrow}_R t'$$

such that s', t' irreducible and distinct.

We first prove that for any proof $\mathcal{P}$ of the form

$$s' \overset{+}{\longleftarrow}_R \sigma(s) \longleftrightarrow_{C_i} \sigma(t) \overset{*}{\longrightarrow}_R t'$$

such that s', t' irreducible and distinct and $t \not> s$, for some $i \geq 0$, there exists a proof $\mathcal{P}'$ in $R \cup C_j$ for some $j \geq 0$, such that $c(\mathcal{P}) >_c c(\mathcal{P}')$.

By looking at the proof of Lemma 22.4, this is true if $s = t$ does not persist. If $s = t$ persists, fairness implies that C_* is a covering set for $s = t$. So there exists an inconsistent ground proof $\tau(p) = \tau(q)$ with $p = q \in C_j$ for some $j \geq 0$ such that $c(\sigma(s), \sigma(t)) >_c c(\tau(p), \tau(q))$. Let $\mathcal{P}'$ be the proof $u' \overset{*}{\longleftarrow}_R \tau(p) \longleftrightarrow_{C_j} \tau(q) \overset{*}{\longrightarrow}_R v'$ where u' and v' irreducible. Since $\tau(p) = \tau(q)$ is inconsistent, $u' \neq v'$. So $c(\mathcal{P}') = c(\tau(p), \tau(q))$ and $c(\mathcal{P}) >_c c(\mathcal{P}')$.

Now from $\mathcal{P}_0$, we can construct a sequence $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, ...$ such that $\mathcal{P}_i$ is a proof using $R \cup C_{j_i}$ and $c(\mathcal{P}_0) >_c c(\mathcal{P}_1) >_c c(\mathcal{P}_2) >_c ....$ Since $>_c$ is well-founded, this sequence is finite. So for some $k > 0$, $\mathcal{P}_k$ is a proof of the form

- either $\sigma(s) \longleftrightarrow_{C_{j_k}} \sigma(t) \overset{*}{\longrightarrow}_R t'$ with $\sigma(s)$ irreducible and $\sigma(s) \neq t'$ and $s > t$,
- or $\sigma(s) \longleftrightarrow_{C_{j_k}} \sigma(t)$ with $\sigma(s)$ and $\sigma(t)$ irreducible and $\sigma(s) \neq \sigma(t)$.

This implies that C_{j_k} is provably inconsistent and yields the result. □

Lemma 22.5 A set C of equalities is consistent with R iff C is a covering set for itself and is not provably inconsistent.

Proof: If C is consistent with R, it is easy to check that it is a covering set for itself and it is not provably inconsistent.

Conversely, assume that C is a covering set for itself and is not provably inconsistent. If there exists an inconsistency in C, it is possible to find an indirect inconsistency witness and to construct an infinite sequence of indirect inconsistency witnesses which is strictly decreasing w.r.t. $>_c$. This yields a contradiction to the well-foundedness of $>_c$. □

This lemma directly implies the following result:

Proposition 22.6 Let R be the initial ground convergent rewrite system presenting the theory E, C_0 be the set of conjectures to be proved, L_0 a set of inductive lemmas of R, and $(L_0, C_0) \mapsto\!\!\!\rightarrow (L_1, C_1) \mapsto\!\!\!\rightarrow ....$ a fair derivation. If there exists a step i such that C_i is a covering set for itself and is not provably inconsistent, then $\bigcup_{j \leq i} C_j$ is consistent with R.

Proof: According to Lemma 22.5, C_i is consistent with R and according to Proposition 22.4, each C_j for $j \leq i$ is also consistent with R. □

Note that, according to Proposition 22.5, C_i will be a covering set for itself if $CP(R, C_i) \subseteq \bigcup_{j \leq i} C_j$.

Computation of covering sets is usually based on critical pairs, but it is not always necessary to compute all critical pairs of R into C. In order to improve the efficiency of the proof by consistency procedure, several results are useful [Bac87]:

- If an equality in C is orientable, then it is enough to superpose rules of R in the greater term of the equality only. This case is actually the same as for inductive completion where such an equality is always oriented. If both members of the equality are incomparable (neither $g > d$ nor $d > g$), it is also enough to superpose rules of R in any of the two sides g or d, provided both are ground reducible.

- If an equality $(c(t_1, ..., t_n) = c(t'_1, ..., t'_n))$ with c a constructor is generated, and provided that no rule in R applies on ground terms built only with constructors, the equality can be replaced by the system of equalities $\{(t_i = t'_i) | i = 1, ..., n\}$. It is straightforward to adapt the transition rules given in Figure 22.3 to incorporate them with $\mathcal{CP}$.

- If an equality is generated which is a variant of an already existing equality in C, it is useless to consider it.

- If an equality $t[u] = t[v]$ is generated and if $u = v$ is already in C, then it is useless to consider the first one, since **Compose** and **Delete** would apply and make it disappear.

- A last optimization is to restrict superpositions to some subterms that correspond intuitively to induction subterms. Occurrences of such subterms are called *comprehensive sets of positions* in [Bac87] and *sets of complete superpositions* in [Fri86].

Definition 22.12 A position ω in a term t is said to be *comprehensive* with respect to R if $t_{|\omega}$ is not a variable and each instance by an irreducible ground substitution $\sigma(t_{|\omega})$ is an instance of a left-hand side in R.

Example 22.5 Consider again the following signature:

$$\begin{array}{llll} sort & Int & & \\ 0: & & \mapsto & Int \\ succ: & Int & \mapsto & Int \\ +: & Int\ \ Int & \mapsto & Int \end{array}$$

and the set R of rules:

$$\begin{array}{rcl} \forall x : Int,\ x + 0 & \rightarrow & x \\ \forall x, y : Int,\ x + succ(y) & \rightarrow & succ(x + y). \end{array}$$

In the term $t = x + (y + z)$ the position 2 referring to the subterm $y + z$ is comprehensive.

If ω is a comprehensive position in the term p, the set of all critical pairs obtained by superposing R on the equality $(p = q)$ at position ω in p is a covering set for $(p = q)$.

22.6.2 A proof by consistency procedure

A *proof by consistency procedure* is any program which takes as inputs a ground convergent rewrite system R, a reduction ordering that contains R, a set of inductive lemmas L and a set of conjectures C, and uses the transition rules of $\mathcal{IC}$ to generate a derivation from (L, C).

The procedure IND-PROVE presented in Figure 22.5 implements the proof by consistency method where R is implicit. It has three possible issues: success when the conjectures are valid, disproof when there exists an inconsistency and it may not terminate and generate an infinite number of covering sets of conjectures.

The `SIMPLIFICATION` procedure computes simplified forms of l and r, using equalities in C, rules in R and lemmas in L, according to transition rules **Simplify** and **Compose**.

The `COVERING-SET` procedure computes the necessary superpositions of R on the equality $(l' = r')$. The equality is marked whenever the necessary critical pairs with rules in R have been computed and added to C.

Example 22.6 Let consider the specification

$$\begin{array}{rll} sort & Int & \\ 0: & \mapsto & Int \\ succ: Int & \mapsto & Int \end{array}$$

$$\begin{array}{rcl} \forall x : Int,\ x + 0 & \rightarrow & x \\ \forall x, y : Int,\ x + succ(y) & \rightarrow & succ(x + y). \end{array}$$

```
PROCEDURE IND-PROVE (C, L, >)
IF all equalities in C are marked
THEN STOP with SUCCESS
ELSE Choose an unmarked equality (l = r) fairly;
     C := C-{(l=r)};
     (l'=r') := SIMPLIFICATION (l = r, C, L);
     CASE l'=r' in ITh(R)  THEN IND-PROVE(C, L U {(l'=r')}, >)
          (l'= r') is provably inconsistent THEN STOP with DISPROOF;
          ELSE C := C U {(l'=r')} U COVERING-SET (R,l'=r');
               Mark the equality (l'=r') in C;
               IND-PROVE(C, L, >)
     END CASE
END IF
END IND-PROVE
```

Figure 22.5: A proof by consistency procedure

In order to prove associativity of $+$, let $C = \{x + (y + z) = (x + y) + z\}$ and chose the ordering such that $x + (y + z) > (x + y) + z$. Then it is enough to superpose R on $x + (y + z)$. This gives, by unifying $(y + z)$ and $x + 0$, the critical pair $x + y = (x + y) + 0$, that is simplified into $x + y = x + y$ and eliminated.

By unifying $(y + z)$ and $x + succ(y)$, the critical pair $x + (y + succ(z)) = (x + y) + succ(z)$ is found. It is reduced to $succ(x + (y + z)) = succ((x + y) + z)$. Both terms are C-equivalent and this equality disappears.

Assume now that the ordering is chosen such that $x + (y + z) < (x + y) + z$. Then more critical pairs are obtained. By superposition of R on $(x + y)$, we get

$$x + (0 + z) = x + z \tag{22.1}$$
$$x + (succ(y) + z) = succ(x + y) + z \tag{22.2}$$

By superposition of R on $(x + y) + z$, we get two new critical pairs that are convergent.

Then R is superposed on the subterm $(0 + z)$ of Equation 22.1 and two convergent critical pairs are obtained.

R is superposed on $(succ(x + y) + z)$ of Equation 22.2 and gives four critical pairs that are convergent.

Eventually R is superposed on the subterm $(succ(y) + z)$ of Equation 22.2 and we get

$$x + succ(y) = succ(x + y) + 0 \tag{22.3}$$
$$x + succ((succ(y) + z)) = succ(x + y) + succ(z) \tag{22.4}$$

Both sides of the critical pair 22.3 reduce to $succ(x + y)$. The critical pair 22.4 reduces to

$$succ(x + (succ(y) + y)) = succ(succ(x + y) + z).$$

Both sides are C-equal using Equation 22.2. All the critical pairs have been computed and the procedure stops.

The proof of commutativity of $+$ is similar. After adding $C = \{x + y = y + x\}$, the procedure finds two critical pairs

$$0 + x = x \tag{22.5}$$
$$succ(y) + x = succ(x + y). \tag{22.6}$$

Again critical pairs of R on Equation 22.5 are now convergent. Considering now superposition of R on Equation 22.6, one gets

$$succ(0) + x = succ(x) \tag{22.7}$$
$$succ(succ(y + x)) = succ(succ(x + y)). \tag{22.8}$$

The equality 22.7 satisfies, using 22.6,

$$succ(0) + x \longleftrightarrow_C succ(x + 0) \to_R succ(x)$$

thus it can disappear. Both terms of equality 22.8 are C-equivalent, so the procedure terminates.

Example 22.7 Consider an operation *alter* on lists built with construtors *nil* (empty list) and *push*, that shuffle two lists and produces a third one. The list structure and the *alter* operation are described by the following presentation giving ranks of operations

$$\begin{array}{rcl} sorts: & & Elt, List \\ nil: & \mapsto & List \\ push: Elt\ List & \mapsto & List \\ alter: List\ List & \mapsto & List \end{array}$$

and rewrite rules to define *alter*:

$$\begin{array}{rcl} alter(nil, z) & \rightarrow & z \\ alter(push(x, y), z) & \rightarrow & push(x, alter(z, y)) \end{array}$$

It is easy to check that this set of rewrite rules R is terminating and confluent. Consider now the conjecture $alter(y, nil) = y$ which is not an equational consequence of the previous rules. The proof by consistency procedure initialized with $C_0 = \{alter(y, nil) = y\}$ works as follows:

- first ground reducibility of $alter(y, nil)$ is checked. A test set for y is $\{nil, push(x, y)\}$ and obviously $alter(nil, nil)$ and $alter(push(x, y), nil)$ are R-reducible.

- second critical pairs are computed from R into $alter(y, nil)$. They are two:

with $y = nil$, we get $(nil = nil)$,

with $y = push(x', y')$, we get $(push(x', alter(nil, y')) = push(x', y'))$ which is obviously reducible to $(push(x', y') = push(x', y'))$. So the process terminates without detecting inconsistency, which proves the conjecture.

Exercice 63 — Run the inductive completion procedure of Section 22.5 on Example 22.7. Compare.
Answer: Just do it.

Comparisons between inductive completion and proof by consistency procedures lead to interesting remarks. On one hand, it can be argued that inductive completion attempts to solve all possible induction schemes and often fails to terminate, while the use of inductive positions in the proof by consistency procedure can select a specific induction schema. On the other hand, it is possible that inductive completion deduces equalities that are useful for simplification but cannot be deduced with the more restrictive deduction rule of $\mathcal{CP}$. This is illustrated by the following example [BD89b, Küc89].

Example 22.8 Consider the operations *append* and *rev* on lists built with construtors *nil* (empty list) and *cons*.

$$\begin{array}{rcl} sorts: & Elt, List & \\ nil: & \mapsto & List \\ cons: Elt\ List & \mapsto & List \\ append: List\ List & \mapsto & List \\ rev: List & \mapsto & List \end{array}$$

Let R be the ground convergent rewrite system

$$\begin{array}{rcl} append(nil, x) & \rightarrow & x \\ append(cons(x, y), z) & \rightarrow & cons(x, append(y, z)) \\ rev(nil) & \rightarrow & nil \\ rev(cons(x, y)) & \rightarrow & append(rev(y), cons(x, nil)) \end{array}$$

and $rev(rev(x)) = x$ the conjecture to be proved.

The last rule in R can be superposed on the conjecture to produce

$$rev(append(rev(y), cons(x, nil))) = cons(x, y)$$

The initial conjecture can be superposed on this new one to give

$$rev(append(y), cons(x, nil)) = cons(x, rev(y))$$

which can be oriented into a rewrite rule

$$rev(append(y), cons(x, nil)) \rightarrow cons(x, rev(y))$$

The first critical pair can now be simplified and deleted. The remaining set of rules is convergent, which implies that the conjecture holds.

On the other hand, with a linear deduction strategy that only allows for superpositions of R into conjectures, the above rewrite rule cannot be deduced and an infinite derivation may be produced:

$$\begin{array}{rcl} rev(append(rev(y), cons(x, nil))) & = & cons(x, y) \\ rev(append(append(rev(z), cons(y, nil)), cons(x, nil))) & = & cons(x, cons(y, z)) \\ & \dots & \end{array}$$

Exercice 64 — Consider the specification

$$\begin{array}{rcl} sort & Int & \\ 0: & \mapsto & Int \\ succ: Int & \mapsto & Int \end{array}$$

$$\begin{array}{rcl} \forall x: Int,\ x + 0 & \rightarrow & x \\ \forall x, y: Int,\ x + succ(y) & \rightarrow & succ(x + y) \\ \forall x: Int,\ x * 0 & \rightarrow & x \\ \forall x, y: Int,\ x * succ(y) & \rightarrow & (x * y) + x. \end{array}$$

Assume given the following lemmas:

$$\begin{array}{rcl} \forall x, y: Int,\ x + y & = & y + x \\ \forall x, y, z: Int,\ x + (y + z) & = & (x + y) + z \end{array}$$

and prove the following conjectures:

$$\begin{array}{rcl} \forall x, y, z: Int,\ x * (y + z) & = & (x * y) + (x * z) \\ \forall x, y: Int,\ x * y & = & y * x \\ \forall x, y, z: Int,\ x * (y * z) & = & (x * y) * z \end{array}$$

Answer: Just do it.

22.7 Inductive proofs by rewriting and implicit induction

One major drawback of the previous methods is that the rewrite systems used to describe the underlying theory have to be confluent or at least ground confluent. In this section, we drop this strong hypothesis and describe a method initiated in [KR90, Red90], called *rewriting induction* and developed in [BKR92, Bou94]. The essential idea underlying rewriting induction is that, when a (conditional or unconditional) rewrite system R is terminating, then the corresponding rewrite relation is a well-founded ordering on terms and can be used as support for inductive reasoning. The method combines simplification by rewriting with selection of significant terms selected in test sets. The key idea of the simplification strategy is to use axioms, previously proved conjectures and instances of the conjecture itself as soon as they are smaller than the current conjecture with respect to some well-founded relation. Using these ideas, a general inference system to perform induction in conditional and equational theories is proposed in [BKR92, Bou94]. It is proved correct in general and refutationally complete for convergent conditional (or equational) theories.

Formulas to be proved are clauses and $>$ is a well-founded ordering on terms that extends to clauses. R is a set of conditional equational clauses $l \rightarrow r$ if $(s_1 = t_1 \wedge \dots \wedge s_n = t_n)$ such that $\mathcal{V}ar(r) \bigcup_{i=1,\dots,n} \mathcal{V}ar(s_i) \cup \mathcal{V}ar(t_i)$ is a subset of $\mathcal{V}ar(l)$ and for any ground substitution σ, $\{\sigma(l)\} >^{mult} \{\sigma(r), \sigma(s_1), \sigma(t_1) \dots, \sigma(s_n), \sigma(t_n)\}$.

Inductively valid clauses are implications that hold in the initial algebra defined by R.

Definition 22.13 A clause $(\bigwedge_{i=1,\dots,n} s_i = t_i \text{if} \bigvee_{j=1,\dots,m} u_j = v_j)$ is an *inductive theorem of* R if for any ground substitution σ, whenever $\forall i = 1, \dots, n, \sigma(s_i) \overset{*}{\longleftrightarrow}_R \sigma(t_i)$, then there exists $j \in [1, \dots, m]$ such that $\sigma(u_j) \overset{*}{\longleftrightarrow}_R \sigma(v_j)$.

The clause c is said inductively valid in R, which is denoted by $R \models_{ind} c$.

We describe here a general method that may be specialized with stronger hypotheses on the specifications. For instance, for boolean specifications where conditions are boolean expressions, some of the concepts defined below may be efficiently implemented. The interested reader can refer to [Bou94].

22.7.1 Selection of induction schemes

To perform a proof by induction, it is necessary to provide some induction schemes. In the approach described below, these schemes are provided by first selecting *inductive variables* on which induction has to be applied, and second by a special set of terms called *test-set*.

A sort s is said *infinitary* if there exists an infinite set of ground terms of sort s irreducible by R. Otherwise it is *finitary*.

Definition 22.14 Let R be a conditional rewrite system and c be a clause or a term. x of sort s is an *inductive variable* of c if either s is finitary, or x occurs in a non-variable subterm u of c and there exists a conditional rewrite rule in R $l \rightarrow r$ if $(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ such that

- u and l are unifiable
- the position ω of x in u satisfies:

either $l_{|\omega}$ is not a variable, or $l_{|\omega}$ is a non-linear variable of l, or $l_{|\omega}$ is an inductive variable of s_i or t_i for some $i = 1, \ldots, n$.

The set of inductive variables of c is denoted by $Var - ind(c)$.

Example 22.9 Consider the following system that defines the two predicates *odd* and *even* on natural numbers.

$$\begin{array}{rrlrcl}
 & & & true & \neq & false \\
 & & & even(0) & \rightarrow & true \\
 & & & even(s(0)) & \rightarrow & false \\
 & & & even(s(s(x))) & \rightarrow & even(x) \\
even(x) & = & true \quad \text{if} & odd(x) & \rightarrow & false \\
even(x) & = & false \quad \text{if} & odd(x) & \rightarrow & true
\end{array}$$

The variable x is an inductive variable both of the term $even(x)$ and of the term $odd(x)$.

In order to illustrate the computation of inductive variables, let us consider the pure equational case. Induction variables are determined thanks to the preliminary computation of inductive positions of function symbols.

Definition 22.15 Given a set of rewrite rule R, the set of *inductive positions* for a function symbol f is

$$Occ - ind(R, f) = \{\omega \mid \exists(l \rightarrow r) \in R, l(\Lambda) = f, \omega \in \mathcal{D}om(l) - \{\Lambda\},\ \textit{and } l_{|\omega} \textit{ is not a linear variable of } l\ \}$$

An *induction variable* of a term t is either t itself if t is a variable, or a finitary variable, or a variable that occurs at a position $\upsilon\omega$ where ω is an inductive position of $t_{|\upsilon}$.

For a set of conditional rewrite rules R, a term t is weakly R-irreducible if for any subterm u such that there exists a rule *condplr* and a substitution σ with $\sigma(l) = u$, $\sigma(p)$ is unsatisfiable in the theory defined by R.

For a term t and a set of terms T, a T-substitution is a substitution that instantiates each inductive variable of t by a term of T whose variables have been renamed.

The notion of cover sets needed here is in essence a finite description of the initial model for a set of conditional rewrite rules R. So any ground R-irreducible term is an instance of some element in the cover set. The notion was introduced in [Red90, ZKK88].

Definition 22.16 A *cover set* for a set of conditional rewrite rules R is a finite set $CS(R)$ of R-irreducible terms that satisfies: for any R-irreducible ground term u, there exists a term t in $CS(R)$ and a ground substitution σ such that $\sigma(t) = u$.

Test-sets have an additional property:

Definition 22.17 A *test set* for a set of conditional rewrite rules R is a finite set $S(R)$ of R-irreducible terms that satisfies:

a. $S(R)$ is a cover set for R.

b. for any term t in $S(R)$ and any $S(R)$-substitution σ of t, if $\sigma(t)$ is weakly R-reducible, then there exists a ground substitution τ such that $\tau(\sigma(t))$ is ground and R-irreducible.

The first condition in the definition of test sets allows restricting attention to the set of terms in R-normal forms. The second condition **b.** is fundamental to refute theorems. It ensures that when an instance of an equality by a $S(R)$ substitution does not match any left-hand side of rules, an inconsistency is revealed.

Example 22.10 Consider the specification (Σ, R):

$$\begin{array}{rcl} sort & Int & \\ 0: & \mapsto & Int \\ succ: Int & \mapsto & Int \\ pred: Int & \mapsto & Int \end{array}$$

$$\begin{array}{rcl} \forall x: Int,\ x+0 & \rightarrow & x \\ \forall x: Int,\ 0+x & \rightarrow & x \\ \forall x,y: Int,\ succ(x)+y & \rightarrow & x+succ(y) \\ \forall x,y: Int,\ x+succ(y) & \rightarrow & succ(x+y) \\ \forall x,y: Int,\ pred(x)+y & \rightarrow & x+pred(y) \\ \forall x,y: Int,\ x+pred(y) & \rightarrow & pred(x+y) \\ \forall x: Int,\ succ(pred(x)) & \rightarrow & x \\ \forall x: Int,\ pred(succ(x)) & \rightarrow & x. \end{array}$$

Then $S(R) = \{0, succ(x), pred(x)\}$.

The construction of a test set for a rewrite system R is decidable and algorithms are given for the equational case in [Kou90] and for the conditional case with free constructors in [KR90, BR93]. The set of function symbols is divided into a set of completely defined function symbols and others called constructors. As before, the depth of a term t, denoted $depth(t)$, is the maximal size of positions ω in $\mathcal{D}om(t)$. The strict depth of a term t, denoted $sdepth(t)$, is the maximal size of non-variable positions ω in $\mathcal{D}om(t)$. The (strict) depth of a rewrite rule set R, denoted by $depth(R)$ (resp. $sdepth(R)$), is the maximum of the (strict) depths of rules left-hand sides. Let $|R|$ be defined as
$depth(R) - 1$ if $sdepth(R) < depth(R)$ and R is left-linear
$depth(R)$ otherwise. When all function symbols are completely defined with respect to a set of free constructors, then the set of constructor terms of depth $\leq |R|$ where variables may occur only at depth $|R|$ is a test-set for R.

The role of test-set for refuting conjecture is enlightened by the following criterion of inconsistency witness and its relation to non-inductive consequence.

Definition 22.18 Let R be a conditional rewrite system and $>$ a well-founded ordering on clauses that contains R. A clause $c = \neg(t_1 = u_1) \vee \ldots \vee \neg(t_n = u_n) \vee (v_1 = w_1) \vee \ldots \vee (v_m = w_m)$ is an *inconsistency witness* if there exists a $S(R)$-substitution σ such that for any $i = 1, \ldots, n$, $R \models_{ind} \sigma(t_i) = \sigma(u_i)$ for any $j = 1, \ldots, m$, $\sigma(v_j)$ and $\sigma(w_j)$ are distinct and the maximal elements of $\{\sigma(v_j), \sigma(w_j)\}$ w.r.t. $>$ are weakly R-irreducible.

This definition is simpler when R is non-conditional:

Definition 22.19 Let R be a rewrite system and $>$ a well-founded ordering on terms that contains R. An equality $(g = d)$ is an inconsistency witness w.r.t. R if there exists a $S(R)$-substitution σ such that $\sigma(g)$ and $\sigma(d)$ are distinct and the maximal elements of $\{\sigma(g), \sigma(d)\}$ w.r.t. $>$ are R-irreducible.

This notion of inconsistency witness provides the basis for a proof by refutation.

Theorem 22.5 *[Bou94] Let R be a ground confluent conditional rewrite system, $>$ a well-founded ordering on terms that contains R, and c a clause. If c is an inconsistency witness, then it is not an inductive consequence of R.*

Example 22.11 In example 22.9, the system R is confluent on ground terms. The conjecture $even(x) = true \vee odd(x) = false$ is an inconsistency witness as shown by the instance $even(s(0)) = true \vee odd(s(0)) = false$.

Example 22.12 Consider the "union" operator defined on lists built on constructors $\{cons, nil\}$:

$$\begin{array}{rcl} union(nil, nil) & \rightarrow & nil \\ union(cons(x, nil), nil) & \rightarrow & cons(x, nil) \\ union(nil, cons(x, l)) & \rightarrow & cons(x, l) \\ union(cons(x, l), cons(y, l')) & \rightarrow & cons(x, cons(y, union(l, l'))) \end{array}$$

R is ground confluent and terminating. $S(R) = \{nil, cons(x.l)\}$. Consider the conjecture

$$union(l, l') = union(l', l)$$

Among the test-set instances, we have:
$union(cons(x, l), cons(y, l')) = union(cons(y, l'), cons(x, l))$
which simplifies to
$cons(x, cons(y, union(l, l'))) = cons(y, cons(x, union(l', l)))$.
Among the test-set instances of this last conjecture, we have:
$cons(x, cons(y, union(nil, nil))) = cons(y, cons(x, union(nil, nil)))$
which simplifies to
$cons(x, cons(y, nil)) = cons(y, cons(x, nil))$
which is an inconsistency witness. So $union(l, l') = union(l', l)$ is not an inductive consequence of R.

22.7.2 Transition rules for rewrite induction

The rewriting induction process described below is based on the expansion of a clause (or an equality) to be proved, using test sets and rewriting. Intuitively, an expansion is a reduced instance of an equality. Rewriting is performed on the greatest term of the instantiated equality or on both terms if they are uncomparable in the ordering.

Definition 22.20 Given R, a set of conjectures C and a set of inductive hypotheses H, the *expansion set* of a clause c in C, denoted $Exp(c)$, is the set of clauses $\{c_1, \ldots, c_n\}$ such that for any $CS(R)$-substitution σ and any ground substitution τ, there exists a k such that $\tau(\sigma(c)) > \tau(c_k)$ and $\tau(c_k) \Leftrightarrow \tau(\sigma(c))$ is an inductive consequence of $R \cup \{\theta(S) \mid S \in E \cup H \cup \{c\} and \tau(\sigma(c)) > \theta(S)\}$.

This definition is quite general and a particular instance may be given for equalities.

Definition 22.21 The *expansion set* of an equality $(p = q)$ such that $p \not< q$, w.r.t. a test set $S(R)$ is the set of equalities $Exp(p = q)$ defined as

- if $p > q$,
$$Exp(p = q) = \{p' = \sigma(q) \mid \sigma \in S(R) \text{ and } \sigma(p) \rightarrow_R p'\}$$
- if p and q are uncomparable,
$$Exp(p = q) = \{p' = q' \mid \sigma \in S(R) \text{ and } \sigma(p) \rightarrow_R p',\ \sigma(q) \rightarrow_R q'\}$$

A given clause may also be replaced by a set of smaller clauses.

Definition 22.22 Given R, a set of conjectures C and a set of inductive hypotheses H, the *simplification set* of a clause c in E, denoted $Simp(c)$ is the set of clauses $\{c_1, \ldots, c_n\}$ such that for any $CS(R)$-substitution σ, there exists a k such that $\sigma(c) > \sigma(c_k)$ and $\sigma(c_k) \Leftrightarrow \sigma(c)$ is an inductive consequence of $R \cup \{\theta(S) \mid S \in E \textit{ and } \sigma(c) > \theta(S)\} \cup \{\theta(S) \mid S \in H \textit{ and } \sigma(c) \geq_C \theta(S)\}$.

Definition 22.23 Given R, a set of conjectures C and a set of inductive hypotheses H, a clause c in E is redundant if for any ground substitution σ, $\sigma(c)$ is an inductive consequence of $R \cup \{\theta(S) \mid S \in E \textit{ and } \sigma(c) > \theta(S)\} \cup \{\theta(S) \mid S \in H \textit{ and } \sigma(c) \geq \theta(S)\}$.

For instance in the equational case, an equality $(p = p)$ trivially holds and thus may be deleted. Such simple forms can be obtained from the original equalities to prove and their expansions by a process of simplification. Let us first describe the rewriting induction process for the equational case. Given a set R of rewrite rules, the rewriting induction process is parameterized by a reduction ordering $>$ and transforms two sets of equalities: C contains the set of equalities to be proved and H registers equalities from C, that can be used as inductive hypotheses, after being expanded, The process is described by the following set of transition rules $\mathcal{RI}$ in Figure 22.6.

These rules are instances of more complex rules for conditional theories given in [Bou94]. The process is described by the following set of rules $\mathcal{CRI}$ in Figure 22.7.

Proposition 22.7 (Correctness of rewriting induction)
If there exists a derivation $(C, \emptyset) \mapsto\!\!\!\rightarrow (C_1, H_1) \mapsto\!\!\!\rightarrow \ldots . \mapsto\!\!\!\rightarrow (\emptyset, H)$, for some set H, then all equalities in C are inductive consequences of R.

$$\begin{array}{llll}
\textbf{Expand} & C \cup \{p = q\}, H & \mapsto\!\!\!\rightarrow & C \cup C', H \cup \{p = q\} \\
 & & & \text{if } p \not< q, C' = Exp(p = q) \\
\textbf{Delete} & C \cup \{p = p\}, H & \mapsto\!\!\!\rightarrow & C, H \\
\textbf{Simplify} & C \cup \{p = q\}, H & \mapsto\!\!\!\rightarrow & C \cup \{p' = q\}, H \\
 & & & \text{if } p \rightarrow_{R \cup H \cup C - \{p=q\}} p'
\end{array}$$

Figure 22.6: Induction by rewriting

$$\begin{array}{llll}
\textbf{Expand} & C \cup \{c\}, H & \mapsto\!\!\!\rightarrow & C \cup Exp(c), H \cup \{c\} \\
\textbf{Delete} & C \cup \{c\}, H & \mapsto\!\!\!\rightarrow & C, H \\
 & & & \text{if } c \text{ redundant} \\
\textbf{Simplify} & C \cup \{c\}, H & \mapsto\!\!\!\rightarrow & C \cup Simp(c), H
\end{array}$$

Figure 22.7: Induction by conditional rewriting

Proof: See [Red90] for the equational case, and [Bou94] for the conditional case. □

Example 22.13 An easy illustration of the method is provided by the proof of the commutativity of + on natural numbers. Consider the specification (Σ, R)

$$\begin{array}{lll}
sort & Int & \\
0: & \mapsto & Int \\
succ: & Int \mapsto & Int
\end{array}$$

$$\begin{array}{rcl}
\forall x : Int,\ x + 0 & \rightarrow & x \\
\forall x, y : Int,\ x + succ(y) & \rightarrow & succ(x + y)
\end{array}$$

The chosen ordering is the recursive path ordering induced by $+ > succ > 0$. Note that $S(R) = \{0, succ(z)\}$. Let $C_0 = \{u + v = v + u\}$. The equality is first expanded into $C_1 = \{u = 0 + u, succ(u + z) = succ(z) + u\}$. Then $H_1 = \{u + v = v + u\}$.

Again the first equality in C_1 is expanded.

$C_2 = \{0 = 0, succ(y) = 0 + succ(y), succ(u + z) = succ(z) + u\}$ and

$H_2 = \{u + v = v + u, u = 0 + u\}$.

The first equality in C_2 is deleted and the second is simplified using the orientable equality $0 + u \rightarrow 0$ in H_2 to $succ(y) = succ(y)$ that can be deleted too. We are left with

$C_3 = \{succ(u + z) = succ(z) + u\}$ and $H_3 = \{u + v = v + u, u = 0 + u\}$.

Again the first equality in C_3 is expanded.

$C_4 = \{succ(0 + z) = succ(z), succ(succ(y) + z) = succ(succ(z) + y)\}$ and

$H_4 = \{u + v = v + u, u = 0 + u, succ(u + z) = succ(z) + u\}$.

$succ(0 + z) = succ(z)$ is simplified (by $0 + u \rightarrow u$) and deleted. $succ(succ(y) + z) = succ(succ(z) + y)$ is simplified twice by $succ(z) + u \rightarrow succ(z + u)$ and yields $succ(succ(y + z)) = succ(succ(z + y))$. Using then $u + v = v + u$ at position 22 to simplify again, we eventually get a trivial equality which is deleted. We end with

$C_4 = \emptyset$ and $H_4 = \{u + v = v + u, u = 0 + u, succ(u + z) = succ(z) + u\}$.

More experiments with this technique, including with conditional rewrite systems, can be found in [Bou91].

Coming back to the assumption of confluence for the rewrite system R, the previous transition rules in $\mathcal{RI}$ may be used in a refutationally complete method for inductive proofs.

In order to get this refutation completeness result, a new transition rule, the **Refute** rule given in Figure 22.8, must be added to the set $\mathcal{RI}$ of Figure 22.6.

Theorem 22.6 *Let R be a ground convergent conditionnal rewrite system. Let $(C_0, \emptyset) \mapsto\!\!\!\rightarrow (C_1, H_1) \mapsto\!\!\!\rightarrow \ldots.$ be a derivation using $\mathcal{CRI}$ and* **Refute**. *If there exists j such that* **Refute** *applies to (C_j, H_j), then C_0 is not an inductive consequence of R.*

$$\textbf{Refute} \quad C \cup \{c\}, H \quad \mapsto\!\!\!\rightarrow \quad \begin{array}{l} refute \\ \text{if } c \text{ is an inconsistency witness} \end{array}$$

Figure 22.8: Refutation rule

Proof: See [Bou94]. □

Exercice 65 — Consider the specification

$$\begin{array}{llll} sort & Int & & \\ 0 & : & \mapsto & Int \\ succ & : Int & \mapsto & Int \\ + & : Int, Int & \mapsto & Int \\ - & : Int, Int & \mapsto & Int \end{array}$$

$$\begin{array}{rcl} \forall x : Int,\ x + 0 & \rightarrow & x \\ \forall x, y : Int,\ x + succ(y) & \rightarrow & succ(x + y) \\ \forall x : Int,\ x - 0 & \rightarrow & x \\ \forall x : Int,\ 0 - x & \rightarrow & 0 \\ \forall x, y : Int,\ succ(x) - succ(y) & \rightarrow & x - y. \end{array}$$

Prove the following conjecture:

$$\forall x, y, z : Int,\ (x + y) - y \quad = \quad x$$

Prove that the following conjecture

$$\forall x, y, z : Int,\ (x + y) - y \quad = \quad x$$

is not valid in the initial algebra of the specification.
Answer: Derive the following equalities:

$$\begin{array}{rcl} (x + y) - y & = & x \\ x - 0 & = & x \\ succ(x + y) - succ(y) & = & x \\ x + 0 & = & x \\ x & = & x \\ (x + y) - y & = & x \end{array}$$

and conclude validity.
Derive the following equalities:

$$\begin{array}{rcl} (x - y) + y & = & x \\ x + 0 & = & x \\ 0 + y & = & y \\ (x - y) + succ(y) & = & succ(x) \\ x & = & x \\ 0 + 0 & = & 0 \\ succ(0 + y) & = & 0 \\ succ(0) & = & 0 \end{array}$$

and conclude non validity.

22.8 Conclusion

Inductive reasoning is basic to computer science, proof theory and number theory. Automation of inductive proofs is becoming a hot topic in automated deduction community. Due to the undecidable nature of the domain of application, it is not likely that inductive theorem proving can be mechanized to as great a degree as first-order theorem proving. Thus, for an inductive theorem prover, issues such as user-interface, strategies, heuristics and tactics provided to the user, easy analysis of the proof steps and introduction of lemmas may become extremely crucial to the success of the prover. Current research problems are combining inductive completion and explicit induction in an adequate formalism, finding mechanisms, based for instance on generalisation, for suggesting missing lemmas, choosing induction rule in a (semi-)automatic way.

Chapter 23

Enrichment proofs

23.1 Introduction

Programming langages based on many-sorted equational logic or Horn clause logic with equality most often provide also tools for modular and hierarchical algebraic specifications, namely importation, parametrization, and combination of specifications. Such languages are promising for a safe software development with proof assistance, including in industrial applications. However this precludes that automatic or partially automatic verification tools are provided too. We focus here on equational languages (i.e. languages based on equational logic) and examine what kind of proofs can be done.

This chapter is concerned with some properties of programs related to their modularity and with their automatisation using rewriting and completion techniques. In building programs by successive enrichments of existing parts, it is crucial to check that enrichments are protecting the imported parts. In other words, the enrichment creates no junk and no confusion. Each of these requirements is more technically described by two properties respectiveley called sufficient completeness and (relative) consistency. When programs are given by sets of rewrite rules enjoying confluence and termination on ground terms, these properties can effectively be checked.

Parameterization is a generic way for building families of specifications and for reusing specifications. As a matter of fact, there is a strong connection between (protected) enrichment and (persistent) parameterized specifications. Intuitively persistency means that every parameter is protected. Similar rewrite and completion techniques can thus provide effective tools to prove that a parameterized specification is persistent.

An important concern is to also make use of parameterization at the proof level and to develop a generic proof method. So generic proofs are performed in parameterized equational specifications and generic theorems hold for any instance of the parameter. This approach has several advantages: First, it allows performing proofs in a structured way that reflects the program structure. Second, a generic proof must be given only once and can be reused for each instantiation of the parameter.

23.2 Enrichments

We stay in the context of many-sorted specifications presented in Section 22.2 of Chapter 22. Componentwise inclusion of specifications corresponds to enrichments.

Definition 23.1 An *enrichment* of a specification $SP = (\Sigma, E)$ is a specification $SP' = (\Sigma', E')$ such that $\Sigma \subseteq \Sigma'$ and $E \subseteq E'$.

SP is often referred to as the *primitive* or *basic* specification, while SP' is called the *enriched* specification.

Definition 23.2 Sorts and operators of Σ are called *primitive* sorts and operators. A variable whose sort is primitive is called a *primitive variable.* Terms of $\mathcal{T}(\Sigma, \mathcal{X})$, with $\mathcal{X}$ a set of primitive variables, are called *primitive terms.* Equalities that contain only primitive terms are called *primitive equalities.*

Note that a term of primitive sort is not always a primitive term.

A forgetful functor is associated to an enrichment.

Definition 23.3 Assume that $SP \subseteq SP'$. Then the *forgetful functor* $\mathcal{V}$ from $ALG(SP')$ to $ALG(SP)$ is defined as follows:

- $\forall A' \in ALG(SP')$, $A = \mathcal{V}(A')$ is the SP-algebra such that $\forall s \in \Sigma, A_s = A'_s$ and $\forall f \in \Sigma, f_A = f_{A'}$.
- $\forall h'$ SP'-morphism, $h = \mathcal{V}(h')$ is the SP-homomorphism such that $h_s = h'_s$ for any $s \in \Sigma$.

23.2.1 Properties of enrichments

Enrichments are classified according to their effect on the initial algebra of the enriched specification. Mainly, enrichments can produce *junks*, that is new terms that are not equivalent to an already existing term, or *confusions*, that is new equivalences between terms originally distincts.

Definition 23.4 Let $SP = (\Sigma, E) \subseteq SP' = (\Sigma', E')$ be an enrichment.

The enrichment is *consistent* if for any sort $s \in \Sigma$, any ground terms t and t' of sort s in $\mathcal{T}(\Sigma)$, $t \overset{*}{\longleftrightarrow}_E t'$ iff $t \overset{*}{\longleftrightarrow}_{E'} t'$.

The enrichment is *sufficiently complete* if for any sort $s \in \Sigma$, any ground term t' of sort s in $\mathcal{T}(\Sigma')$, there exists a term t of sort s in $\mathcal{T}(\Sigma)$ such that $t \overset{*}{\longleftrightarrow}_{E'} t'$.

An enrichment which is both consistent and sufficiently complete is said *protected*.

Example 23.1 Let consider the following specification (Σ, E) of integers.

$$\begin{array}{rcl} sort & Int & \\ 0: & \mapsto & Int \\ succ: Int & \mapsto & Int \\ pred: Int & \mapsto & Int \end{array}$$

$$\begin{array}{rcl} \forall x: Int, succ(pred(x)) & = & x \\ \forall x: Int, pred(succ(x)) & = & x. \end{array}$$

$\mathcal{T}(\Sigma)/E$ is a set of equivalence classes of terms, whose representatives are

$$\{...pred(pred(0)), pred(0), 0, succ(0), succ(succ(0)), ...\}.$$

- A first enrichment consists in adding a new operator $+$ and equalities for its definition.

  $$+: Int, Int \mapsto Int$$

  $$\begin{array}{rcl} \forall x: Int, x + 0 & = & x \\ \forall x, y: Int, x + succ(y) & = & succ(x + y) \\ \forall x, y: Int, x + pred(y) & = & pred(x + y). \end{array}$$

  This enrichment does not modify the previous set of ground terms. Each term built with the new symbol $+$ can be proved equivalent to a term in $\mathcal{T}(\Sigma)/E$. The enrichment is consistent and sufficiently complete.

- A second enrichment consists in adding to (Σ, E) two equalities to build integers modulo 2 and the enriched specification $SP' = (\Sigma', E')$:

  $$\begin{array}{rcl} succ(succ(x)) & = & x \\ pred(pred(x)) & = & x. \end{array}$$

  Now some elements in $\mathcal{T}(\Sigma)/E$ are made equivalent, for instance $pred(pred(0))$ and 0. Actually in $\mathcal{T}(\Sigma')/E'$, there are only two equivalence classes whose representatives are 0 and $succ(0)$. The enrichment is not consistent but sufficiently complete.

- A third enrichment consists in adding a new constant to build integers with an infinity element:

  $$\infty: \mapsto Int$$

  $$\begin{array}{rcl} succ(\infty) & = & \infty \\ pred(\infty) & = & \infty. \end{array}$$

  This enrichment does not modify $\mathcal{T}(\Sigma)/E$. But a new element exists now, namely the equivalence class of ∞. The enrichment is consistent but not sufficiently complete.

The notions of consistency and sufficient completeness have been introduced by Guttag (see for instance [Gut78]) in a slightly more restrictive framework when the set of functions in the signature Σ can be split into a set of constructors $\mathcal{C}$ and a set of defined functions $\mathcal{D}$. The definition of functions of $\mathcal{D}$ is *sufficiently complete w.r.t.* $\mathcal{C}$ if any ground term is provably equal to a ground *constructor term*, that is a ground term built only with constructors.

Then E can also be split into $E_{\mathcal{C}} \cup E_{\mathcal{D}-\mathcal{C}}$ where $E_{\mathcal{C}}$ is the subset of equalities that contain only constructors and variables. If $E_{\mathcal{C}} = \emptyset$, the constructors are said *free*. The specification is *consistent w.r.t.* $\mathcal{C}$ if for any ground constructor terms s and t, $s \overset{*}{\longleftrightarrow}_E t$ iff $s \overset{*}{\longleftrightarrow}_{E_{\mathcal{C}}} t$.

23.2.2 Sufficient completeness

The sufficient completeness is in general undecidable [KNZ87] but in some cases, it is equivalent to ground reducibility.

The restriction is that terms built on the imported signature must be preserved.

Definition 23.5 Let consider an enrichment $SP = (\Sigma, R) \subseteq SP' = (\Sigma', R')$ with R' a ground convergent term rewriting system on $\mathcal{T}(\Sigma')$. *R' preserves Σ-terms and Σ-sorts* if for any primitive term $t \in \mathcal{T}(\Sigma)$, its R'-normal form is a primitive term in $\mathcal{T}(\Sigma)$, for any term $t \in \mathcal{T}(\Sigma')_s$ of primitive sort $s \in \Sigma$, its R'-normal form is also of primitive sort.

Some sufficient conditions on the rewrite system R' can be proposed to guarantee preservation of Σ-terms and Σ-sorts. For instance, the property will be satisfied as soon as the rewrite rules fulfill the following conditions: $\forall (l \to r) \in R'$, if $l \in \mathcal{T}(\Sigma, \mathcal{X})$, then $r \in \mathcal{T}(\Sigma, \mathcal{X})$ and if l is of sort $s \in \Sigma$, then r is also of sort $s' \in \Sigma$. The property of preserving the Σ-terms and Σ-sorts is in practice very often satisfied. In a structured programming methodology where new functions are defined w.r.t. existing data, this hypothesis does not appear as a restriction.

Proposition 23.1 Let consider an enrichment $SP = (\Sigma, R) \subseteq SP' = (\Sigma', R')$ with R' a ground convergent term rewriting system on $\mathcal{T}(\Sigma')$ and R' preserves Σ-terms. Then the two following propositions are equivalent:

1. $\forall f \in \Sigma' - \Sigma$, whose co-arity is a sort $s \in \Sigma$, $f(x_1, ..., x_n)$ is ground reducible with R',
2. $\forall t' \in \mathcal{T}(\Sigma')$ of sort $s \in \Sigma$, $\exists t \in \mathcal{T}(\Sigma)$ such that $t \overset{*}{\longleftrightarrow}_{R'} t'$.

Proof:

- Let us first prove that (1) implies (2).
Consider a term $t' \in \mathcal{T}(\Sigma')$ of sort $s \in \Sigma$ and its R'-irreducible form t''. Since R' is preserves Σ-sorts, t'' has a primitive sort $s'' \in \Sigma$. Either $t'' \in \mathcal{T}(\Sigma)$ and then $t = t''$, or $t'' \notin \mathcal{T}(\Sigma)$. In this case, t'' contains a subterm $f(u_1, ..., u_n)$, with $u_1, ..., u_n \in \mathcal{T}(\Sigma')$ and $f \in \Sigma' - \Sigma$, which moreover has a co-arity $s' \in \Sigma$. (Either $f(u_1, ..., u_n)$ is t'' of sort s'' which is primitive, or $f(u_1, ..., u_n)$ is a strict-subterm of t'' that may be chosen so that any symbol above is in Σ. Since any symbol in Σ must have its arguments of primitive sorts, $f(u_1, ..., u_n)$ must be of primitive sort s''). The subterm $u = f(u_1, ..., u_n)$ is indeed a ground instance of $f(x_1, .., x_n)$, so is reducible with R', which contradicts the hypothesis that t'' is R'-irreducible.
- Let us now prove that (2) implies (1).
If $f(x_1, .., x_n)$ is not ground reducible for R', there exists a ground substitution σ such that $t' = f(\sigma(x_1), .., \sigma(x_n))$ is irreducible for R'. Assume now that $\exists t \in \mathcal{T}(\Sigma)$ of sort $s \in \Sigma$, such that $t \overset{*}{\longleftrightarrow}_{R'} t'$. Since R' is convergent on $\mathcal{T}(\Sigma \cup X_{SP})$, this implies that t' is the R'-normal form of t. Which is impossible if R' preserves Σ-terms.

□

In this result, the hypothesis for R' to be convergent is essential, as shown by the following counter-example.

Example 23.2 Let consider the following specification (Σ, R) with $R = \emptyset$:

$$\begin{array}{rcl} \textit{sort} & s & \\ c: & \mapsto & s. \end{array}$$

and its enrichment (Σ', R') with

$$\begin{array}{rcl} g : s & \mapsto & s \\ \forall x : s, g(g(x)) & \rightarrow & g(x) \\ \forall x : s, g(g(x)) & \rightarrow & x. \end{array}$$

The term $g(x)$ is not ground reducible with R'. However the enrichment is sufficiently complete because $g(c) \overset{*}{\longleftrightarrow}_{R'} c$.

Also the hypothesis that R' preserves Σ-terms cannot be dropped, as shown next:

Example 23.3 Let consider the following specification (Σ, R) with $R = \emptyset$:

$$\begin{array}{cccc} sort & s & & \\ f: & s & \mapsto & s \\ c: & & \mapsto & s. \end{array}$$

and its enrichment (Σ', E') with

$$\begin{array}{rcl} g : s & \mapsto & s \\ \forall x : s, f(x) & \rightarrow & g(x). \end{array}$$

The term $g(x)$ is not ground reducible with R'. However the enrichment is sufficiently complete because any term that contains g is R'-equivalent to a term without g.

The hypothesis that R' is terminating cannot be dropped as well:

Example 23.4 Let consider the following specification (Σ, R) with $R = \emptyset$:

$$\begin{array}{ccc} sort & s & \\ c: & \mapsto & s. \end{array}$$

and its enrichment (Σ', E') with

$$\begin{array}{rcl} f : s & \mapsto & s \\ \forall x : s, f(x) & \rightarrow & f(f(x)). \end{array}$$

The term $f(x)$ is ground reducible with R'. However the enrichment is not sufficiently complete because any term that contains f cannot be R'-equivalent to a term without f.

Exercice 66 — Let consider the following specification (Σ, R) with $R = \emptyset$:

$$\begin{array}{ccc} sort & s & \\ c: & \mapsto & s. \end{array}$$

and its enrichment (Σ', E') with

$$\begin{array}{rcl} a: & \mapsto & s \\ b: & \mapsto & s \\ a & \rightarrow & c \\ b & \rightarrow & c. \end{array}$$

What can be said about this enrichment?
Answer: The enrichment is consistent and sufficiently complete.

23.2.3 Consistent enrichments

There exists a strong relation between the notion of consistent enrichment and inductive theorems. This connection is explicited in the following result:

Proposition 23.2 Let $SP = (\Sigma, E) \subseteq SP' = (\Sigma, E')$ be an enrichment with only new equalities: $E' = E \cup E_0$. The enrichment is consistent iff any equality in E_0 is an inductive consequence of E, i.e. E_0 is consistent with E.

Proof: Let $(u = v) \in E_0$. For any ground substitution σ, $\sigma(u) \overset{*}{\longleftrightarrow}_{E_0} \sigma(v)$ and since $E_0 \subseteq E'$, $\sigma(u) \overset{*}{\longleftrightarrow}_{E'} \sigma(v)$. Then by consistency of the enrichment, $\sigma(u) \overset{*}{\longleftrightarrow}_{E} \sigma(v)$.

Conversely, assume that t, t' are in $\mathcal{T}(\Sigma)$ and satisfy $t \overset{*}{\longleftrightarrow}_{E'} t'$. In order to prove that $t \overset{*}{\longleftrightarrow}_{E} t'$, it is enough to prove that whenever $t \longleftrightarrow_{E_0} t'$, then $t \overset{*}{\longleftrightarrow}_{E} t'$. This is clear, since then, there exist $(u = v) \in E_0$, a ground substitution σ and an occurrence ω in t such that $t_{|\omega} = \sigma(u)$ and $t'_{|\omega} = \sigma(v)$. Then $\sigma(u) \overset{*}{\longleftrightarrow}_{E} \sigma(v)$, which implies $t \overset{*}{\longleftrightarrow}_{E} t'$. $\Box$

It must be emphasized that this result is no more true if the enrichment adds new sort symbols or new function symbols, as proved by the following counter-example.

Example 23.5 Let consider the following specification (Σ, E)

$$\begin{array}{rcl} sort & Int & \\ 0: & \mapsto & Int \\ s: Int & \mapsto & Int \end{array}$$

$$succ(succ(0)) = 0$$

and its enrichment (Σ', E') with

$$pred: Int \mapsto Int$$

$$\forall x: Int, pred(x) = succ(x).$$

This enrichment is of course consistent but the equality $pred(x) = succ(x)$ is not an inductive theorem of E.

The reason is that Σ also defines the language of assertions. So an equality that contains a function symbol or a sort in $\Sigma' - \Sigma$ cannot be an assertion of $SP = (\Sigma, E)$.

Turning back to the case where the theories are presented by ground convergent term rewriting systems, the completion process appears as the important tool to both prove consistency of an enrichment and produce simultaneously a ground convergent term rewriting system for the enriched specification.

23.2.4 Completion process for consistency proof

Let us consider an enrichment $SP = (\Sigma, R_0) \subseteq SP' = (\Sigma', R_0 \cup E_0)$ with R_0 a ground convergent term rewriting system on $\mathcal{T}(\Sigma)$. The general idea is to complete $(R_0 \cup E_0)$ into a ground convergent system R' on $\mathcal{T}(\Sigma')$ and to check that whenever a rewrite rule, whose left and right-hand sides both belong to $\mathcal{T}(\Sigma)$, is added, then this rule is an inductive consequence of R_0. Transition rules for a completion process that checks consistency of enrichments are the same as transition rules for inductive completion. The only modification is the condition for applying the orientation rule *Orient*. More precisely, let $\sqsupset$ be defined by $s \sqsupset t$ if

1. $p > q$ and
2. either $p \notin \mathcal{T}(\Sigma, \mathcal{X})$, or $p, q \in \mathcal{T}(\Sigma, \mathcal{X})$ and p ground reducible with R_0.

Let P be a set of equalities (quantified pairs of terms), R_0 the initial convergent rewriting system, R be the current rewriting system, and $>$ a reduction ordering that contains R_0. The completion procedure for consistency is expressed by the set of transition rules $\mathcal{CE}$ presented in Figure 23.1.

If a sucessful completion sequence starting from (P_0, R_0) is found, R_0 and R_∞ define the same normal forms on ground terms of $\mathcal{T}(\Sigma)$. This implies that the enrichment is consistent. Moreover if the process generates a rule whose left-hand side is not ground reducible by R_0, the enrichment is inconsistent.

Theorem 23.1 *Let consider an enrichment $SP = (\Sigma, R_0) \subseteq SP' = (\Sigma', R_0 \cup E_0)$ with R_0 a ground convergent term rewriting system on $\mathcal{T}(\Sigma)$. Let $P_0 = E_0$ and $>$ be a reduction ordering that contains R_0. If $(P_0, R_0) \mapsto\!\!\!\twoheadrightarrow (P_1, R_1) \mapsto\!\!\!\twoheadrightarrow ....$ is a derivation such that $P_\infty = \emptyset$, R_∞ is reduced and $CP(R_\infty)$ is a subset of P_*, then R_∞ is convergent on $\mathcal{T}(\Sigma')$ and the enrichment is consistent.*

If $(P_0, R_0) \mapsto\!\!\!\twoheadrightarrow (P_1, R_1) \mapsto\!\!\!\twoheadrightarrow ....(P_i, R_i)$ is a derivation such that P_i contains an equality $(l = r)$ with $l, r \in \mathcal{T}(\Sigma)$, $l > r$ and l non ground reducible with R_0, then the enrichment is inconsistent.

$$\begin{array}{llll}
\textbf{Orient} & P \cup \{p = q\}, R & \Mapsto & P, R \cup \{p \to q\} \\
 & & & \text{if } p \supset q \\
\textbf{Deduce} & P, R & \Mapsto & P \cup \{p = q\}, R \\
 & & & \text{if } (p, q) \in CP(R) \\
\textbf{Simplify} & P \cup \{p = q\}, R & \Mapsto & P \cup \{p' = q\}, R \\
 & & & \text{if } p \to_R p' \\
\textbf{Delete} & P \cup \{p = p\}, R & \Mapsto & P, R \\
\textbf{Compose} & P, R \cup \{l \to r\} & \Mapsto & P, R \cup \{l \to r'\} \\
 & & & \text{if } r \to_R r' \\
\textbf{Collapse} & P, R \cup \{l \to r\} & \Mapsto & P \cup \{l' = r\}, R \\
 & & & \text{if } l \to_R^{g \to d} l' \ \& \ l \to r >> g \to d
\end{array}$$

Figure 23.1: Consistent enrichment by completion

Proof: Let consider the case where the completion process does not stop with a non-orientable equality. Then R_∞ is convergent on $\mathcal{T}(\Sigma')$. Let $t, t' \in \mathcal{T}(\Sigma)$, $t \overset{*}{\longleftrightarrow}_{R_0 \cup P_0} t'$ iff $t \overset{*}{\longleftrightarrow}_{R_\infty} t'$ iff $t \downarrow_{R_\infty} = t' \downarrow_{R_\infty}$. But $t \downarrow_{R_\infty} = t \downarrow_{R_0}$ and $t' \downarrow_{R_\infty} = t' \downarrow_{R_0}$. Thus $t \overset{*}{\longleftrightarrow}_{R_0} t'$.

If an equality $(l = r)$, with $l, r \in \mathcal{T}(\Sigma)$, $l > r$ and l non ground reducible with R_0, is generated, then there exists a ground substitution σ such that $\sigma(l)$ is R_0-irreducible. If $(l = r)$ is an inductive theorem, then $\sigma(l) \downarrow_{R_0} = \sigma(r) \downarrow_{R_0}$ and $\sigma(r) \overset{*}{\longrightarrow}_{R_0} \sigma(l)$. Thus $\sigma(r) \geq \sigma(l)$, which contradicts $l > r$ and $\sigma(l) > \sigma(r)$. So the enrichment is not consistent because $\sigma(l) \overset{*}{\longleftrightarrow}_{R_0 \cup P_0} \sigma(r)$ and not $\sigma(l) \overset{*}{\longleftrightarrow}_{R_0} \sigma(r)$. $\Box$

Note that if there exists an equality $(l = r)$ such that $l \in \mathcal{T}(\Sigma)$, $r \notin \mathcal{T}(\Sigma)$ and $l > r$, the process stops with failure, but nothing can be precluded. This does not mean that the enrichment is not consistent, as shown in the following example.

Example 23.6 Consider again the following specification (Σ, E)

$$\begin{array}{rcl}
sort & Int & \\
0: & \mapsto & Int \\
succ: Int & \mapsto & Int
\end{array}$$

$$succ(succ(0)) \to 0$$

and its enrichment (Σ', E') with

$$pred: Int \mapsto Int$$

$$\forall x: Int, pred(x) = succ(x).$$

Assume in addition that the given reduction ordering $>$ satisfies $succ(x) > pred(x)$. Although this enrichment is of course consistent, the process will stop with failure.

The way out is of course to design an unfailing completion process, since in addition the interesting property for the equalities of the enrichment is the Church-Rosser property on ground terms and the proof by consistency of every equation generated in the primitive specification.

23.2.5 An unfailing completion process for consistency proof

In the completion process for consistency proof, the set of equalities is split into two parts: one, called C, which contains only primitive equalities, and the other, called P, that contains non-primitive ones. The enrichment is consistent iff, at each step, all equalities in C are consistent with R. In order to prove that, a proof by induction procedure is needed, for instance the one given by Bachmair [Bac88] which relies on the operational notion of *provable inconsistency.*

$OCP(P \cup R, P)$ denote the set of ordered critical pairs obtained by superposition of $P \cup R$ on P and conversely. $CP(R, C)$ as usual denote the set of critical pairs obtained by superposition of R on C.

In order to cover a larger class of proofs, introduction of lemmas is allowed in the imported theory.

Let P be a set of equalities (quantified pairs of terms), C be a set of conjectures (primitive equalities), R the implicit rewrite system, terminating and confluent on ground primitive terms, L be a set of inductive lemmas (primitive equalities) consistent with R, and $>$ a reduction ordering that contains R. The consistency proof procedure is expressed by the set $\mathcal{UC}$ of transition rules given in Figure 23.2.

Deduce P, C, L
$\mapstochar\twoheadrightarrow$
$P \cup \{p = q\}, C, L$
if $(p, q) \in OCP(P \cup R, P)$

Deflation $P \cup \{p = q\}, C, L$
$\mapstochar\twoheadrightarrow$
$P, C \cup \{p = q\}, L$
if p *and* q *primitive*

Delete $P \cup \{p = p\}, C, L$
$\mapstochar\twoheadrightarrow$
P, C, L

Collapse $P \cup \{p = q\}, C, L$
$\mapstochar\twoheadrightarrow$
$P \cup \{p' = q\}, C, L$
if $(p \rightarrow^{l \rightarrow r}_{R \cup P>} p'$ with $p \sqsupset l)$ or $(p = l$ and $q > r)$

Conjecture P, C, L
$\mapstochar\twoheadrightarrow$
$P, C \cup \{p = q\}, L$
if $(p, q) \in CP(R, C)$

Induce P, C, L
$\mapstochar\twoheadrightarrow$
$P, C, L \cup \{p = q\}$
if $(p = q) \in ITh(R)$ & p, q *primitive*

Discharge $P, C \cup \{p = q\}, L$
$\mapstochar\twoheadrightarrow$
P, C, L
if $(p = q) \in ITh(R)$

Simplify $P, C \cup \{p = q\}, L$
$\mapstochar\twoheadrightarrow$
$P, C \cup \{p' = q\}, L$
if $p > p'$ & $p \overset{+}{\longleftrightarrow}_{R \cup L} p'$

Compose $P, C \cup \{p = q\}, L$
$\mapstochar\twoheadrightarrow$
$P, C \cup \{p' = q\}, L$
if $p > p'$ & $p \longleftrightarrow^{g=d}_{C} p'$ & $p \sqsupset g > d$

Disproof $P, C \cup \{p = q\}, L$
$\mapstochar\twoheadrightarrow$
Disproof
if $(p = q)$ *provably inconsistent*

Figure 23.2: Unfailing consistency rules

Note that this set of transition rules is more general than the sets $\mathcal{U}$ for unfailing completion, obtained as a subset when $C = L = \emptyset$, and $\mathcal{IC}$ for proof by consistency, obtained for $P = \emptyset$.

The soundness of this transition system $\mathcal{UC}$ is given by the following lemmas.

Lemma 23.1 If $(P, C, L) \mapstochar\twoheadrightarrow (P', C', L')$, then the congruences on ground terms $\overset{*}{\longleftrightarrow}_{P \cup C \cup L \cup R}$ and $\overset{*}{\longleftrightarrow}_{P' \cup C' \cup L' \cup R}$ coincide.

Proof: by looking at the different transition rules. □

Lemma 23.2 If $(P,C,L) \mapsto\!\!\!\rightarrow (P',C',L')$, then $=_{ind(P\cup C\cup L\cup R)}$ and $=_{ind(P'\cup C'\cup L'\cup R)}$ coincide.

Proof: It is a consequence of Lemma 23.1. □

The considered set of proofs are proofs on ground terms using equalities in $P \cup C \cup L \cup R$. The goal is to transform such a proof $s \overset{*}{\longleftrightarrow}_{P\cup C\cup L\cup R} t$ into a proof of the form

$$s \overset{*}{\longrightarrow}_{R\cup P^>} s' \overset{*}{\longleftrightarrow}_R t' \overset{*}{\longleftarrow}_{R\cup P^>} t$$

or to find a provable inconsistency.

For that, the proof transformation relation must eliminate several kinds of subproofs:

- First of all, since L is a set of inductive lemmas in R, any use of a lemma to prove an equality can be replaced by a sequence of R-equality steps.
- The same remark holds for an equality step using a conjecture in C, provided this conjecture is valid in R.
- So the proof by refutation of conjectures in C is performed and this amounts reducing ground proofs $s' \overset{+}{\longleftarrow}_R \sigma(g) \longleftrightarrow_C \sigma(d) \overset{*}{\longrightarrow}_R t'$ to $\tau(u) \longleftrightarrow_{C'} \tau(v)$ where $(u = v) \in C'$, $\tau(u)$ R-irreducible and either $\tau(v)$ R-irreducible or $u > v$. Either a provable inconsistency is detected and the process stops, or when all conjectures are marked, this means that C-equality steps can be replaced by R-equality steps. Moreover whenever a conjecture is proved, it can be added to the set of inductive lemmas L and thus used for simplification.
- Finally peaks between R and P are eliminated thanks to the ordered critical pairs computation between rules of R and $P^>$.

The proof reduction relation that reflects the transition rule system $\mathcal{UC}$, is now defined.

1. Deduce: $t' \longleftarrow^{g=d}_{P^>\cup R} t \longrightarrow^{l=r}_{P^>} t'' \Longrightarrow t' \longleftrightarrow^{p=q}_{P} t''$
2. Deflation: $t \longleftrightarrow^{p=q}_{P} t' \Longrightarrow t \longleftrightarrow^{p=q}_{C} t'$
3. Delete: $t \longleftrightarrow^{p=p}_{P} t' \Longrightarrow \Lambda$ where Λ is the empty proof.
4. Collapse: $t \longleftrightarrow^{p=q}_{P} t' \Longrightarrow t \longrightarrow^{l\to r}_{P^>\cup R} t'' \longleftrightarrow^{p'=q}_{P} t'$.
5. Conjecture: $t' \longleftarrow^{l\to r}_{R} t \longleftrightarrow^{g=d}_{C} t'' \Longrightarrow t' \longleftrightarrow^{p=q}_{C} t''$
6. Induce: $t \overset{+}{\longleftrightarrow}_R t' \Longrightarrow t \longleftrightarrow^{p=q}_{L} t'$
7. Discharge: $t \longleftrightarrow^{p=q}_{C} t' \Longrightarrow t \overset{*}{\longleftrightarrow}_R t'$
8. Simplify: $t \longleftrightarrow^{p=q}_{C} t' \Longrightarrow t \overset{+}{\longleftrightarrow}_R t'' \longleftrightarrow^{p'=q}_{C} t'$
9. Compose: $t \longleftrightarrow^{p=q}_{C} t' \Longrightarrow t \longleftrightarrow^{g=d}_{C} t'' \longleftrightarrow^{p'=q}_{C} t'$
10. Peak without overlap: $t' \longleftarrow^{g=d}_{R\cup P^>} t \longrightarrow^{l=r}_{P^>} t'' \Longrightarrow t' \longrightarrow^{l=r}_{P^>} t_1 \longleftarrow^{g=d}_{R\cup P^>} t''$
11. Peak with variable overlap: $t' \longleftarrow^{g=d}_{R\cup P^>} t \longrightarrow^{l=r}_{P^>} t'' \Longrightarrow t' \overset{*}{\longrightarrow}_{R\cup P^>} t_1 \overset{*}{\longleftarrow}_{R\cup P^>} t''$

Lemma 23.3 If $>$ is a reduction ordering that can be extended to a reduction ordering $\gg$ on $\mathcal{T}(\Sigma')$ total on E'-equivalence classes, then the proof reduction relation is noetherian.

Proof: Let define the complexity measure of elementary proof steps by:

$$\begin{array}{lcll}
c(s \longleftrightarrow^{g=d}_{P} t) & = & (2,s,g,t) & if \quad s \gg t \\
c(s \longleftrightarrow^{g=d}_{P} t) & = & (2,t,d,s) & if \quad t \gg s \\
c(s \longleftrightarrow^{g=d}_{C} t) & = & (1,s,g,t) & if \quad s \gg t \\
c(s \longleftrightarrow^{g=d}_{C} t) & = & (1,t,d,s) & if \quad t \gg s \\
c(s \rightarrow^{g\to d}_{R} t) & = & (0,s,g,t) &
\end{array}$$

Let $\gg \cup \rhd_{sub}$ denote the union of the complete reduction ordering $\gg$ and the strict subterm ordering $\rhd_{sub}$. Complexities of elementary proof steps are compared using the lexicographic combination, denoted $>_{ec}$ of the standard ordering on natural numbers, $\gg \cup \rhd_{sub}$, $\sqsupset$ and $\gg \cup \rhd_{sub}$.

The complexity of a non-elementary proof is the multiset of the complexities of elementary proof steps that it contains. Complexities of non-elementary proofs are compared using the multiset extension $>_c$ of $>_{ec}$. □

An unfailing completion procedure that handles all the ordered critical pairs of $(P \cup R)$ on P produces a set of rules and equalities that has the Church-Rosser property on $\mathcal{T}(\Sigma')$. Moreover if all critical pairs of R on equalities in C are computed and no inconsistency is generated, then the enrichment is consistent.

Theorem 23.2 *Let consider an enrichment $SP = (\Sigma, E) \subseteq SP' = (\Sigma', E')$ with R a ground convergent term rewriting system on $\mathcal{T}(\Sigma)$ presenting E. Let split the set of equalities $E' - E$ added in the enrichment, into P_0 and C_0 such that C_0 contains only primitive equalities and P_0 contains all other equalities.*

Let $>$ be a reduction ordering that contains R and can be extended to a reduction ordering $\gg$ on $\mathcal{T}(\Sigma')$ total on E'-equivalence classes, such that no primitive term is greater than a non-primitive one.

If $(P_0, C_0, L_0) \mapsto\!\!\!\rightarrow (P_1, C_1, L_1) \mapsto\!\!\!\rightarrow ....$ is a derivation such that $OCP(P_\infty \cup R, P_\infty) \cup CP(R, C_\infty)$ is a subset of $P_* \cup C_*$. Then

- *If C_∞ is empty, then $(P_\infty \cup R)$ is Church-Rosser with respect to $\gg$ on $\mathcal{T}(\Sigma')$ and the enrichment is consistent.*
- *If there exists a step i and an equality in C_i that is provably inconsistent, then the enrichment is not consistent.*

Proof:

- If C_∞ is empty, let us prove by induction on $\Longrightarrow$ that for any $i \geq 0$, for any proof $s \overset{*}{\longleftrightarrow}_{P_i \cup C_i \cup L_i \cup R} t$, there exists a proof

$$s \overset{*}{\longrightarrow}_{R \cup P^\infty} s' \overset{*}{\longleftrightarrow}_R t' \overset{*}{\longleftarrow}_{R \cup P^\infty} t.$$

  Since no provable inconsistency has been found, this means that C_i is consistent with R and the C_i-equality steps can be replaced by R-equality steps. The same remark holds for L_i-steps. So if the proof $s \overset{*}{\longleftrightarrow}_{P_i \cup C_i \cup L_i \cup R} t$ is not already of the desired form, then either it contains a peak of $R \cup P^\infty$, or a non-persisting equality. In both cases, the proof is reducible by $\Longrightarrow$ into $s \overset{*}{\longleftrightarrow}_{P_j \cup C_j \cup L_j \cup R} t$ and the induction hypothesis gives the result.

  Given two ground primitive terms, s and t, such that $s \overset{*}{\longleftrightarrow}_{P_0 \cup C_0 \cup L_0 \cup R} t$, or equivalently $s \overset{*}{\longleftrightarrow}_{P_0 \cup C_0 \cup R} t$, there exists a proof

$$s \overset{*}{\longrightarrow}_{R \cup P_\infty} s' \overset{*}{\longleftrightarrow}_R t' \overset{*}{\longleftarrow}_{R \cup P_\infty} t.$$

  But since s and t are primitive and greater then any other term appearing in the proof, all these terms must be primitive. So no equality in P_∞ can apply since P_∞ contains non-primitive terms. So we get $s \overset{*}{\longleftrightarrow}_R t$. Then the enrichment is of course consistent.
- If a provable inconsistency is detected at some step k, there exists a proof $\tau(g) \longleftrightarrow_{C_k} \tau(d)$, such that $\tau(g)$ is R-irreducible and $\tau(d)$ is R-irreducible or $g > d$. So we get $\tau(g) \overset{*}{\longleftrightarrow}_{P_0 \cup C_0 \cup L_0 \cup R} \tau(d)$ and not $\tau(g) \overset{*}{\longleftrightarrow}_R \tau(d)$, which proves that the enrichment is not consistent.

□

The following procedure `CONSISTENT`, described in Figure 23.3, implements the proof of consistency of an enrichment. L has been chosen empty and R is implicit. It has three possible issues: success when the enrichment is consistent, disproof when there exists an inconsistency and it may not terminate and generate an infinite number of equalities.

The `SIMPLIFICATION` procedure computes simplified forms of l and r, according to transition rules **Collapse**, **Simplify** and **Compose**.

The `CRITICAL-PAIRS` procedure computes all superpositions of R on the equality $(l' = r')$ if $(l' = r')$ is in $\mathcal{T}(\Sigma)$, or oriented critical pairs of $(l' = r')$ with $P \cup R$. The equality is marked whenever all these critical pairs have been computed and added to C if they are primitive or to P otherwise.

Another procedure is proposed, that gives priority to detection of inconsistencies. This is achieved by a sub-procedure `PROVE` that implements the proof by consistency procedure described by the set of transition

```
PROCEDURE CONSISTENT (P, C, >)
IF all equalities in P U C are marked
THEN RETURN P; STOP with SUCCESS
ELSE Choose an unmarked equality (l = r) fairly in P U C;
     P U C := (P U C)-{(l=r)};
     (l'=r') := SIMPLIFICATION (l = r, P, C, R);
     CASE l' = r'  THEN P := CONSISTENT(P, C, >)
          l',r' primitive AND l' =ind(R) r'
                  THEN P := CONSISTENT(P, C, >)
          l',r' primitive AND (l'= r') provably inconsistent
                  THEN STOP with DISPROOF;
          ELSE P U C := P U C U {(l'=r')}
                           U CRITICAL-PAIRS (P,R,l'=r');
               Mark the equality (l'=r') in P U C;
               P := CONSISTENT(P, C, >)
     END CASE
END IF
END CONSISTENT
```

Figure 23.3: A consistency completion procedure

rules $\mathcal{IC}$. The control is given back to `PROVE` each time new conjectures built on primitive terms are found. The procedure is described in Figure 23.4.

Exercice 67 — Consider the following specification SP_0

$$\begin{array}{rcl} sort & Nat & \\ 0: & \mapsto & Nat \\ s: Nat & \mapsto & Nat \\ -: Nat, Nat & \mapsto & Nat \\ \forall n: Nat,\ n-0 & = & n \\ \forall n: Nat,\ 0-n & = & 0 \\ \forall n,m: Nat,\ s(n)-s(m) & = & n-m \\ \forall n: Nat,\ n-n & = & 0. \end{array}$$

Prove that that these four equalities define a convergent rewrite system.

Consider the following enrichment SP_1 of SP_0 by:

$$\begin{array}{rcl} a: & \mapsto & Nat \\ \forall n: Nat,\ s(a) & = & a \end{array}$$

Prove that this enrichment is consistent.

Consider the second enrichment SP of SP_0 by:

$$\forall n: Nat,\ s(n)-n \quad = \quad s(0)$$

Prove that there still exists a convergent rewrite system for SP.

Prove that the enrichment SP_2 of SP by

$$\begin{array}{rcl} a: & \mapsto & Nat \\ \forall n: Nat,\ s(a) & = & a \end{array}$$

is inconsistent now.

Answer: In the first enrichment SP_1 the superposition of $s(a) = a$ with $s(n)-s(m) = n-m$ gives two new equalities

$$\begin{array}{rcl} a-s(m) & = & a-m \\ s(n)-a & = & n-a \end{array}$$

Other critical pairs are convergent.

In the second enrichment SP_2, there is also a superposition with $s(n) - n = s(0)$, that gives $a - a = s(0)$, and the inconsistency $0 = s(0)$.

Exercice 68 — A specification $SP' = (\Sigma', R')$ is *operationally sufficiently complete* w.r.t. $SP = (\Sigma, R)$ if for any sort $s \in \Sigma$, any ground terms t' of sort s in $\mathcal{T}(\Sigma')$, there exists a term t of sort s in $\mathcal{T}(\Sigma)$ such that $t \xrightarrow{*}_{R'} t'$.

```
PROCEDURE CONSISTENT (P, C, L, >)
PROVE(C, L, >);
IF DISPROOF THEN STOPS with INCONSISTENT
ELSE L := C; C := emptyset;
     IF all equalities in P are marked
     THEN RETURN P; STOP with SUCCESS
     ELSE Choose an unmarked equality (l = r) fairly in P;
          P := P - {(l=r)};
          (l'=r') := SIMPLIFICATION (l = r, P, C, L, R);
          CASE l' = r'  THEN P := CONSISTENT(P, C, L, >)
               l',r' primitive
                        THEN C := C  U {(l'= r')};
                             P := CONSISTENT(P, C, L, >)
               ELSE P := P U {(l'=r')} U CRITICAL-PAIRS (P,R,l'=r');
                    Mark the equality (l'=r') in P;
                    P := CONSISTENT(P, C, L, >)
          END CASE
     END IF
END IF
END CONSISTENT
```

Figure 23.4: Another consistency completion procedure

1. Prove that SP' operationally sufficiently complete w.r.t. SP implies SP' sufficiently complete w.r.t. SP.
2. Prove that the converse does not hold.
3. Prove that if R is confluent on $\mathcal{T}(\Sigma')$ and if $\mathcal{T}(\Sigma)$ is closed by $\rightarrow_{R'}$, then SP' sufficiently complete w.r.t. SP implies SP' operationally sufficiently complete w.r.t. SP.
4. Let SP' be operationally sufficiently complete w.r.t. SP. Assume that R is confluent on $\mathcal{T}(\Sigma')$ and that terms in $\mathcal{T}(\Sigma)$ are irreducible by $R' - R$. Prove that the two following properties are equivalent:
   - SP' is consistant w.r.t. SP
   - R' is confluent on $\mathcal{T}(\Sigma')$
5. Prove that this equivalence is no more true if SP' is only sufficiently complete w.r.t. SP.

Answer:

1. Easy because $\xrightarrow{*}_{R'} \subseteq \longleftrightarrow *_{R'}$.
2. Consider $S = \{s\}, \mathcal{F} = \{a_0\}, R = \emptyset$ for SP and $S' = \{s\}, \mathcal{F}' = \{a, b\}, R = \{b \to a, b \to a_0\}$ for SP', with a_0, a, b nullary operators.
3. If $t \longleftrightarrow *_{R'} t'$, there exists t'' such that $t \xrightarrow{*}_{R'} t'' \xleftarrow{*}_{R'} t'$ and $t'' \in \mathcal{T}(\Sigma)$.
4. Let SP' be operationally sufficiently complete w.r.t. SP. Assume that R is confluent on $\mathcal{T}(\Sigma')$ and that terms in $\mathcal{T}(\Sigma)$ are irreducible by $R' - R$.
   - If SP' is consistent w.r.t. SP, consider $t \xleftarrow{*}_{R'} t'' \xrightarrow{*}_{R'} t'$, whic implies $t \longleftrightarrow *_R t'$ and thus $t \xrightarrow{*}_R u \xleftarrow{*}_R t'$ which yields the confluence of R' since $R \subseteq R'$.
   - Conversely if R' is confluent on $\mathcal{T}(\Sigma')$, for $t, t' \in \mathcal{T}(\Sigma)$, $t \longleftrightarrow *_{R'} t'$ implies $t \xrightarrow{*}_{R'} u \xleftarrow{*}_{R'} t'$ and $t \xrightarrow{*}_R u \xleftarrow{*}_R t'$. So If SP' is consistent w.r.t. SP.
5. Consider $S = \{s\}, \mathcal{F} = \{a_0\}, R = \emptyset$ for SP and $S' = \{s\}, \mathcal{F}' = \{a_0, a, b\}, R = \{b \to a, b \to a_0\}$ for SP', with a_0, a, b nullary operators.

23.3 Parameterization

In this section we first establish the strong connection between (protected) enrichment and (persistent) parameterized specifications. This makes possible to reduce a persistency proof to a proof of a protected enrichment, using an adequate notion of generic ground reducibility. Moreover the rewrite and completion techniques used in Section 23.2 are easily adapted.

Persistency is a property important at the semantic level, but also for theorem proving: an equational theorem holds in the generic theory of the parameterized specification if and only if it is an inductive theorem in a particular initial algebra, called the generic algebra. Provided persistency, such a theorem is generic: for any specification morphism m, the translated equality using m holds in the initial algebra of the instantiated specification.

Tehniques for inductive proofs can thus be generalized and provide effective tools to prove that a parameterized specification is persistent, to prove generic theorems and to prove generic ground reducibility.

23.3.1 Parameterized specifications

Definition 23.6 A parameterized specification PSP is a pair (SP, SP') of specifications where $SP = (\Sigma, E)$ is called the *formal parameter*, $SP' = SP + (\Sigma'', E'')$ is called the *target specification*, and (Σ'', E'') is called the *body* of the parameterized specification.

Terms built on the signature Σ and a denumerable set X_{SP} of *parameter variables*, i.e. variables of sort $s \in \Sigma$, are called *parameter terms*.

Example 23.7 Let us axiomatize the list structure on any kind of elements. Classically "nil'' is the empty list and "$cons''$ the constructor for lists. Concatenation of lists is denoted by the function "$append''$. In order to define a product operation on lists that computes the product of its elements, it is needed to constrain the elements to be in a monoid with an identity element. Then the formal parameter SP is the following specification MONOID, in which equalities have been oriented into rewrite rules:

$$\begin{array}{rcl}
sort & Elem & \\
id: & \mapsto & Elem \\
\pi : Elem, Elem & \mapsto & Elem \\
\forall e : Elem,\ \pi(e, id) & \rightarrow & e \\
\forall e : Elem,\ \pi(id, e) & \rightarrow & e \\
\forall e_1, e_2, e_3 : Elem,\ \pi(\pi(e_1, e_2), e_3) & \rightarrow & \pi(e_1, \pi(e_2, e_3)).
\end{array}$$

Let us consider the following parameterized specification with the formal parameter MONOID and the body:

$$\begin{array}{rcl}
sort & List & \\
cons : Elem, List & \mapsto & List \\
nil: & \mapsto & List \\
append : List, List & \mapsto & List \\
prod : List & \mapsto & Elem \\
\forall l : List,\ append(nil, l) & \rightarrow & l \\
\forall e : Elem, l, l' : List,\ append(cons(e, l), l') & \rightarrow & cons(e, append(l, l')) \\
\forall l : List,\ prod(nil) & \rightarrow & id \\
\forall e : Elem, l : List,\ prod(cons(e, l)) & \rightarrow & \pi(e, prod(l)).
\end{array}$$

23.3.2 Semantics

Semantics for parameterized specifications have been widely studied, for instance in the many-sorted case in [EM85, Gan83, Pad88] and in the order-sorted case in [Poi86, GMP83, Gog84]. The case of Horn clauses parameterized specifications is considered for example in [Gan87, NO87].

To give a semantics to a parameterized specification consists in associating to SP the class $ALG(SP)$ with its SP-homomorphisms, to SP' the class $ALG(SP')$ with its SP'-homomorphisms and to PSP a functor from the category $ALG(SP)$ to $ALG(SP')$.

Let $\mathcal{V}$ be the forgetful functor associated to the enrichment $SP \subseteq SP'$. From a given SP-algebra $\mathcal{A} \in ALG(SP)$, a SP'-algebra denoted $\mathcal{T}_{SP'}(\mathcal{A})$ can be explicitely built, in the following way: let

- $Const(\mathcal{A}) = \{a :\mapsto s \mid a \in \mathcal{A}_s, s \in \Sigma\}$.
- $Eqns(\mathcal{A}) = \{f(a_1, ..., a_n) = f_{\mathcal{A}}(a_1, ..., a_n) \mid \forall a_i \in \mathcal{A}, \forall f \in \Sigma\}$.
- $SP'(\mathcal{A}) = (\Sigma' \cup Const(\mathcal{A}), E' \cup Eqns(\mathcal{A}))$.
- $\mathcal{T}_{SP'}(\mathcal{A}) = \mathcal{V}_{\mathcal{A}}(\mathcal{T}_{SP'(\mathcal{A})})$, where $\mathcal{V}_{\mathcal{A}}$ is the forgetful functor from the category of the $SP'(\mathcal{A})$-algebras to the category of the SP'-algebras.

$\mathcal{T}_{SP'}(\mathcal{A})$ is *the free construction* on $\mathcal{A}$ w.r.t. $\mathcal{V}$. The concepts of free construction and free functor are precisely defined for instance in [EM85].

Theorem 23.3 *[EM85] Let $\mathcal{V}$ be the forgetful functor from $ALG(SP')$ to $ALG(SP)$. For any $\mathcal{A} \in ALG(SP)$, let $\mathcal{F}(\mathcal{A})$ be defined as $\mathcal{T}_{SP'}(\mathcal{A})$. Then $\mathcal{F}$ extends to a free functor from $ALG(SP)$ to $ALG(SP')$ called the* free functor *w.r.t. $\mathcal{V}$.*

Definition 23.7 $\mathcal{F}$, the free functor w.r.t. $\mathcal{V}$, is the *semantics of the parameterized specification PSP.*

Note that a first kind of genericity is obtained with the free functor: from a class $ALG(SP)$ of SP-algebras, the free functor $\mathcal{F}$ generates the class of algebras

$$\mathcal{F}(ALG(SP)) = \{\mathcal{F}(\mathcal{A}) \mid \mathcal{A} \in ALG(SP) \}.$$

A second kind of genericity obtained from a parameterized specification, is to generate specifications and, for this purpose, the notion of parameter passing is necessary.

23.3.3 Parameter passing

Parameter passing is intended to formalize the instantiation of the formal parameter specification SP into an actual specification SP_1.

Definition 23.8 A *signature morphism* m from Σ to Σ_1 is a function $m : \Sigma \mapsto \Sigma_1$ such that:

$$\forall f : s_1, ...s_n \mapsto s \in \Sigma, m(f) : m(s_1), ..., m(s_n) \mapsto m(s) \in \Sigma_1.$$

Given a signature morphism m from Σ to Σ_1 and a Σ-axiom $e = (t = t')$, the *translated axiom* denoted $m^*(e) = (m^*(t) = m^*(t'))$ is inductively defined by

$$m^*(x : s) = x : m(s) \text{ and } m^*(f(t_1, ..., t_n)) = m(f)(m^*(t_1), ..., m^*(t_n)).$$

Definition 23.9 *A* specification morphism m *from $SP = (\Sigma, E)$ to $SP_1 = (\Sigma_1, E_1)$ is a signature morphism from Σ to Σ_1 such that for any axiom $e \in E$, the translated axiom $m^*(e)$ is valid in the initial SP_1-algebra.*

A forgetful functor is associated to a specification morphism m.

Definition 23.10 Assume that m is a specification morphism from $SP = (\Sigma, E)$ to $SP_1 = (\Sigma_1, E_1)$. Then the *forgetful functor* $\mathcal{V}_m$ from $ALG(SP_1)$ to $ALG(SP)$ is defined as follows:

- $\forall A' \in ALG(SP_1)$, $A = \mathcal{V}_m(A')$ is the SP-algebra such that $\forall s \in \Sigma, A_s = A'_{m(s)}$ and $\forall f \in \Sigma, f_A = m(f)_{A'}$.
- $\forall h'$ SP_1-morphism, $h = \mathcal{V}(h')$ is the SP-homomorphism such that $h_s = h'_{m(s)}$ for any $s \in \Sigma$.

Given a specification morphism from the formal parameter specification to the actual parameter specification, an instantiated specification can be built.

Definition 23.11 Given a parameterized specification $PSP = (SP, SP')$ and a specification morphism m from SP to SP_1, a *parameter passing* is given by:

- The specification morphism m' defined by

$$\begin{array}{rl} \forall s' \in \Sigma', m'(s') = & \text{if } s' \in S \text{ then } m(s') \text{ else } s' \\ \forall f' : s'_1, ..., s'_n \mapsto s' \in \Sigma', m'(f') = & \text{if } f' \in \Sigma \\ & \text{then } m(f') : m(s'_1), ..., m(s'_n) \mapsto m(s') \\ & \text{else } f' : m'(s'_1), ..., m'(s'_n) \mapsto m'(s'). \end{array}$$

- The *instantiated specification* $SP'_1 = SP_1 + (m'(\Sigma' - \Sigma), m'^*(E' - E))$.
- The specification morphisms $p : SP \mapsto SP'$ and $p_1 : SP_1 \mapsto SP'_1$ which are inclusions.

Notation: $\mathcal{V}_m$, $\mathcal{V}_{m'}$, $\mathcal{V}_p$ and $\mathcal{V}_{p_1}$ denote the forgetful functors [EM85] respectively associated to specification morphisms m, m' and inclusions p, p_1.

Paramater passing is usually represented by a parameter passing diagram.

$$\begin{tikzcd} SP \arrow[r, "p"] \arrow[d, "m"'] & SP' \arrow[d, "m'"] \\ SP_1 \arrow[r, "p"] & SP'_1 \end{tikzcd} \qquad \begin{tikzcd} ALG(SP) \arrow[r, shift left, "\mathcal{F}"] & ALG(SP') \arrow[l, shift left, "\mathcal{V}_p"] \\ ALG(SP_1) \arrow[u, "\mathcal{V}_m"] & ALG(SP'_1) \arrow[u, "\mathcal{V}_{m'}"'] \arrow[l, "\mathcal{V}_{p_1}"'] \end{tikzcd}$$

The unicity of m' and p_1 comes from the fact that a parameter passing diagram is a pushout in the category of specifications with their morphisms. This property also implies that the composition of two such diagrams is again a parameter passing diagram and this composition is associative.

Example 23.8 Let us consider the parameterized specification of Example 23.7 and the actual parameter NAT:

$$\begin{array}{rcl} sort & Nat & \\ 0: & \mapsto & Nat \\ s: Nat & \mapsto & Nat \\ +: Nat, Nat & \mapsto & Nat \\ \forall n: Nat,\ n+0 & \to & n \\ \forall n, m: Nat,\ n+s(m) & \to & s(n+m). \end{array}$$

Let m be the specification morphism defined by:

$$m(Elem) = Nat, m(id) = 0, m(\pi) = +.$$

Instantiation of MONOID by NAT needs to prove that the equalities

$$\begin{array}{rcl} \forall n: Nat,\ n+0 & = & n \\ \forall n: Nat,\ 0+n & = & n \\ \forall n_1, n_2, n_3: Nat,\ ((n_1+n_2)+n_3 & = & n_1+(n_2+n_3)). \end{array}$$

hold in the initial algebra of NAT.

The instantiated specification is:

$$\begin{array}{rcl} sort & Nat & \\ sort & List & \\ 0: & \mapsto & Nat \\ s: Nat & \mapsto & Nat \\ +: Nat, Nat & \mapsto & Nat \\ cons: Nat, List & \mapsto & List \\ nil: & \mapsto & List \\ append: List, List & \mapsto & List \\ prod: List & \mapsto & Nat \\ \forall n: Nat,\ n+0 & \to & n \\ \forall n, m: Nat,\ n+s(m) & \to & s(n+m) \\ \forall l: List,\ append(nil, l) & \to & l \\ \forall n: Nat, l, l': List,\ append(cons(n,l), l') & \to & cons(n, append(l, l')) \\ \forall l: List,\ prod(nil) & \to & 0 \\ \forall n: Nat, l: List,\ prod(cons(n,l)) & \to & n + prod(l)). \end{array}$$

The question is now: is the syntactic construction of SP'_1 compatible with the semantics respectively chosen for PSP, SP_1 and SP'_1? Here the semantics given to specifications SP_1 and SP'_1 are the initial algebras of these specifications denoted respectively by $\mathcal{T}_{SP_1}$ and $\mathcal{T}_{SP'_1}$.[1] The answer is yes, provided *correctness*.

[1] It is implicitely assumed that SP_1 and SP'_1 have no empty sorts

Definition 23.12 The parameter passing is *correct* for a parameterized specification PSP and a specification morphism m from SP to SP_1 if

1. $\mathcal{V}_{p_1}(\mathcal{T}_{SP'_1}) = \mathcal{T}_{SP_1}$, property called *protection of actual parameter*,
2. $\mathcal{V}_{m'}(\mathcal{T}_{SP'_1}) = \mathcal{F} \circ \mathcal{V}_m(\mathcal{T}_{SP_1})$ property called *compatibility of parameter passing.*

The first property expresses that SP'_1 is a protected enrichment of SP_1. The second property expresses the fact that the semantics $\mathcal{F}$ of PSP agrees with the semantics of the instantiated specification SP'_1.

This definition of correctness is relative to one specification morphism. In order to get a notion of correctness that holds for any specification morphism, a stronger property on functors is needed.

23.3.4 Persistency

The correctness of parameter passing for every specification morphism requires that the functor $\mathcal{F}$ be persistent. Intuitively, persistency means that for any SP-algebra $\mathcal{A} \in ALG(SP)$, $\mathcal{A}$ is protected in $\mathcal{F}(\mathcal{A})$.

Definition 23.13 *[EM85] Given a parameterized specification PSP and the forgetful functor $\mathcal{V}_p : ALG(SP') \mapsto ALG(SP)$, the free functor $\mathcal{F} : ALG(SP) \mapsto ALG(SP')$ is said* persistent *if $\mathcal{V}_p \circ \mathcal{F} = \mathcal{I}$, where $\mathcal{I}$ is the identity functor on $ALG(SP)$, up to a natural isomorphism.*

The parameterized specification PSP is also said persistent.

Proposition 23.3 *[EM85] Given a parameterized specification PSP with a persistent functor $\mathcal{F} : ALG(SP) \mapsto ALG(SP')$ and a specification morphism m from SP to SP_1, there exists a persistent functor $\mathcal{F}_1 : ALG(SP_1) \mapsto ALG(SP'_1)$, called* extension *of $\mathcal{F}$ according to m. Moreover $\mathcal{F}_1$ is uniquely defined by*

$$\mathcal{V}_{m'} \circ \mathcal{F}_1 = \mathcal{F} \circ \mathcal{V}_m \text{ and } \mathcal{V}_{p_1} \circ \mathcal{F}_1 = \mathcal{I}_1$$

where $\mathcal{I}_1$ is the identity functor on $ALG(SP_1)$.

If $\mathcal{F}$ is persistent, then for any specification morphism m, the functor $\mathcal{F}_1$ exists and is persistent. This is exactly what is needed for correctness of parameter passing, for each specification morphism.

Theorem 23.4 *[EM85] Given a parameterized specification PSP, the parameter passing is correct for PSP and any specification morphism m iff PSP is persistent in $ALG(SP)$.*

Proof: With the same notations as before:

- If PSP is persistent, $\mathcal{F}$ is persistent and $\mathcal{F}_1$ is uniquely defined by $\mathcal{V}_{p_1} \circ \mathcal{F}_1 = \mathcal{I}_1$ and $\mathcal{V}_{m'} \circ \mathcal{F}_1 = \mathcal{F} \circ \mathcal{V}_m$, according to Proposition 23.3. The two properties defining correctness are just obtained by applying them to the initial object of $ALG(SP_1)$, namely $\mathcal{T}_{SP_1}$.
- Conversely, assume that parameter passing is correct and let us prove that $\forall \mathcal{A} \in ALG(SP), \mathcal{V}_p \circ \mathcal{F}(\mathcal{A}) = \mathcal{A}$. Let us choose the special morphism $m_{\mathcal{A}} : SP \mapsto SP(\mathcal{A}) = SP + (\emptyset, Const(\mathcal{A}), Eqns(\mathcal{A}))$. It can be proved that $\mathcal{V}_{m_{\mathcal{A}}}(\mathcal{T}_{SP(\mathcal{A})}) = \mathcal{A}$ and thus $\mathcal{V}_p \circ \mathcal{F}(\mathcal{A}) = \mathcal{V}_p \circ \mathcal{F} \circ \mathcal{V}_{m_{\mathcal{A}}}(\mathcal{T}_{SP(\mathcal{A})}) = \mathcal{V}_p \circ \mathcal{V}_{m'_{\mathcal{A}}}(\mathcal{T}_{SP'(\mathcal{A})}) = \mathcal{V}_{m_{\mathcal{A}}} \circ \mathcal{V}_{p_1}(\mathcal{T}_{SP'(\mathcal{A})}) = \mathcal{V}_{m_{\mathcal{A}}}(\mathcal{T}_{SP(\mathcal{A})}) = \mathcal{A}$.

□

Example 23.9 A example of a non-persistent parameterized specification [EM85] is given by $PSP = (SP, SP')$ where $SP = \{\{s\}, \emptyset, \emptyset\}$ and $SP' = \{\{s\}, \{e\}, \emptyset\}$.

Let NAT be the usual specification of natural numbers,
NAT $= \{\{Nat\}, \{0 :\mapsto Nat, succ : Nat \mapsto Nat\}, \emptyset\}$,
consider now the actual parameter
$SP_1 =$ NAT $+\{succ(succ(x)) = x\}$,
and the specification morphism m defined by $m(s) = Nat$.

Then the respective domains of sort Nat of $\mathcal{T}_{SP_1}$, $\mathcal{F} \circ \mathcal{V}_m(\mathcal{T}_{SP_1})$, and $\mathcal{T}_{SP'_1}$ are respectively $\{0, succ(0)\}$, $\{0, succ(0), e\}$ and $\{0, succ(0), e, succ(e)\}$. Neither the compatibility of parameter passing nor the protection of actual parameter are satisfied.

23.3.5 Generic algebra

We now consider different questions: how to prove correctness of parameter passing, for any specification morphism m? How to prove a generic assertion, that is, how to prove that for any specification morphism m, the translated assertion (using m) is valid in the initial algebra of the instantiated specification? Both questions have answers that need the introduction of a generic algebra.

Let $SP = (\Sigma, E)$ be a specification and $\mathcal{X}$ a denumerable set of variables whose sorts are in Σ. Let $\mathcal{T}_{SP}(\mathcal{X})$ be the initial term algebra associated to the specification $(\Sigma \cup \mathcal{X}, E)$. $\mathcal{T}_{SP}(\mathcal{X})$ is a Σ-algebra whose carrier is the quotient $\mathcal{T}(\Sigma \cup \mathcal{X})/E$ of the set of terms $\mathcal{T}(\Sigma \cup \mathcal{X})$ by the congruence $\overset{*}{\longleftrightarrow}_E$. Note that if $\mathcal{X}$ is any set of variables with sorts in Σ, $\mathcal{T}_{SP}(\mathcal{X})$ is the free SP-algebra generated by $\mathcal{X}$. $\mathcal{T}_{SP}(\emptyset)$ is the initial SP-algebra. The *PSP-generic algebra* is obtained for a third choice of $\mathcal{X}$:[2]

Definition 23.14 Let $PSP = (SP, SP')$ be a parameterized specification and X_{SP} a set of variables of parameter sorts. The *PSP-generic algebra* is the Σ'-algebra $\mathcal{T}_{SP'}(X_{SP})$, whose carrier $\mathcal{T}(\Sigma' \cup X_{SP})/E'$ is the quotient by $\overset{*}{\longleftrightarrow}_{E'}$ of the set $\mathcal{T}(\Sigma' \cup X_{SP})$ of *PSP-generic terms*.

23.3.6 Generic theory of a parameterized specification

The set of theorems valid in the class of algebras $\mathcal{F}(ALG(SP))$ associated to a parameterized specification, defines the *generic theory* of the parameterized specification.

Definition 23.15 *The* generic theory *of the parameterized specification PSP denoted $Th(PSP)$ is the set*

$$\{(\forall X, t = t') | \forall A \in ALG(SP), \mathcal{F}(A) \models (\forall X, t = t')\}.$$

The generic theory is also called equational theory of PSP in [Pad88]. Note that in the degenerated case where SP is the empty specification, then $Th(PSP)$ is the inductive theory of SP'.

Definition 23.16 Any substitution $\sigma : X \mapsto \mathcal{T}(\Sigma' \cup X_{SP})$, is called a *$PSP$-generic substitution*.

As a consequence of previous definitions, an equality $(\forall X, t = t')$ holds in the PSP-generic algebra $\mathcal{T}_{SP'}(X_{SP})$, which is denoted by $\mathcal{T}_{SP'}(X_{SP}) \models (\forall X, t = t')$, if for any PSP-generic substitution $\sigma : X \mapsto \mathcal{T}(\Sigma' \cup X_{SP})$, $\sigma(t) \overset{*}{\longleftrightarrow}_E \sigma(t')$.

Example 23.10 Let us consider again the parameterized specification of Example 23.7.

The term $append(cons(e, nil), nil)$ with e a variable of sort $Elem$, is a PSP-generic term.

Given a variable $l : List$, substitutions $(l \mapsto nil)$, $(l \mapsto cons(e, nil))$, $(l \mapsto append(cons(e, nil), nil))$ are PSP-generic substitutions.

The equality $(\forall e : Elem, l : List,\ append(cons(e, nil), l) = cons(e, l))$ holds in the PSP-generic algebra, just because it holds in the whole class of SP'-algebras.

The next theorem states that the generic theory is exactly the set of theorems valid in the PSP-generic algebra.

Theorem 23.5 *[Pad88] Let $PSP = (SP, SP')$ be a parameterized specification, X_{SP} a set of variables of parameter sorts and $\mathcal{T}_{SP'}(X_{SP})$ the PSP-generic algebra.*

$$\mathcal{T}_{SP'}(X_{SP}) \models (\forall X, t = t') \text{ iff } (\forall X, t = t') \in Th(PSP).$$

Proof: (Sketch) If $(\forall X, t = t') \in Th(PSP)$, then it holds in $\mathcal{F}(\mathcal{T}_{SP}(X_{SP}))$ that is isomorphic to $\mathcal{T}_{SP'}(X_{SP})$.

Conversely, assume that $\mathcal{T}_{SP'}(X_{SP}) \models (\forall X, t = t')$. To prove that the equality holds in $\mathcal{F}(\mathcal{A})$ for any $\mathcal{A}$, let us prove that it holds in $\mathcal{T}_{SP'}(\mathcal{A})$ which is isomorphic to $\mathcal{F}(\mathcal{A})$. Any assignment $\sigma : X \mapsto \mathcal{T}_{SP'}(\mathcal{A})$ can be decomposed into $\mu : X \mapsto \mathcal{T}_{SP'}(X_{SP})$ and $\alpha : \mathcal{T}_{SP'}(X_{SP}) \mapsto \mathcal{T}_{SP'}(\mathcal{A})$. Then from $\mu(t) = \mu(t')$, it is easily deduced that $\sigma(t) = \alpha(\mu(t)) = \alpha(\mu(t')) = \sigma(t')$. □

The following result explains in which sense a theorem in $Th(PSP)$ is generic: validity of an equality in the PSP-generic algebra means validity of the translated equality in any instantiation of the parameterized specification, provided that the parameterized specification is persistent. A similar result is given in [DJvP89].

Theorem 23.6 *Let $PSP = (SP, SP')$ be a persistent parameterized specification and X_{SP} a set of variables of parameter sorts. Then the two following properties are equivalent:*

[2]The word *generic* is currently used for free algebras [Tay79] but less often in the context of parameterization [DJvP89].

1. $\mathcal{T}_{SP'}(X_{SP}) \models (\forall X, t = t')$

2. *for any specification morphism m, $\mathcal{T}_{SP'_1} \models (\forall m'^*(X), m'^*(t) = m'^*(t'))$.*

Proof: If $\mathcal{T}_{SP'}(X_{SP}) \models (\forall X, t = t')$, then the equality holds in $\mathcal{F}(\mathcal{A})$ for any $\mathcal{A}$, for instance in $\mathcal{F} \circ \mathcal{V}_m(\mathcal{T}_{SP_1})$ for any specification morphism m. Assuming persistency, the compatibility of parameter passing for m yields $\mathcal{F} \circ \mathcal{V}_m(\mathcal{T}_{SP_1}) = \mathcal{V}_{m'}(\mathcal{T}_{SP'_1})$. Then $\mathcal{V}_{m'}(\mathcal{T}_{SP'_1}) \models (\forall X, t = t')$, so $\mathcal{T}_{SP'_1} \models (\forall m'^*(X), m'^*(t) = m'^*(t'))$.

Conversely, for any $\mathcal{A} \in ALG(SP)$, there exists a specification morphism $m_{\mathcal{A}}$ and a specification $SP(\mathcal{A})$ such that $\mathcal{A} = \mathcal{V}_{m_{\mathcal{A}}}(\mathcal{T}_{SP(\mathcal{A})})$. Since $\mathcal{T}_{SP'(\mathcal{A})} \models (\forall m'^*_{\mathcal{A}}(X), m'^*_{\mathcal{A}}(t) = m'^*_{\mathcal{A}}(t'))$ implies $\mathcal{V}_{m'_{\mathcal{A}}}(\mathcal{T}_{(SP'(\mathcal{A}))}) \models (\forall X, t = t')$, then $\mathcal{F} \circ \mathcal{V}_{m_{\mathcal{A}}}(\mathcal{T}_{SP(\mathcal{A})}) \models (\forall X, t = t')$ and $\mathcal{F}(\mathcal{A}) \models (\forall X, t = t')$. By Theorem 23.5, $\mathcal{T}_{SP'}(X_{SP}) \models (\forall X, t = t')$. $\Box$

Actually only the compatibility of parameter passing is used in this proof. But this property is equivalent to persistency, as shown in [Ore87].

The next theorem relates the notion of persistency with proof theoretical properties.

Theorem 23.7 *[Gan83] Let $PSP = (SP, SP')$ be a parameterized specification and X_{SP} a set of variables of parameter sorts. PSP is persistent iff the following two properties are satisfied:*

1. *$PSP = (SP, SP')$ is* generic sufficiently complete*, i.e.: $\forall t \in \mathcal{T}(\Sigma' \cup X_{SP})$, t of parameter sort, $\exists t_0 \in \mathcal{T}(\Sigma \cup X_{SP})$ such that $t \overset{*}{\longleftrightarrow}_{E'} t_0$.*

2. *$PSP = (SP, SP')$ is* generic consistent*, i.e.: $\forall t, t' \in \mathcal{T}(\Sigma \cup X_{SP})$ of parameter sorts, $t \overset{*}{\longleftrightarrow}_{E'} t'$ iff $t \overset{*}{\longleftrightarrow}_{E} t'$.*

This definition expresses in other words that the enrichment $(\Sigma \cup X_{SP}, E) \subseteq (\Sigma' \cup X_{SP}, E')$ is protected.

In order to go further and design effective tools for parameterized proofs, we now focus on equational theories described by rewrite systems.

Persistency is a very strong property that sometimes one may want to drop. So a first question that arises is the following: which results remain true if persistency is not assumed? Actually persistency is assumed in Theorem 23.6, and only the compatibility of parameter passing is used in its proof. Theorem 23.6 could be weakened as follows: if $\mathcal{T}_{SP'}(X_{SP}) \models (\forall X, t = t')$, then for any specification morphism m satisfying the compatibility of parameter passing, $\mathcal{T}_{SP'_1} \models (\forall m'^*(X), m'^*(t) = m'^*(t'))$. However the problem of checking the compatibility of parameter passing, even for a specific specification morphism, is not solved. Moreover assuming compatibility of parameter passing for every specification morphism has been proved equivalent to persistency in [Ore87].

23.3.7 Generic ground reducibility

In order to check persistency of a parameterized specification and validity in the PSP-generic algebra, the notion of PSP-generic ground reducibility is needed.

Let $PSP = (SP, SP')$ be a parameterized specification with $SP = (\Sigma, R)$ and $SP' = (\Sigma', R')$ where R and R' are rewrite systems. Let X_{SP} be an infinite set of variables of parameter sorts.

Definition 23.17 *Given a rewrite system R', terminating on $\mathcal{T}(\Sigma' \cup X_{SP})$, a term $t \in \mathcal{T}(\Sigma' \cup X)$ is PSP-*generic ground reducible *with R' if for any PSP-generic substitution $\sigma : X \mapsto \mathcal{T}(\Sigma' \cup X_{SP})$, $\sigma(t)$ is reducible using R'.*

*An equality $(\forall X, t = t')$ is PSP-*generic ground reducible *with R' if for any PSP-generic substitution $\sigma : X \mapsto \mathcal{T}(\Sigma' \cup X_{SP})$, such that $\sigma(t) \neq \sigma(t')$, either $\sigma(t)$ or $\sigma(t')$ is reducible using R'.*

Algorithms for checking ground reducibility can be extended to check PSP-generic ground reducibility. Here, the test for ground reducibility in the case of left-linear rules given in [JK86b] is generalized to a test for generic ground reducibility. The goal is to exhibit a finite set of substitutions $\mathcal{S}$ such that a term is generic ground reducible iff all its instances by substitutions in $\mathcal{S}$ are reducible.

Given a set of rewrite rules R' of depth d, let

$$\mathcal{S}(R') = \{\ top(t_0, d) \mid t_0 \text{ is an } R'\text{-irreducible } PSP\text{-generic term}\ \}$$

be called the *PSP-generic test set.* For practical reasons, variables in terms of $\mathcal{S}(R')$ are assumed distinct, which is always possible by renaming them: $\forall t, t' \in \mathcal{S}(R')$, $\mathcal{V}(t) \cap \mathcal{V}(t') = \emptyset$. The set $\mathcal{S}(R')$ is computed as the limit of a stationary sequence of sets $\mathcal{S}_i$ defined as follows:

$$\mathcal{S}_i = \{top(t_0, d) \mid t_0 \text{ is an } R'\text{-irreducible } PSP\text{-generic term such that } depth(t_0) \leq i\}.$$

Then $\mathcal{S}(R') = S_k$ as soon as $S_k = S_{k+1}$ for some k.

Theorem 23.8 *A term t is PSP-generic ground reducible by a left-linear rewrite system R' iff all its instances*

$$\{\sigma(t) \mid \sigma : \mathcal{V}(t) \mapsto \mathcal{S}(R')\},$$

obtained by substituting variables of t by terms in $\mathcal{S}(R')$, are reducible by R'.

Proof: The proof is similar to the proof in [JK89] and of Theorem 22.1 of Chapter 22.

- Assume that all instances of t obtained by substituting variables of t by terms in $\mathcal{S}(R')$, are reducible by R'. For any PSP-generic substitution σ', if σ' is not R'-normalized, then $\sigma'(t)$ is R'-reducible. Otherwise, let us define for any variable x of t, $\sigma(x) = top(\sigma'(x), d)$. Then $\sigma \in \mathcal{S}(R')$ and $\sigma(t)$ is R'-reducible by hypothesis. Since σ' is an instance of σ, $\sigma'(t)$ is also R'-reducible.
- Assume now that t is PSP-generic ground reducible. Given $\sigma \in \mathcal{S}(R')$, let us define for any variable x_i of t, $t_i = \sigma(x_i)$. Then there exists t'_i an R'-irreducible instance of t_i such that $t_i = top(t'_i, d)$. Finally let us define σ' such that $\sigma'(x_i) = t'_i$. Since t is PSP-generic ground reducible, $\sigma'(t)$ is R'-reducible by a rewrite rule $l \to r$, at some non-variable position ω in t because σ' is normalized. But for any position υ in l, the top symbols of $l_{|\upsilon}$, $\sigma(t)_{|\omega.\upsilon}$ and $\sigma'(t)_{|\omega.\upsilon}$ are the same, because of the definition of d and the fact that $t_i = top(t'_i, d)$. Now let W be the set of variable occurrences in l and for any variable y at occurrence $\upsilon \in W$ define $\sigma''(y) = \sigma(t)_{|\omega.\upsilon}$. Note that σ'' is well-defined because y has only one occurrence in l. So $\sigma(t)_{|\omega} = \sigma''(l)$ and so $\sigma(t)$ is R'-reducible.

□

Example 23.11 Consider the parameterized specification of Example 23.7. In order to check the PSP-generic ground reducibility of $prod(append(l, l'))$, the following generic test set needs to be considered:

$$\{nil, cons(e_0, nil), cons(e_1, cons(e_2, nil))\}.$$

where e_0, e_1, e_2 are new variables of sort $Elem$ in X_{SP}. For each deduced substitution α, $\alpha(t)$ is reducible using R'.

23.3.8 Proof of generic sufficient completeness

We now extend the proof of sufficient completeness for convergent rewrite systems of Kapur, Narendran and Zhang in [KNZ87]. Their result states the equivalence between sufficient completeness and a check for ground reducibility, provided that terms built on the imported signature are preserved.

Definition 23.18 Let $PSP = (SP, SP')$ be a parameterized specification such that $SP = (\Sigma, R)$ and $SP' = (\Sigma', R')$. R' *preserves parameters* if for any parameter term $t \in \mathcal{T}(\Sigma)$, its R'-normal form is a parameter term in $\mathcal{T}(\Sigma)$, for any term $t \in \mathcal{T}(\Sigma')_s$ of parameter sort $s \in \Sigma$, its R'-normal form is also of parameter sort.

Some sufficient conditions on the rewrite system R' can be proposed to guarantee preservation of parameters. For instance, the property will be satisfied as soon as the rewrite rules fulfill the following conditions: $\forall(l \to r) \in R'$, whenever l has a parameter sort, r has a parameter sort, and whenever l is a parameter term, r is a parameter term.

Proposition 23.4 Let $PSP = (SP, SP')$ be a parameterized specification such that $SP = (\Sigma, R) \subseteq SP' = (\Sigma', R')$, X_{SP} is a denumerable set of variables of parameter sorts, and R' is a convergent rewrite system on $\mathcal{T}(\Sigma' \cup X_{SP})$ preserving parameters.

Then the following propositions are equivalent:

1. $\forall f \in \Sigma' - \Sigma$, whose range is a sort $s \in \Sigma$, $f(x_1, ..., x_n)$ is PSP-generic ground reducible with R',
2. $\forall t' \in \mathcal{T}(\Sigma' \cup X_{SP})$ of sort $s \in \Sigma$, $\exists t \in \mathcal{T}(\Sigma \cup X_{SP})$ such that $t \overset{*}{\longleftrightarrow}_{R'} t'$.

Proof: The proof is an extension of [KNZ87] and similar to the proo of Proposition 23.1.

- Let us first prove that (1) implies (2).
Consider a term $t' \in \mathcal{T}(\Sigma' \cup X_{SP})$ of sort $s \in \Sigma$ and its R'-irreducible form t''. Since R' is preserves parameters, t'' has a parameter sort $s'' \in \Sigma$. Either $t'' \in \mathcal{T}(\Sigma \cup X_{SP})$ and then $t = t''$, or $t'' \notin \mathcal{T}(\Sigma \cup X_{SP})$. In this case, t'' contains a subterm $f(u_1, ..., u_n)$, with $u_1, ..., u_n \in \mathcal{T}(\Sigma' \cup X_{SP})$ and $f \in \Sigma' - \Sigma$, which moreover has a co-arity $s' \in \Sigma$. The subterm $u = f(u_1, ..., u_n)$ is indeed a PSP-generic instance of $f(x_1, .., x_n)$, so is reducible for R', which contradicts the hypothesis that t'' is R'-irreducible.
- Let us now prove that (2) implies (1).
If $f(x_1, .., x_n)$ is not PSP-generic ground reducible for R', there exists a PSP-generic substitution σ such that $t' = f(\sigma(x_1), .., \sigma(x_n))$ is irreducible for R'. Assume now that $\exists t \in \mathcal{T}(\Sigma \cup X_{SP})$ of sort $s \in \Sigma$, such that $t \overset{*}{\longleftrightarrow}_{R'} t'$. Since R' is convergent on $\mathcal{T}(\Sigma \cup X_{SP})$, this implies that t' is the R'-normal form of t. Which is impossible if R' preserves parameters.

□

23.3.9 Proof of generic consistency

In Sections 23.2.4 and 23.2.5, given an enrichment $SP = (\Sigma, R) \subseteq SP' = (\Sigma', E')$ with R a ground convergent rewrite system on $\mathcal{T}(\Sigma)$, the completion mechanism appeared as an interesting tool to both prove consistency of an enrichment and produce simultaneously a ground convergent rewrite system for the enriched specification. We now design a similar consistency proof procedure in the slightly more general framework of a parameterized specification PSP with $SP = (\Sigma, R) \subseteq SP' = (\Sigma', E')$ with R a convergent rewrite system on $\mathcal{T}(\Sigma \cup X_{SP})$. (This can be checked by completion). PSP is generic consistent if whenever an equality, whose left and right-hand sides both belong to $\mathcal{T}(\Sigma \cup X_{SP})$, is added, then this is a theorem valid in $ALG(SP)$, that is $t \overset{*}{\longleftrightarrow}_R t'$. In order to detect inconsistencies, we need the following definition:

Definition 23.19 Let us consider a parameterized specification PSP with $SP = (\Sigma, R) \subseteq SP' = (\Sigma', E')$ with R a convergent rewrite system. A set of equalities C is *provably inconsistent with* $Th(SP)$ if it contains an equality $(\forall X, t = t')$ such that t and t' are in $\mathcal{T}(\Sigma \cup X_{SP})$ and are not $\overset{*}{\longleftrightarrow}_R$-equivalent.

If SP is itself a parameterized specification, say $SP = (SP_0, SP_1)$, then $Th(SP)$ is its generic theory and the previous definition must be refined by replacing $\overset{*}{\longleftrightarrow}_R$-equivalence by validity in the SP_0-generic algebra.

In the completion process described below, it is convenient to split the set of equalities $E' - R$ into two parts: one, called C, which contains only parameter equalities (i.e. built on terms of $\mathcal{T}(\Sigma \cup X_{SP})$), and the other, called P, that contains non-parameter ones. $OCP(P \cup R, P)$ denote the set of ordered critical pairs [Bac87, BDP89] obtained by superposition of $P \cup R$ on P and conversely. There is no need to superpose rules in R with themselves. $CP(R, C)$ as usual denote the set of critical pairs obtained by superposition of R on C.

Let P be a set of equalities (quantified pairs of terms), C a set of conjectures (parameter equalities), R the underlying rewrite system on parameter terms, terminating and confluent on $\mathcal{T}(\Sigma \cup X_{SP})$, and $>$ a reduction ordering that contains R and can be extended to a reduction ordering on $\mathcal{T}(\Sigma' \cup X_{SP})$ total on E'-equivalence classes. The generic consistency proof procedure is expressed by the set $\mathcal{GC}$ of transition rules given in Figure 23.5, in which R is implicit:

In these transition rules, $\sqsupset$ denotes the strict encompassment ordering. Note that this set of transition rules is more general than the one for unfailing completion [Bac87], obtained as a subset when $C = \emptyset$, and the one for proof by consistency [Bac87], obtained for $P = \emptyset$. The *Deflation* transition rule is used first to split the set of equalities $E' - E$ added in the enrichment, into P and C such that C contains only parameter equalities and P_0 contains all other equalities.

Lemma 23.4 If $(P, C) \mapsto\!\!\!\!\twoheadrightarrow (P', C')$, then the congruences $\overset{*}{\longleftrightarrow}_{P \cup C \cup R}$ and $\overset{*}{\longleftrightarrow}_{P' \cup C' \cup R}$ coincide on $\mathcal{T}(\Sigma' \cup X_{SP})$.

Proof: by looking at the different transition rules. □

The considered set of proofs are proofs on PSP-generic terms using equalities in $P \cup C \cup R$. The goal is to transform such a proof $t \overset{*}{\longleftrightarrow}_{P \cup C \cup R} t'$ into a proof of the form

$$t \overset{*}{\longrightarrow}_{R \cup P>} t'' \overset{*}{\longleftarrow}_{R \cup P>} t'$$

or to find a provable inconsistency with $Th(SP)$.

The proof reduction relation that reflects the transition rule system $\mathcal{GC}$, is now defined.

Deduce	P, C
	$\Vdash\!\!\twoheadrightarrow$
	$P \cup \{p = q\}, C$
	if $(p, q) \in OCP(P \cup R, P)$
Deflation	$P \cup \{p = q\}, C$
	$\Vdash\!\!\twoheadrightarrow$
	$P, C \cup \{p = q\}$
	if $p\ and\ q\ parameter\ terms$
Delete	$P \cup \{p = p\}, C$
	$\Vdash\!\!\twoheadrightarrow$
	P, C
Collapse	$P \cup \{p = q\}, C$
	$\Vdash\!\!\twoheadrightarrow$
	$P \cup \{p' = q\}, C$
	if $(p \rightarrow^{l\rightarrow r}_{R\cup P>} p'$ with $p \sqsupset l)$ or $(p = l$ and $q > r)$
Conjecture	P, C
	$\Vdash\!\!\twoheadrightarrow$
	$P, C \cup \{p = q\}$
	if $(p, q) \in CP(R, C)$
Discharge	$P, C \cup \{p = q\}$
	$\Vdash\!\!\twoheadrightarrow$
	P, C
	if $(p = q) \in Th(SP)$
Simplify	$P, C \cup \{p = q\}$
	$\Vdash\!\!\twoheadrightarrow$
	$P, C \cup \{p' = q\}$
	if $p > p'$ & $p \overset{+}{\longleftrightarrow}_R p'$
Compose	$P, C \cup \{p = q\}$
	$\Vdash\!\!\twoheadrightarrow$
	$P, C \cup \{p' = q\}$
	if $p > p'$ & $p \longleftrightarrow^{g=d}_{C} p'$ & $p \sqsupset g > d$
Disproof	$P, C \cup \{p = q\}$
	$\Vdash\!\!\twoheadrightarrow$
	$Disproof$
	if $(p = q)$ $provably\ inconsistent\ with\ Th(SP)$

Figure 23.5: Generic consistency by completion

1. Deduce: $t' \leftarrow^{g=d}_{P>\cup R} t \rightarrow^{l=r}_{P>} t'' \Longrightarrow t' \longleftrightarrow^{p=q}_{P} t''$
2. Deflation: $t \longleftrightarrow^{p=q}_{P} t' \Longrightarrow t \longleftrightarrow^{p=q}_{C} t'$
3. Delete: $t \longleftrightarrow^{p=p}_{P} t' \Longrightarrow \Lambda$ where Λ is the empty proof.
4. Collapse: $t \longleftrightarrow^{p=q}_{P} t' \Longrightarrow t \rightarrow^{l\rightarrow r}_{P>\cup R} t'' \longleftrightarrow^{p'=q}_{P} t'$.
5. Conjecture: $t' \leftarrow^{l\rightarrow r}_{R} t \longleftrightarrow^{g=d}_{C} t'' \Longrightarrow t' \longleftrightarrow^{p=q}_{C} t''$
6. Discharge: $t \longleftrightarrow^{p=q}_{C} t' \Longrightarrow t \overset{*}{\longleftrightarrow}_R t'$
7. Simplify: $t \longleftrightarrow^{p=q}_{C} t' \Longrightarrow t \overset{+}{\longleftrightarrow}_R t'' \longleftrightarrow^{p'=q}_{C} t'$
8. Compose: $t \longleftrightarrow^{p=q}_{C} t' \Longrightarrow t \longleftrightarrow^{g=d}_{C} t'' \longleftrightarrow^{p'=q}_{C} t'$
9. Peak without overlap: $t' \leftarrow^{g=d}_{R\cup P>} t \rightarrow^{l=r}_{P>} t'' \Longrightarrow t' \rightarrow^{l=r}_{P>} t_1 \leftarrow^{g=d}_{R\cup P>} t''$
10. Peak with variable overlap: $t' \leftarrow^{g=d}_{R\cup P>} t \rightarrow^{l=r}_{P>} t'' \Longrightarrow t' \overset{*}{\longrightarrow}_{R\cup P>} t_1 \overset{*}{\longleftarrow}_{R\cup P>} t''$

Lemma 23.5 If $>$ is a reduction ordering that can be extended to a reduction ordering $>$ on $\mathcal{T}(\Sigma' \cup X_{SP})$ total on E'-equivalence classes, then the proof reduction relation is noetherian.

Proof: Let us define the complexity measure of elementary proof steps by:

$$\begin{array}{rcll}
c(s \longleftrightarrow_P^{g=d} t) & = & (2, s, g, t) & if \quad s > t \\
c(s \longleftrightarrow_P^{g=d} t) & = & (2, t, d, s) & if \quad t > s \\
c(s \longleftrightarrow_C^{g=d} t) & = & (1, s, g, t) & if \quad s > t \\
c(s \longleftrightarrow_C^{g=d} t) & = & (1, t, d, s) & if \quad t > s \\
c(s \rightarrow_R^{g \rightarrow d} t) & = & (0, s, g, t) &
\end{array}$$

where $s, t \in \mathcal{T}(\Sigma' \cup X_{SP})$. Let $> \cup \rhd_{sub}$ denote the union of the reduction ordering $>$ and the strict subterm ordering $\rhd_{sub}$. Complexities of elementary proof steps are compared using the lexicographic combination, denoted $>_{ec}$ of the standard ordering on natural numbers, $> \cup \rhd_{sub}$, $\sqsupset$, and $> \cup \rhd_{sub}$. The complexity of a non-elementary proof is the multiset of the complexities of elementary proof steps that it contains. Complexities of non-elementary proofs are compared using the multiset extension $>_c$ of $>_{ec}$. For any proof reduction rule $L \Longrightarrow R$ defining the proof reduction relation, $c(L) >_c c(R)$. □

Theorem 23.9 *Let us consider a parameterized specification PSP with $SP = (\Sigma, E) \subseteq SP' = (\Sigma', E')$ with R a convergent rewrite system on $\mathcal{T}(\Sigma \cup X_{SP})$ presenting E.*

Let $>$ be a reduction ordering that contains R and can be extended to a reduction ordering $>$ on $\mathcal{T}(\Sigma' \cup X_{SP})$, total on E'-equivalence classes, and such that no parameter term is greater than a non-parameter one.

Let $(P_0, C_0) = (E', \emptyset) \mapsto\!\!\!\rightarrow (P_1, C_1) \mapsto\!\!\!\rightarrow \ldots$ be a derivation using $\mathcal{GC}$, $P_* = \bigcup_{i \geq 0} P_i$, $C_* = \bigcup_{i \geq 0} C_i$, $P_\infty = \bigcup_{i \geq 0} \bigcap_{j > i} P_j$, $C_\infty = \bigcup_{i \geq 0} \bigcap_{j > i} C_j$. Assume that $OCP(P_\infty \cup R, P_\infty) \cup CP(R, C_\infty)$ is a subset of $P_* \cup C_*$.

- *If C_∞ is empty, $(P_\infty \cup R)$ is Church-Rosser with respect to $>$ on $\mathcal{T}(\Sigma' \cup X_{SP})$ and the parameterized specification is generic consistent.*
- *If a provable inconsistency with $Th(SP)$ has been detected, the parameterized specification is not generic consistent.*

Proof:

- Assume that C_∞ is empty. Let us prove by induction on $\Longrightarrow$ that for any $i \geq 0$, for any proof $t \overset{*}{\longleftrightarrow}_{P_i \cup C_i \cup R} t'$ where $t, t' \in \mathcal{T}(\Sigma' \cup X_{SP})$, there exists a proof

  $$t \overset{*}{\longrightarrow}_{R \cup P_\infty^{\geq}} t'' \overset{*}{\longleftarrow}_{R \cup P_\infty^{\geq}} t'.$$

  Since C_∞ is empty, this means that any C_i-equality step has been replaced by R-equality steps. So if the proof $t \overset{*}{\longleftrightarrow}_{P_i \cup C_i \cup R} t'$ is not already of the desired form, then either it contains a peak of $R \cup P_\infty^{>}$, or a non-persisting equality in P. In both cases, the proof is reducible by $\Longrightarrow$ into $t \overset{*}{\longleftrightarrow}_{P_j \cup C_j \cup R} t'$ and the induction hypothesis gives the result.

  Given two terms, t and t' of $\mathcal{T}(\Sigma \cup X_{SP})$ of parameter sorts, such that $t \overset{*}{\longleftrightarrow}_{P_0 \cup C_0 \cup R} t'$, there exists a proof

  $$t \overset{*}{\longrightarrow}_{R \cup P_\infty^{\geq}} t'' \overset{*}{\longleftarrow}_{R \cup P_\infty^{\geq}} t'.$$

  But since t and t' are parameter terms and greater then any other term occurring in the proof, all these terms must be parameter terms. So no equality in P_∞ can apply since P_∞ contains non-parameter terms. So we get $t \overset{*}{\longleftrightarrow}_R t'$.

- If a provable inconsistency with $Th(SP)$ has been detected, this means that there exist $g, d \in \mathcal{T}(\Sigma \cup X_{SP})$ such that $g \longleftrightarrow_{C_k} d$, for some k, but do not satisfy $g \overset{*}{\longleftrightarrow}_R d$. So g and d are $\overset{*}{\longleftrightarrow}_{P_0 \cup C_0 \cup R}$-equivalent but not $\overset{*}{\longleftrightarrow}_R$-equivalent, which proves that the parameterized specification is not generic consistent.

□

Example 23.12 The parameterized specification of Example 23.7 is persistent.

The first step is to prove that the enrichment $(\Sigma \cup X_{SP}, R) \subseteq (\Sigma' \cup X_{SP}, R')$ is generic consistent. For that, the consistency proof procedure is applied. Since there is no critical pair, the enrichment is obviously consistent.

Second, in order to prove that the enrichment $(\Sigma \cup X_{SP}, R) \subseteq (\Sigma' \cup X_{SP}, R')$ is generic sufficiently complete, we check that R' preserves parameters.

Then we have to check that $prod(l)$ is PSP-generic ground reducible with R'. The instantiations to be checked are:

$$\begin{array}{lll} (l & \mapsto & nil) \\ (l & \mapsto & cons(e_0, nil)) \\ (l & \mapsto & cons(e_1, cons(e_2, l))) \end{array}$$

where $e_0, e_1, e_2 \in X_{SP}$ and l is a variable of sort $List$. For these three instantiations σ, the term $\sigma(prod(l))$ is clearly reducible.

23.3.10 Proof of a generic theorem

In the context of proving and disproving theorems in a parameterized specification, we need a slightly different notion of provable inconsistency, that is directly inspired by the one used in [Bac88]. We assume in this section that PSP is a parameterized specification defined by the formal parameter $SP = (\Sigma, R)$ and $SP' = (\Sigma', R')$, where R and R' are rewrite systems.

Definition 23.20 Let us consider a parameterized specification PSP with $SP = (\Sigma, R) \subseteq SP' = (\Sigma', R')$, where R' is a terminating rewrite system and $>$ a reduction ordering that contains R'. A set of equalities C is *provably inconsistent with* $Th(PSP)$ if it contains an equality $(\forall X, t = t')$ which satisfies either $t > t'$ and t is not PSP-generic ground reducible, or $(\forall X, t = t')$ is not PSP-generic ground reducible.

Replacing the notion of provable inconsistency by the notion of provable inconsistency with $Th(PSP)$ allows applying the proof by consistency method to the proof of theorems in $Th(PSP)$.

Let C be a set of conjectures, R' the underlying rewrite system of the parameterized specification, terminating and confluent on $\mathcal{T}(\Sigma' \cup X_{SP})$, and $>$ a reduction ordering that contains R'. The generic proof procedure is expressed by the following set $\mathcal{GI}$ of transition rules, given in Figure 23.6. This is a subset of $\mathcal{GC}$, obtained with $P = \emptyset$.

Conjecture C
$\Vdash\!\!\!\twoheadrightarrow$
$C \cup \{p = q\}$
if $(p, q) \in CP(R', C)$

Discharge $C \cup \{p = q\}$
$\Vdash\!\!\!\twoheadrightarrow$
C
if $(p = q) \in Th(PSP)$

Simplify $C \cup \{p = q\}$
$\Vdash\!\!\!\twoheadrightarrow$
$C \cup \{p' = q\}$
if $p > p'$ & $p \overset{+}{\longleftrightarrow}_{R'} p'$

Compose $C \cup \{p = q\}$
$\Vdash\!\!\!\twoheadrightarrow$
$C \cup \{p' = q\}$
if $p \longleftrightarrow^{g=d}_{C} p'$ & $p \sqsupset g > d$

Disproof $C \cup (p = q)$
$\Vdash\!\!\!\twoheadrightarrow$
$Disproof$
if $(p = q)$ *provably inconsistent with* $Th(PSP)$

Figure 23.6: Generic proof by consistency rules

Theorem 23.10 *Let $PSP = (SP, SP')$ be a parameterized specification and X_{SP} a set of variables of parameter sorts. Let R' be a terminating and confluent rewrite system on $\mathcal{T}(\Sigma' \cup X_{SP})$ presenting E', C_0 be the set of PSP-generic conjectures to be proved, and $>$ be a reduction ordering that contains R'. Let $C_0 \Vdash\!\!\!\twoheadrightarrow C_1 \Vdash\!\!\!\twoheadrightarrow \ldots.$ be a derivation using $\mathcal{GI}$ such that $CP(R', C_\infty)$ is a subset of C_*. If no provable inconsistency with $Th(PSP)$ has been detected, then C_0 is included in $Th(PSP)$.*

Proof: It is a consequence of the proof by consistency method in $\mathcal{T}_{SP'}(X_{SP})$ [Bac87, Bac88]. □

Example 23.13 In the parameterized specification of Example 23.7, let us prove that the PSP-generic conjecture

$$\forall l, l' : List,\ prod(append(l, l')) = \pi(prod(l), prod(l'))$$

is valid.

This conjecture is PSP-generic ground reducible. By choosing a precedence such that $prod > \pi$, the left-hand side is bigger than the right-hand side. Superpositions on the bigger term are enough. So two superpositions have to be computed:

- With the rule $\forall l : List,\ append(nil, l) \rightarrow l$, we get

$$prod(l') = \pi(prod(nil), prod(l')).$$

The term $\pi(prod(nil), prod(l'))$ reduces to $prod(l')$ and the conjecture becomes a trivial equality.
- With the rule $\forall e : Elem, l, l' : List,\ append(cons(e, l), l') \rightarrow cons(e, append(l, l'))$, we get

$$prod(cons(e, append(l_1, l_2))) = \pi(prod(cons(e, l_1)), prod(l_2)).$$

After reduction on both sides, we get

$$\pi(e, prod(append(l_1, l_2))) = \pi(e, \pi(prod(l_1), prod(l_2)))$$

that can be simplified using the initial conjecture.

23.4 Conclusion

Very few programming environments currently benefit from automated theorem provers, even in the case of rewrite programs that do have adequate proof tools. One obstacle to their use is often a matter of size of programs and this is why hierarchical and parameterized programs are so important.

Proofs in parameterized specifications have been experimented in several systems like CEC [Gan87] or Reveur4 [RZ84]. Now several research directions can be outlined.

- This chapter focussed on equational parameterized specifications. In the case of conditional specifications, some results can be extended along the lines of Navarro and Orejas [NO87], but an adequate notion of persistency needs the introduction of so-called LOG-algebras, that have an initial boolean domain, together with a property of persistency with respect to booleans. Note that Theorem 23.7 has been proved by Ganzinger [Gan83] in the case where the considered class of algebras is the class $ALG(SP)$ of all algebras satisfying SP, and by Orejas and Navarro [NO87] in the case where the class of algebras is restricted to LOG-algebras.

- We have only considered here the techniques of proofs by consistency. Everone agrees now that they are sometimes inefficient and more direct inductive proof methods have been developed, for instance by U.Reddy [Red90] in the equational case, or by E.Kounalis and M.Rusinowitch [KR90] in the Horn clause case. These methods are briefly described in Section 22.7 of Chapter 22. Extending these methods for proving theorems in parameterized specifications does not seem too difficult, and would lead to interesting applications.

Chapter 24

Gröbner bases

24.1 Introduction

The motivation of this chapter is to describe some applications of rewriting techniques to polynomial equations. In 1965, Buchberger introduced the notion of Gröbner bases and devised an algorithm to compute the Gröbner basis of a finite set of polynomials. Buchberger's algorithm can be roughly sketched as follows: Assume that all monomials are totally ordered with a well-founded ordering $<$ compatible with the product. Each polynomial equation $g =^? 0$ in E (for instance $2x - 6y^2 + 4z =^? 0$) is considered as a reduction rule that reduces the greatest monomial in the equation. For instance $2x - 6y^2 + 4z =^? 0$ is considered as $x \rightarrow 3y^2 - 2z$ if x is greater than y^2 and z. Now two reduction rules whose left-hand sides are not mutually prime are said to overlap. In this case the least common multiple of the two left-hand sides can be rewritten in two different ways, producing a critical pair (g, h). If further rewriting does not succeed in simplifying the critical pair to a same polynomial, the reduced critical pair (g', h') produces a new polynomial equation $g' - h' =^? 0$ added to E. By iterating the process, a confluent reduction system R is obtained and is called the Gröbner basis of E. The main result, due to Buchberger, states that a polynomial g belongs to I if and only if g reduces to 0 with R.

This algorithm has been widely used in computer algebra over the past few years. It provides algorithmic solutions to a variety of problems, such as exact solutions of a set of algebraic equations, computations in the residue class ring modulo an ideal, decision about various properties of an ideal, polynomial solution of linear homogeneous equation with polynomial coefficients, word problems modulo ideals and in commutative semigroups. Such examples are described in [Buc85]. It was also applied to propositional logic or geometrical theorem proving.

24.2 Polynomial ideal theory

Let $A = K[x_1, \ldots, x_n]$ by the polynomial ring on the field K with n variables $x_1, \ldots, x_n$, also called indeterminates. The function symbols $+$ and $\times$ respectively denote usual addition and multiplication of the ring. Most often the symbol $\times$ is omitted, for instance in the notation x^p used to denote the product of p elements equal to x.

Definition 24.1 A power product is a polynomial of the form $x_1^{i_1} \ldots x_n^{i_n}$. $x_1^0 \ldots x_n^0$ is denoted by 1. A *monomial* m is a product of a constant in K by a power product $x_1^{i_1} \ldots x_n^{i_n}$. Note that 0 is not a monomial. A *polynomial* p is a sum of monomials whose power products are pairwise distinct.

This definition of polynomials amounts to choose the usual distributive normal form representation of polynomials.

A finite set of polynomials determines a congruence, thanks to the notion of ideal.

Definition 24.2 Let $E \subseteq K[x_1, \ldots, x_n]$ be a finite set of polynomials. The *ideal* generated by E is defined by:

$$Ideal(E) = \{\sum_{i=1}^{m} h_i f_i \mid f_i \in E \text{ and } h_i \in K[x_1, \ldots, x_n]\}.$$

Two polynomials f, g are *congruent modulo* E, denoted by $f \equiv_E g$ if $f - g \in Ideal(E)$.

The goal is now to show the congruence modulo E is decidable. This will be done by proving its equivalence with the reflexive transitive closure of a reduction relation on polynomials. But this reduction relation needs to introduce first a suitable ordering on power products.

Definition 24.3 A total ordering $>$ on the set P of power products is *admissible* if

- for every $p \in P$, $p \neq 1$, $p > 1$.
- for any $p_1, p_2, p \in P$, $p_1 > p_2$ implies $p \times p_1 < p \times p_2$.

Two specific orderings are examples of admissible ordering, especially useful for polynomials.

1. The *pure lexicographic ordering* defined by:
$$x_1^{i_1} ... x_n^{i_n} > x_1^{j_1} ... x_n^{j_n}$$
if there exists $k \leq n$ such that $i_l = j_l$ for $l < k$ and $i_k > j_k$.
2. The *total degree ordering* defined by:
$$x_1^{i_1} ... x_n^{i_n} > x_1^{j_1} ... x_n^{j_n}$$
if $i = i_1 + ... + i_n > j = j_1 + ... + j_n$ or $i = j$ and there exists $k \leq n$ such that $i_l = j_l$ for $l < k$ and $i_k > j_k$.

Example 24.1 In the pure lexicographic ordering,
$$x^3 > x^2y^5 > x^2y^3z^2 > x^2y^3 > x^2yz^4.$$

Example 24.2 In the total degree ordering,
$$x^2y^5 > x^2y^3z^2 > x^2yz^4 > x^2y^3 > x^3.$$

Theorem 24.1 *Every admissible ordering $>$ is well-founded.*

Proof: Assume that there exists an infinite strictly decreasing sequence of power products $p_1 > p_2 > ... > p_k > ....$ To $p_i = x_1^{i_1} \dots x_n^{i_n}$ is associated the n-tuple $e_i = (i_1 \dots i_n)$. According to Dickson's lemma [Dic13], in the infinite sequence $e_1, \dots, e_k \dots$ of n-tuples of natural numbers, there exist indices i, j with $i < j$ such that $i_k \leq j_k$ for $k = 1, \dots n$. So there exists p such that $p_i \times p = p_j$.
If $p = 1$, $p_i = p_j$ which contradicts $p_i > p_j$.
If $p \neq 1$, then $p > 1$ since $>$ is admissible. For the same reason, $p_j = p_i \times p > p_i \times 1 = p_i$, which contradicts $p_i > p_j$. $\Box$

In the following, $>$ denotes an admissible ordering.
We need some additional definitions on polynomials.

Definition 24.4 Consider any non-zero polynomial $t = c_1p_1 + ... + c_kp_k$ where the c_i are non-zero coefficients in K and $p_1, ..., p_k$ are decreasing power products $p_1 > ... > p_k$.
$M(t) = c_1p_1, ..., c_kp_k$ denotes the set of monomials of t.
$P(t) = p_1, ..., p_k$ denotes the set of power products of t.
The monomial c_1p_1 is called the *leading monomial* in t and is denoted by $lm(t)$.
The power product p_1 is called the *leading power product* in t and is denoted by $lp(t)$.
$lc(t) = c_1$ is the *leading coefficient.*
Finally, $rm(t)$ denotes the polynomial $t - lm(t)$.

24.3 Polynomial reduction

A reduction relation on polynomials can be defined with a set of polynomials E.

Definition 24.5 A *polynomial reduction system* (PRS for short) is a pair $(E, >)$ composed of a finite set of polynomials E that does not contain 0 and an admissible ordering $>$. To any $f \in E$ corresponds the *polynomial reduction rule*
$$f^{>} : lm(f) \rightarrow -rm(f)$$

These polynomial reduction rules induce a relation $\rightarrow_E$ as follows:

Definition 24.6 Let E be a set of polynomials. A polynomial t reduces to t' if there exist a monomial $m \in M(t)$, a polynomial $f \in E$ (a reduction rule $lm(f) \rightarrow -rm(f)$), and a monomial m' s.t. $m = m' \times lm(f)$ and $t' = t - m - m' \times rm(f)$. This relation is denoted by $t \rightarrow_E t'$.

Example 24.3 Let $E = \{xy+x-1\}$ and $t = x^2y-y+x^2-1$. The polynomial t is reducible to $t' = x-y-1$.

Exercise: Consider $E = \{2x^2y - x^2 - 2,\ 3y^2 - xy + 3x\}$. Find all possible reduction sequences issued from the polynomial $6x^2y^2 - y^2$ using the PRS E.

By associating to any PRS E the abstract reduction system $\langle K[x_1, \dots, x_n], \rightarrow_E\rangle$, all notions of Chapter 4 carry over to polynomial reduction.

However it must be emphasized that one big difference with term rewriting is that polynomial reduction is not closed under contexts, i.e. the implication $t \rightarrow_E t'$ *implies* $t + u \rightarrow_E t' + u$ does not hold. This considerably complicates the theory.

Example 24.4 Consider $E = \{x^2 - y\}$ and the polynomials $t = 2x^2 + xy$, $t' = xy + 2y$ and $u = x^2 - xy$. Then $t \rightarrow_E t'$ but $t+u = 3x^2$ reduces to $3y$, and $t'+u = x^2+2y$. However one can notice that $t'+u$ reduces to $3y$.

This is the case in general since it can be proved that $t \rightarrow_E t'$ *implies* $t + u \downarrow_E t' + u$.

We do not prove here the next properties.

Proposition 24.1 Let E be a PRS.

- The relation $\xleftrightarrow{*}_E$ is closed under multiplication by monomial: if $t \xleftrightarrow{*}_E t'$ then $m \times t \xleftrightarrow{*}_E m \times t'$.
- The relation $\xleftrightarrow{*}_E$ is closed under context: if $t \xleftrightarrow{*}_E t'$ then $t + u \xleftrightarrow{*}_E t' + u$.
- The relations $\equiv_E$ and $\xleftrightarrow{*}_E$ coincide.

Proof: See for instance [MS92] □

As a corollary, we get that for two PRS E and E', E and E' define the same ideal iff $\xleftrightarrow{*}_E$ and $\xleftrightarrow{*}_{E'}$ are the same.

Another important difference with term rewriting is that the reduction always terminates. This is a consequence of Theorem 24.1. We just sketch the proof here.

Theorem 24.2 *Every PRS E is terminating.*

Proof: The admissible ordering on power products is extended to polynomials by multiset extension. It is then possible to prove that if $t \rightarrow_E t'$ then $t > t'$. The proof is obtained by contradiction. See for instance [MS92] for more details. □

Thus any polynomial has a normal form for the reduction rellation associated to a PRS E. However in general this normal form is not unique. Unicity is obtained if E is a Gröbner basis, i.e. if its reduction relation is confluent. Note that confluence is sufficient to get decidability of the congruence $\equiv_E$.

24.4 Gröbner bases

Several definitions of a Gröbner basis have been given in the literature.

Definition 24.7 A confluent PRS is called a *Gröbner basis.*

Theorem 24.3 *The following statements are equivalent:*

- *E is a Gröbner basis.*
- *Every polynomial t has a unique normal form for $\rightarrow_E$.*
- *Every polynomial $t \in Ideal(E)$ reduces to 0.*

- 0 *is the only normal form in* $Ideal(E)$.

Exercise: Prove the equivalences of Theorem 24.3.

Whenever an ideal I has a finite Gröbner basis, the last theorem provides a method for deciding whether a polynomial belongs to I.

But in general a given PRS E is not confluent and the problem is now to find a PRS E' which is a Gröbner basis and generates the same ideal. This is the result of a completion process described now.

The algorithm is based on the computation of overlappings between polynomials. Its efficiency is improved when reductions are performed as during the completion processes. However a big difference with completion is that the algorithm always terminates.

The first step is to define critical pairs and prove a lemma similar to Newman's Lemma.

Definition 24.8 Let $l_1 \to r_1$ and $l_2 \to r_2$ be two distinct polynomial reduction rules, with $l_1 = c_1p_1$ and $l_2 = c_2p_2$. The *lower product* $lcm(l_1, l_2)$ is the least common multiple of p_1 and p_2. So $lcm(l_1, l_2)$ is equal to l_1m_1 and to l_2m_2 for some monomials m_1 and m_2 and can be reduced to both r_1m_1 and to r_2m_2. The pair (r_1m_1, r_2m_2) is called the *critical pair of the two polynomial reduction rules.*

The critical pairs (r_1m_1, r_2m_2) originating from $l_1 \to r_1$ and $l_2 \to r_2$ and the critical pairs (r_2m_1, r_1m_1) originating from $l_2 \to r_2$ and $l_1 \to r_1$ are identified. So a PRS with n rules has $C_n^2 = n(n-1)/2$ critical pairs.

Definition 24.9 The critical pair (r_1m_1, r_2m_2) is *connected* if r_1m_1 and r_2m_2 are connected below $lcm(l_1, l_2)$, i.e. $r_1m_1 \downarrow_E r_2m_2$ and $lcm(l_1, l_2)$ is greater than every polynomial of this proof.

With these definitions, we get a result similar to Newman's Lemma (Theorem 16.1 of Chapter 16).

Theorem 24.4 *A PRS E is confluent (i.e. is a Gröbner basis) iff all its critical pairs are connected.*

Proof: See [MS92]. □

The algorithm for building a Gröbner basis from a PRS E can be informally described as follows:

1. Build all the critical pairs between all the polynomials of $E_0 = E$.
2. Reduce all critical pairs to their normal forms w.r.t. E_0. Add all the non-trivial normal forms to obtain a new set of polynomials E_1.
3. Repeat the same procedure for E_1 to obtain E_2 and so on.

This naïve process always terminates but its efficiency can be improved by interreducing polynomial rewrite rules during the process, as in Knuth-Bendix completion.

Definition 24.10 A PRS E is called *interreduced* if any $t \in E$ is a normal form w.r.t. $E - \{t\}$ and $lc(t) = 1$.

Proposition 24.2 Any PRS E can by transformed into an interreduced PRS $E!$ s.t. $\overset{*}{\longleftrightarrow}_E$ and $\overset{*}{\longleftrightarrow}_{E'}$ are the same and the sets of normal forms are the same.

Proof: See [MS92]. □

A polynomial completion algorithm is described in Figure 24.1. It takes as input argument a polynomial rewrite system `E` aand returns an irreducible Gröbner basis `G` with the same conversion as `E`. The simplification of polynomial reduction rules is performed by the procedure `SIMPLIF` and the computation of critical pairs of `G` is done by `CP(G)`. Critical pair criteria may improve this last step.

This algorithm is actually a simple version of Buchberger's algorithm described in [Buc85]. Especially the statement `P := CP(G)` is too coarse since many critical pairs of G are already known to be connected. The reader may find in [Buc85] the precise bookkeeping of unuseless and necessary critical computations.

Theorem 24.5 *The procedure* `GROBNER` *applied to E always terminates and computes a reduced Gröbner basis G s.t.* $\overset{*}{\longleftrightarrow}_E$ *and* $\overset{*}{\longleftrightarrow}_G$ *are the same.*

Proof: See [MS92]. □

```
PROCEDURE GROBNER (E)
G := SIMPLIF(E)
P := CP(G)
WHILE P is not empty
DO   choose a pair (p,q) in P;
     p':= G-normal form(p-q);
     CASE p' = 0  THEN  P := P - {(p,q)};
          ELSE G := SIMPLIF(G U {p'})
               P := CP(G)
     END CASE;
END DO
return G
END GROBNER
```

Figure 24.1: Buchberger's algorithm

Example 24.5 The procedure GROBNER applied to $G_0 = \{x^2y - y + x^2 - 1, xy + x - 1, xy^3 + y^3 + y + 2\}$ produces the following sets:

$$\begin{array}{lcl} G_1 & = & \{x - y - 1, x^2 - x + y, x - y^3 - y^2 - 3\} \\ G_2 & = & \{x - y - 1, y^3 + y^2 - y + 2, y^4 + 2y^3 + y^2 + 3y + 2\} \\ G_3 & = & \{x - y - 1, y^3 + y^2 - y + 2, -y^2 - 2y\} \\ G_4 & = & \{x + 1, y + 2\} \end{array}$$

The Gröbner basis obtained in G_4 allows for checking whether the polynomial

$$p = x^3y^5 - 4x^2y^3 + 5xy - 1$$

belongs to the ideal generated by G_0 just by reducing it to its normal form:

$$\begin{array}{l} x^3y^5 - 4x^2y^3 + 5xy - 1 \\ \to -x^2y^5 - 4x^2y^3 + 5xy - 1 \\ \to xy^5 - 4x^2y^3 + 5xy - 1 \\ \to -y^5 - 4x^2y^3 + 5xy - 1 \\ \to -4x^2y^3 + 5xy - y^5 - 1 \\ \to 4xy^3 + 5xy - y^5 - 1 \\ \to 5xy - y^5 - 4y^3 - 1 \\ \to 2y^4 - 4y^3 - 5y - 1 \\ \to -8y^3 - 5y - 1 \\ \to 16y^2 - 5y - 1 \\ \to -37y - 1 \to 73. \end{array}$$

Since $73 \neq 0$, p does not belong to the ideal.

Exercise: Prove that the Gröbner basis of the ideal generated by

$$\{xy^2 - y^2 - x + 1, xy^2 - y, x^3 - x^2 - x + 1\}$$

is $\{x^2 - 1, xy - y - x + 1\}$.

Exercise: One may wonder whether an interreduced PRS generating the same equivalence than a Gröbner basis is a Gröbner basis. This exercise provides a counterexample.

Consider the two PRS

$$E = \{x^2y \to x, \quad xy^2 \to x\}$$

and

$$G = \{x^2y \to x, \quad xy^2 \to x, \quad x^2 \to xy\}.$$

1. Prove that E is interreduced but not confluent.
2. Prove that G is a Gröbner basis not interreduced.
3. Prove that $\overset{*}{\longleftrightarrow}_E$ and $\overset{*}{\longleftrightarrow}_G$ are the same.

This section is concluded with a uniqueness result for intereduced Gröbner bases, similar to Theorem 16.4 of Chapter 16 for rewrite systems.

Theorem 24.6 *Let G_1 and G_2 be two interreduced Gröbner bases that generate the same equivalence relation ($\overset{*}{\longleftrightarrow}_{G_1} = \overset{*}{\longleftrightarrow}_{G_2}$). If they have the same underlying admissible ordering, then they are identical up to variable renaming.*

In other words, given an admissible ordering, the interreduced Gröbner basis of an ideal $Ideal(E)$ is unique.

24.5 Application to geometrical problems

Interesting applications of the Gröbner bases method are described in [CSY89]. In this section, a classical geometrical problem is solved by using Gröbner basis techniques.

Let $ABCD$ be a parallelogramme and E be the intersection of AC and DB. Prove that $AE = CE$, or in other words, the diagonals intersect in the middle.

The first point is to choose adequate coordinates. Here let $A = (0,0), B = (u_1, 0), C = (u_2, u_3), D = (x_2, x_1), E = (x_4, x_3)$. Proving that $AE = CE$ amounts proving that $g = 2u_2x_4 + 2u_3x_3 - u_3^2 - u_2^2 = 0$.

The hypothesis of the problems are translated as follows:

- AB and AC are parallel: $h_1 = u_1x_1 - u_1x_3 = 0$.
- DA and CB are parallel: $h_2 = u_3x_2 - (u_2 - u_1)x_1 = 0$.
- E belongs to BD: $h_3 = x_1x_4 - (x_2 - u_1)x_3 - u_1x_1 = 0$.
- E belongs to AC: $h_4 = u_3x_4 - u_2x_3 = 0$.

In general the hypotheses are expressed as a system S of polynomial equations:

$$\begin{array}{rcl} h_1(u_1, .., u_i, x_1, ..., x_j) & = & 0 \\ & ... & \\ h_n(u_1, .., u_i, x_1, ..., x_j) & = & 0 \end{array}$$

where the u_k are independent and the x_m dependent of u_k. The conclusion is also expressed as a polynomial equation $g = g(u_1, .., u_i, x_1, ..., x_j) = 0$. $h_1, ..., h_n$ and g are polynomials in $Q[u_1, .., u_i, x_1, ..., x_j]$.

One looks for the Gröbner basis of the ideal generated by S and proves that g reduces to 0.

By solving the two first equations, one gets $x_1 = u_3, x_2 = u_2 - u_1$. Solving the two last equations yields $x_3 = u_3/2, x_4 = u_2/2$. Then by replacing in g variables by their solutions, one gets $g = 0$. Problems arise with degenerated cases, for instance $u_1 = 0, u_3 = 0$ in which A, B, C are co-linear.

24.6 Comparison with completion modulo AC

Buchberger's algorithm computes the Gröbner basis for an ideal presented by a set of polynomial equations and always terminates. The completion procedure (modulo AC) computes a convergent rewrite systems for a set of first-order equalities. Both procedures are built around a concept of critical pairs computation. Despite these similarities, it is not possible to describe one procedure in terms of the other. On one hand, the term structure is more general than polynomial's one. On the other hand a first-order presentation of polynomials conflicts with the fact that fields may not be presented purely equationally.

So a first direction was an attempt to unify the two procedures and to propose a common ancestor of the two procedures: this is the approach followed by Winkler [Win89], who introduces rewriting modulo a simplification relation. The simplification relation takes into account the specific polynomials simplification that may hide the field specific operations.

A second approach, presented by Bündgen [Bün91], is to propose a convergent rewrite system whose initial model is isomorphic to $A = K[y_1, ..., y_n]$ in the case where K is a finite field of characteristic k. Using this set of rewrite rules and additional ground equalities specifying an ideal, the Buchberger's algorithm for polynomials over finite fields is simulated using completion modulo AC. In this way the Buchberger's algorithm can be considered as a special purpose completion with built-in AC-matching, AC-unification, polynomial normalization and specific critical pair transformation. Indeed this does not mean it is suitable to use this simulation for Gröbner basis computations but brings an explanation of the strong connections between the two procedures.

A third attempt, due to Middeldorp [MS92], showed that the construction of Gröbner bases can also be based on the completion modulo an equivalence relation as described in Chapter 18 and on Theorem 18.1 of Section 18.2. They observe that the lack of closure under context of the polynomial rewriting relation is source of annoyance in the development of the Gröbner bases theory. So they propose to abandon the distributive normal form representation and to write a polynomial as a finite sum of monomials instead. For instance, $x - x + y$, $2y - y$ and y are distinct polynomials but are equivalent because they have the same distributive normal form y. This method does not provide a competitive way to replace Buchberger's algorithm, but explains in yet another way its relation with completion.

24.7 Conclusion

The Gröbner basis approach has been recently be applied to the study of Boolean polynomials, built on the function symbols $\oplus$ and $\cap$ that represent exclusive disjunction and conjunction on booleans [SS88]. This technique is used to build constraint solvers for boolean constraints [SS90a], and constraints over sets [SSS90]. Since constraints are sets of polynomial equations, a Gröbner basis procedure allows checking satisfiability of such systems and also produces a simplified form of the constraints, namely the reduced Gröbner basis of the set of polynomial equations. Such constraint solvers are implemented in the constraint logic programming language CAL [ASS$^+$88].

Index

Bibliography

[Abd87] H. Abdulrab. *Résolution d'équations sur les Mots : étude et Implantation LISP de l'algorithme de Makanin.* Thèse de Doctorat d'Université, Université de Rouen (France), March 1987. III, 10.1

[AH90] Siva Anantharaman and J. Hsiang. An automated proof of the Moufang identities in alternative rings. *Journal of Automated Reasoning*, 6:79–109, 1990. 19.1

[AHM89] Siva Anantharaman, J. Hsiang, and J. Mzali. Sbreve2: A term rewriting laboratory with (AC-)unfailing completion. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 533–537. Springer-Verlag, April 1989. 7.1

[AM90] J. Avenhaus and K. Madlener. Term rewriting and equational reasoning. In R. B. Banerji, editor, *Formal Techniques in Artifial Intelligence*, pages 1–43. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1990. 1

[AN80] A. Arnold and M. Nivat. The metric space of infinite trees. Algebraic and topological properties. *Fundamenta Informaticae*, 3(4):181–205, 1980. 2.2.5

[AP90] H. Abdulrab and J.-P. Pécuchet. Solving word equations. *Journal of Symbolic Computation*, 8(5):499–522, 1990. 10.1

[ASS$^+$88] A. Aiba, K. Sakai, Y. Sato, D. Hawley, and R. Hasegawa. Constraint logic programming language cal. In ICOT, editor, *Proceedings of the International Conference on Fifth Generation Computer Systems*, volume 1, 1988. 24.7

[AT85] S. Arnborg and E. Tiden. Unification problems with one-sided distributivity. In *Proceedings 1st Conference on Rewriting Techniques and Applications, Dijon (France)*, volume 202 of *Lecture Notes in Computer Science*, pages 398–406, Dijon (France), May 1985. Springer-Verlag. 10.1

[B$^+$87] H. P. Barendregt et al. Term graph rewriting. In *Proc. of PARLE'87*, volume 259 of *Lecture Notes in Computer Science*. Springer-Verlag, 1987. 7.7

[Baa86a] Franz Baader. Unification in idempotent semigroups is of type zero. *Journal of Automated Reasoning*, 2(3):283–286, 1986. 10.2.3, 10.1

[Baa86b] Franz Baader. Unification in varieties of idempotent semigroups. Internal report, Institut für Mathematische Maschinen und Datenverarbeitung, Universität Erlangen, 1986. 10.1

[Baa87] Franz Baader. Unification in varieties of idempotent semigroups. *Semigroup Forum*, 36, 1987. 10.2.3

[Baa89a] Franz Baader. Characterizations of unification type zero. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 2–14. Springer-Verlag, April 1989. III, 10.2.2, 10.2.2

[Baa89b] Franz Baader. Unification in commutative theories. *Journal of Symbolic Computation*, 8(5):479–498, 1989. 10.1

[Baa91] F. Baader. Unification, Weak Unification, Upper Bound, Lower Bound, and Generalization Problems. In Book [Boo91], pages 86–97. 2.3.3, 10.1, 10.2.1

[Bac87] Leo Bachmair. *Proof methods for equational theories.* PhD thesis, University of Illinois, Urbana-Champaign, (Ill., USA), 1987. Revised version, August 1988. 5.1, 15.2, 16.3, 16.3, 16.3, 17.3, 17.1, 17.3, 17.5, 17.3, 18.1, 18.2.2, 18.3.2, 18.3.3, 18.6, 18.4.2, 22.6.1, 23.3.9, 23.3.9, 23.3.10

[Bac88] L. Bachmair. Proof by consistency in equational theories. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, pages 228–233, 1988. 22.1, 23.2.5, 23.3.10, 23.3.10

[Bac91] L. Bachmair. *Canonical equational proofs.* Computer Science Logic, Progress in Theoretical Computer Science. Birkhäuser Verlag AG, 1991. 20.5

[Bar84] H. Barendregt. *The Lambda-Calculus, its syntax and semantics.* Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition. 4.6.2

[Bar90] F. Barbarena. Adding algebraic rewriting to the calculus of constructions: strong normalization preserved. In S. Kaplan and M. Okada, editors, *Proceedings 2nd International Workshop on Conditional and Typed Rewriting Systems, Montreal (Canada)*, volume 516 of *Lecture Notes in Computer Science*, pages 262–271. Springer-Verlag, June 1990. 8.6

[BB88] Franz Baader and W. Büttner. Unification in Commutative Idempotent Monoids. *Theoretical Computer Science*, 56(1):345–352, 1988. 10.1

[BB89] W. Buntine and H.-J. Bürckert. On solving equations and disequations. Technical Report SR-89-03, Universität Kaiserslautern, 1989. 21.5

[BBH92] R. Barnett, D. Basin, and J. Hesketh. A recursion planning analysis of inductive completion. In *Proc. 2nd Symposium on Mathematics and Artificial Intelligence*, Annals of Mathematics and Artificial Intelligence, 1992. 22.1

[BCL87] A. Ben Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–160, October 1987. 6.3.2, 7.3.3

[BD86a] L. Bachmair and N. Dershowitz. Commutation, transformation and termination. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 5–20. Springer-Verlag, 1986. 7.3.3, 8.3, 8.1, 8.11

[BD86b] L. Bachmair and N. Dershowitz. Critical pair criteria for the Knuth-Bendix completion procedure. Technical report, University of Illinois at Urbana-Champaign, 1986. 19.3

[BD87] L. Bachmair and N. Dershowitz. Completion for rewriting modulo a congruence. In *Proceedings 2nd Conference on Rewriting Techniques and Applications, Bordeaux (France)*, volume 256 of *Lecture Notes in Computer Science*, pages 192–203, Bordeaux (France), May 1987. Springer-Verlag. 16.3

[BD88] L. Bachmair and N. Dershowitz. Critical pair criteria for completion. *Journal of Symbolic Computation*, 6:1–18, 1988. 19.3

[BD89a] L. Bachmair and N. Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, 67(2-3):173–202, October 1989. 21.1

[BD89b] L. Bachmair and N. Dershowitz. Equational inference, canonical proofs and proof orderings. Draft Notes, 1989. 15.3, 16.3, 16.3, 17.5, 19.3, 19.1, 19.2, 22.6.2

[BD94] Leo Bachmair and Nachum Dershowitz. Equational inference, canonical proofs, and proof orderings. *Journal of Association for Computing Machinery*, 41(2):236–276, 1994. 15.2

[BDJ78] D. Brand, J. A. Darringer, and W. H. Joyner. Completeness of conditional reductions. Technical Report RC 07404, IBM USA Research Center, Yorktown Heights, NY, 1978. 2.6.3

[BDP89] L. Bachmair, N. Dershowitz, and D. Plaisted. Completion without failure. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 1–30. Academic Press inc., 1989. 7.1, 7.1, 11.3, 17.1, 17.5, 23.3.9

[BES$^+$90] W. Büttner, K. Estenfeld, R. Schmid, H.-A. Schneider, and E. Tiden. Symbolic constraint handling through unification in finite algebras. *Applicable Algebra in Engineering, Communication and Computation*, 1(2):97–118, 1990. 13.2.1, 13.2.5

[BG89] H. Bertling and H. Ganzinger. Compile-time optimization of rewrite-time goal solving. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 45–58. Springer-Verlag, April 1989. 7.5

[BG91a] L. Bachmair and H. Ganzinger. Perfect model semantics for logic programs with equality. In K. Furukawa, editor, *Proc. 8th International Conference on Logic Programming*, Logic Programming Series, Research Reports and Notes, pages 645–659. The MIT press, July 1991. 20.5

[BG91b] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. Technical Report MPI-I-91-208, Max-Planck Institut für Informatik, Saarbrücken, 1991. 7.1, 7.5, 20.1, 20.3, 20.1, 20.2, 20.3, 20.3.4, 20.4, 20.4.2

[BG93] L. Bachmair and H. Ganzinger. Associative-Commutative superposition. Technical Report MPI-I-93-250, Max-Planck-Institut für Informatik, Saarbrücken, 1993. 19.2.4

[BGLS92] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In *Proceedings 11th International Conference on Automated Deduction, Saratoga Springs (N.Y., USA)*, pages 462–476, 1992. 21.1, 21.3, 21.1, 21.4

[BGM87] P. G. Bosco, E. Giovannetti, and G. Moiso. Refined strategies for semantic unification. In *Proceedings of TAPSOFT'87*, volume 150 of *Lecture Notes in Computer Science*, pages 276–290. Springer-Verlag, 1987. 14.2.5

[BH91] M. P. Bonacina and J. Hsiang. On Fairness of Completion-Based Theorem Proving Strategies. In Book [Boo91], pages 348–360. 15.3

[BH92] M. P. Bonacina and J. Hsiang. On rewrite programs: semantics and relationship with Prolog. *Journal of Logic Programming*, ??(??):??, 1992. 7.5.4

[BHSS90] H.-J. Bürckert, A. Herold, and M. Schmidt-Schauß. On equational theories, unification and (un)decidability. In Claude Kirchner, editor, *Unification*, pages 69–116. Academic Press inc., London, 1990. 2.4, 2.5, 2.4.8, 10.4, 11.2

[Bir35] G. Birkhoff. On the structure of abstract algebras. *Proceedings Cambridge Phil. Soc.*, 31:433–454, 1935. 2, 2.2, 2.3, 2.5

[Bir67] G. Birkhoff. *Lattice Theory*, volume 25. American Mathematical Society, Providence R.I., third edition, 1967. 3.1.3, 10.2.2

[BJSS88] A. Boudet, J.-P. Jouannaud, and M. Schmidt-Schauß. Unification in boolean rings and abelian groups. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, 1988. III

[BK86] J. A. Bergstra and J. W. Klop. Conditional rewrite rules: Confluency and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986. 2.6.3, 7.1, 7.5, 7.11, 20.1, 20.4, 20.2

[BK89] K. Bundgen and W. Küchlin. Computing ground reducibility and inductively complete positions. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, Lecture Notes in Computer Science, pages 59–75. Springer-Verlag, 1989. 22.1

[BKR92] Adel Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. Technical Report 1636, INRIA, 1992. 22.7

[BKW93a] A. Bockmayr, S. Krischer, and A. Werner. Narrowing strategies for arbitrary canonical rewriting systems. Technical Report No. 2011, INRIA-Lorraine, 1993. Also published as MPI-I-93-233 of MPI Saarbrücken, and Interner Bericht 22/93 of Universität Karlsruhe, and submitted to Fundamenta Informatica. 14.2.4

[BKW93b] A. Bockmayr, S. Krischer, and A. Werner. An optimal narrowing strategy for general canonical systems. In M. Rusinowitch and J.-L. Rémy, editors, *Proceedings of the 3rd International Workshop on Conditional Term Rewriting Systems (CTRS-92)*, volume 656 of *Lecture Notes in Computer Science*, pages 483–497. Springer-Verlag, 1993. 14.2.4

[BL80] G. Butler and D. S. Lankford. Experiments with computer implementations of procedures which often derive decision algorithms for the word problem in abstract algebras. Technical Report Memo MTP-7, Louisiana Tech. University, Mathematics Dept., Ruston LA, 1980. 5.5, 16.5

[BL81] A. Ballantyne and D. S. Lankford. New decision algorithms for finitely presented commutative semigroups. *Computers and Maths. with Appls.*, 7:159–165, 1981. 18.7

[BL86] M. Bellia and G. Levi. The relation between logic and functional languages. *Journal of Logic Programming*, 3:276–290, 1986. 14.2.5

[BM67] G. Birkhoff and S. MacLane. *Algebra*. The Macmillan Company, 1967. Translated in French by J. Weil [BM70]. 2, 10.2.2

[BM70] G. Birkhoff and S. MacLane. *Algèbre*, volume I et II of *Cahiers Scientifiques*. Gauthier–Villard, Paris, 1970. Traduit de l'anglais par J. Weil. 24.7

[BM84] J. A. Bergstra and J.-Ch. Meyer. On specifying sets of integers. *Elektronishe Informationsverarbeitung und Kybernetics*, 20(10 & 11):531–541, 1984. 2.6.5, 2.16

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and all That*. Cambridge University Press, 1998. 1

[Boc87] A. Bockmayr. A note on a canonical theory with undecidable unification and matching problem. *Journal of Automated Reasoning*, 3(1):379–381, 1987. 10.7

[Boc92] Alexander Bockmayr. Algebraic and logic aspects of unification. In Klaus U. Schulz, editor, *Word Equations and Related Topics, First International Workshop, IWWERT '90, Tübingen, Germany, October 1-3, 1990, Proceedings*, volume 572 of *Lecture Notes in Computer Science*, pages 171–180. Springer, 1992. 10.6

[Boc93] Alexander Bockmayr. Conditional narrowing modulo a set of equations. *Applicable Algebra in Engineering, Communication and Computation*, xx(xxx):pp, January 1993. 14.2.4

[Boo47] G. Boole. *The Mathematical Analysis of Logic*. Macmillan, New York, 1847. Reprinted: B. Blackwell, London, England, 1948. 13.2.1, 13.2.3, 13.2

[Boo91] Ronald V. Book, editor. *Rewriting Techniques and Applications, 4th International Conference, RTA-91*, LNCS 488, Como, Italy, April 10–12, 1991. Springer-Verlag. 24.7

[Bou70] N. Bourbaki. *Théorie des ensembles*. Eléments de Mathématique. Hermann, Paris, 1970. 10.2.2

[Bou89] A. Boudet. A new combination technique for ac unification. Internal report 494, LRI, Orsay (France), June 1989. III

[Bou90a] A. Boudet. *Unification dans les mélanges de théories équationelles*. Thèse de Doctorat d'Université, Université de Paris-Sud, Orsay (France), February 1990. 11.2.3, 11.5, 11.3, 11.4

[Bou90b] A. Boudet. Unification in a combination of equational theories: An efficient algorithm. In M. E. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, volume 449 of *Lecture Notes in Computer Science*. Springer-Verlag, July 1990. 11, 11.3, 11.4

[Bou90c] Wadoud Bousdira. *Etude des Propriétés des Systèmes de Réécriture Conditionnelle. Mise en Oeuvre d'un Algorithme de Complétion*. PhD thesis, Institut National Polytechnique de Lorraine, 1990. 20.1

[Bou91] Adel Bouhoula. Preuve automatique par paramodulation, réécriture et induction. Rapport de DEA, Université Henri Poincaré – Nancy 1, September 1991. 22.7.2

[Bou94] A. Bouhoula. *Preuves automatiques par récurrence dans les théories conditionnelles.* Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, March 1994. 22.7, 22.7, 22.5, 22.7.2, 22.7.2, 22.7.2

[BPW89] T. B. Baird, G. E. Peterson, and Ralph W. Wilkerson. Complete sets of reductions modulo associativity, commutativity and identity. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 29–44. Springer-Verlag, April 1989. 21.1, 21.13

[BR87] Wadoud Bousdira and Jean-Luc Rémy. Reveur4: A laboratory for conditional rewriting. In F. J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, editors, *Proceedings of 4th Symposium on Theoretical Aspects of Computer Science*, volume 247 of *Lecture Notes in Computer Science*, pages 472–473. Springer-Verlag, 1987. 20.1

[BR93] Adel Bouhoula and Michaël Rusinowitch. Automatic case analysis in proof by induction. In Ruzena Bajcsy, editor, *Proceedings 13th International Joint Conference on Artificial Intelligence, Chambéry (France)*, volume 1, pages 88–94. Morgan Kaufmann, August 1993. 22.7.1

[Bro75] T. Brown. *A structured design-method for specialized proof procedures.* PhD thesis, California Institute of Technology, Pasadena, California, 1975. 17.7

[BS86] R. V. Book and J. Siekmann. On unification: Equational theories are not bounded. *Journal of Symbolic Computation*, 2:317–324, 1986. 10.9

[BS87] W. Büttner and H. Simonis. Embedding boolean expressions into logic programming. *Journal of Symbolic Computation*, 4(2):191–205, 1987. 13.2.1, 13.2.3

[BS92] Franz Baader and Klaus Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. In *Proceedings 11th International Conference on Automated Deduction, Saratoga Springs (N.Y., USA)*, pages 50–65, 1992. 11, 11.3, 11.3.4, 11.3.4, 11.3.5

[Buc79] B. Buchberger. A criterion for detecting unnecessary reductions in the construction of gröbner bases. In *Proceedings of EUROSAM'79*, volume 72 of *Lecture Notes in Computer Science*, pages 3–21. Springer-Verlag, 1979. 19.3

[Buc85] B. Buchberger. *Multidimensional Systems Theory*, chapter Gröbner bases: an algorithmic method in polynomial ideal theory, pages 184–232. Reidel, Bose, N.K. Ed., 1985. 24.1, 24.4

[Bun87] R. Bundgen. Design, implementation and application of an extended ground reducibility test. Technical Report 88-05, University of Delaware USA, December 1987. 22.4

[Bün91] R. Bündgen. Simulation Buchberger's Algorithm by Knuth-Bendix Completion. In Book [Boo91], pages 386–397. 24.6

[Bür86] H.-J. Bürckert. Some relationships between unification, restricted unification and matching. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 514–524. Springer-Verlag, 1986. 11

[Bür89] H.-J. Bürckert. Matching — A special case of unification? *Journal of Symbolic Computation*, 8(5):523–536, 1989. III, 10.3

[Bür90] H.-J. Bürckert. A resolution principle for clauses with constraints. In M. E. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, volume 449 of *Lecture Notes in Computer Science*, pages 178–192. Springer-Verlag, July 1990. 21.5

[Büt85] W. Büttner. Unification in the datastructure multiset. Memo SEKI 85-V-KL, Universität Kaiserslautern, 1985. 13.1.5, 13.1.5

[CB83] J. Corbin and M. Bidoit. A rehabilitation of Robinson's unification algorithm. In R. Pavon, editor, *Proceedings of 1983 IFIP Congress*, pages 909–914. Elsevier Science Publishers B. V. (North-Holland), 1983. III, 10.1

[CD85] R. J. Cunningham and A. J. J. Dick. Rewrite systems on a lattice of types. *Acta Informatica*, 22(2):149–169, 1985. III

[CD89] E. Contejean and H. Devie. Solving systems of linear diophantine equations. In H.-J. Bürckert and W. Nutt, editors, *Proceedings 3rd International Workshop on Unification, Lambrecht (Germany)*, June 1989. III

[CD91] E. Contejean and H. Devie. Résolution de systèmes linéaires d'équations diophantiennes. *Compte-rendus de l'Académie des Sciences de Paris*, 1991. 13.1.5

[CF90] M. Clausen and A. Fortenbacher. Efficient solution of linear diophantine equations. In Claude Kirchner, editor, *Unification*, pages 377–392. Academic Press inc., London, 1990. 13.1.5

[CG91] P.-L. Curien and G. Ghelli. On Confluence for Weakly Normalizing Systems. In Book [Boo91], pages 215–225. 4.6.3, 4.4, 4.6.3

[Cha94] Jacques Chabin. *Unification Générale par Surréduction Ordonnée Contrainte et Surréduction Dirigée*. Thèse de Doctorat d'Université, Université d'Orléans, January 1994. 14.2.1, 14.2.4

[CHK90] H. Chen, J. Hsiang, and H. C. Kong. On finite representations of infinite sequences of terms. In S. Kaplan and M. Okada, editors, *Proceedings 2nd International Workshop on Conditional and Typed Rewriting Systems, Montreal (Canada)*, volume 516 of *Lecture Notes in Computer Science*, pages 100–114. Springer-Verlag, June 1990. 16.6

[Chr92] J. Christian. Some termination criteria for narrowing and E-narrowing. In D. Kapur, editor, *Proceedings 11th International Conference on Automated Deduction, Saratoga Springs (N.Y., USA)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 582–588. Springer-Verlag, June 1992. 14.2

[CL87] J. Christian and P. Lincoln. Adventures in associative-commutative unification. Technical Report ACA-ST-272-87, MCC, 1987. 13.1.1, 13.1.5

[CL91] E. A. Cichon and P. Lescanne. Polynomial Interpretations and the Complexity of Algorithms. Rapport interne 91-R-151, Centre de Recherche en Informatique de Nancy, Vandœuvre-lès-Nancy, December 1991. to be presented at CADE'92. 6.3.2

[Coh81] Paul M. Cohn. *Universal Algebra*. Reidel, D., Dordrecht, Holland, second edition, 1981. 2

[Col84] A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of FGCS'84*, pages 85–99, November 1984. 3.3, 3.4, III

[Col89] A. Colmerauer. Une introduction à Prolog III. In J.-P. Haton, editor, *Actes des onzièmes journées francophones sur l'informatique : Architectures Avancées pour l'Intelligence Artificielle*, pages 195–218, Nancy (France), January 1989. EC2. III

[Com88a] H. Comon. An effective method for handling initial algebras. In J. Grabowski, P. Lescanne, and W. Wechler, editors, *Proceedings 1st International Workshop on Algebraic and Logic Programming*, pages 108–118. Akademie Verlag, 1988. Also in Springer-Verlag, Lecture Notes in Computer Science, volume 343. 22.4

[Com88b] H. Comon. *Unification et disunification. Théories et applications.* Thèse de Doctorat d'Université, Institut Polytechnique de Grenoble (France), 1988. 10.1, 10.1

[Com91a] H. Comon. Completion of rewrite systems with membership constraints. Research report, CNRS-LRI, August 1991. 21.1

[Com91b] H. Comon. Disunification: a survey. In Jean-Louis Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 9, pages 322–359. The MIT press, Cambridge (MA, USA), 1991. III

[Com92] H. Comon. Completion of rewrite systems with membership constraints. In W. Kuich, editor, *Proceedings of ICALP 92*, volume 623 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992. 21.1, 21.5

[Cou80] B. Courcelle. Infinite trees in normal form and recursive equations having a unique solution. *Math. Syst. Theory*, 1980. 2.2.5

[Cou83] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25(2):95–169, March 1983. 2.2.5

[CP61] A. H. Clifford and G. B. Preston. *The algebraic theory of semigroups.* Number 7 in Mathematical surveys. American Mathematical Society, 1961. Il y a deux tomes. Le second a ete publie en 67. 13.1.4

[CR80] B. Courcelle and J.-C. Raoult. Completions of ordered magmas. *Fundamenta Informaticae*, 3(1):105–116, 1980. 2.2.5

[CR91] J. Chabin and P. Réty. Narrowing Directed by a Graph of Terms. In Book [Boo91], pages 112–123. 14.2.4

[CSY89] S. C. Chou, W. F. Schelter, and J. G. Yang. Characteristics sets and Gröbner bases in geometry theorem proving. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 2, pages 33–91. Academic Press inc., New York, 1989. 24.5

[Cur86] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Pitman, 1986. 16.11

[CZ90] R. Caferra and N. Zabel. A method for simultaneous search for refutations and models using equational problems. submitted, 1990. 21.5

[Dau89] M. Dauchet. Simulation of Turing machines by a left-linear rewrite rule. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 109–120. Springer-Verlag, April 1989. 6.2

[dC84] D. de Champeaux. About the paterson-wegman linear unification algorithm. Technical report, Tulane University, New Orleans, April 1984. 3.2.5

[Del86] J.-P. Delahaye. *Outils logiques pour l'intelligence artificielle.* Eyrolles, Paris (France), 1986. 2

[Der82] N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982. 6.3, 6.3

[Der83a] P. Deransart. An operational algebraic semantics of prolog programs. Technical report, INRIA, Le Chesnay (France), 1983. 14.2.5

[Der83b] N. Dershowitz. Applications of the Knuth-Bendix completion procedure. Technical Report ATR-83(8478)-2, The Aerospace Corporation, El Segundo, Calif. 90245, May 1983. 22.1

[Der85a] N. Dershowitz. Computing with rewrite systems. *Information and Control*, 65(2/3):122–157, 1985. 22.4

[Der85b] N. Dershowitz. Termination. In *Proceedings 1st Conference on Rewriting Techniques and Applications, Dijon (France)*, volume 202 of *Lecture Notes in Computer Science*, pages 180–224, Dijon (France), May 1985. Springer-Verlag. 6.2

[Der87] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1 & 2):69–116, 1987. 6.5, 7.2.2, 24.7

[Der89] N. Dershowitz. Completion and its applications. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 31–86. Academic Press inc., 1989. 16.9

[Der90] N. Dershowitz. A maximal-literal unit strategy for Horn clauses. In S. Kaplan and M. Okada, editors, *Proceedings of the 2nd International Workshop on Conditional and Typed Rewriting Systems*, volume 516 of *Lecture Notes in Computer Science*, pages 14–25. Springer-Verlag, 1990. 20.1, 20.5

[Dev91] H. Devie. *Procédures de complétion équationnelle.* PhD thesis, Université de Paris-Sud, 1991. 15.2

[DG89] J. Darlington and Y. Guo. Narrowing and unification in functional programming — an evaluation mechanism for absolute set abstractions. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 92–108. Springer-Verlag, April 1989. 14.2.5

[DHLT87] M. Dauchet, T. Heuillard, P. Lescanne, and S. Tison. Decidability of the confluence of ground term rewriting systems. In *Proceedings 2nd IEEE Symposium on Logic in Computer Science, Ithaca (N.Y., USA)*, pages 353–359, 1987. 5.7

[Dic13] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Am. J. Math.*, 35:413–426, 1913. 24.2

[DJ90a] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. 1, 3.1.4, 5.1, 8.2, III, 21.2

[DJ90b] D. Dougherty and P. Johann. An improved general E-unification method. In M. E. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, volume 449 of *Lecture Notes in Computer Science*, pages 261–275. Springer-Verlag, July 1990. 14.1.1

[DJ91] N. Dershowitz and J.-P. Jouannaud. Notations for rewriting. *Bulletin of European Association for Theoretical Computer Science*, 43:162–172, February 1991. 1, 5.1

[DJvP89] L. Duponcheel, L. Jadoul, and W. van Puymbroeck. Generic proofs by consistency. Technical report, Bell telephone Mfg. Co., Francis Wellesplein 1, B-2018 Antwerp, Belgium, 1989. 23.3.6, 2

[DK91] N. Doggaz and Claude Kirchner. Completion for unification. *Theoretical Computer Science*, 85(1):231–251, 1991. 12.8, 12.4.3

[DKM84] C. Dwork, P. Kanellakis, and J. C. Mitchell. On the sequential nature of unification. *Journal of Logic Programming*, 1(1):35–50, 1984. 3.2.5

[DKR93] E. Domenjoud, F. Klay, and C. Ringeissen. Combination of Unification Algorithms for Non-disjoint Equational Theories. In *Presented at UNIF'93*, Boston (MA, USA), 1993. 11.4

[DM79] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979. 4.4

[DMR76] M. Davis, Y. Matijasevič, and J. A. Robinson. Hilbert's tenth problem: Positive aspects of a negative solution. In *F. E. Browder, Editor, Mathematical Developments Arising from Hilbert Problems, American Mathematical Society*, pages 323–378, 1976. III, 10.1, 10.3

[DMT88] N. Dershowitz, L. Marcus, and A. Tarlecki. Existence, uniqueness and construction of rewrite systems. *SIAM Journal of Computing*, 17(4):629–639, August 1988. 17.1

[DO88] N. Dershowitz and M. Okada. Conditional equational programming and the theory of conditional term rewriting. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 337–346. Institute for New Generation Computer Technology, 1988. 7.5.1

[DO90] N. Dershowitz and M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990. 7.5, 7.5.1, 7.5.1, 7.5, 7.5.2, 7.5.4, 20.1, 20.2

[Dom91a] E. Domenjoud. *Outils pour la déduction automatique dans les théories associatives-commutatives.* Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, September 1991. 21.1

[Dom91b] E. Domenjoud. Solving systems of linear diophantine equations: An algebraic approach. In A. Tarlecki, editor, *Proceedings 16th International Symposium on Mathematical Foundations of Computer Science, Kazimierz Dolny (Poland)*, volume 520 of *Lecture Notes in Computer Science*, pages 141–150. Springer-Verlag, September 1991. 13.1.5

[Dom92] E. Domenjoud. A technical note on AC-unification. the number of minimal unifiers of the equation $\alpha x_1 + \cdots + \alpha x_p \doteq_{AC} \beta y_1 + \cdots + \beta y_q$. *Journal of Automated Reasoning*, 8:39–44, 1992. Also as research report CRIN 89-R-2. 21.1

[DOS87] N. Dershowitz, M. Okada, and G. Sivakumar. Confluence of conditional rewrite systems. In J.-P. Jouannaud and S. Kaplan, editors, *Proceedings 1st International Workshop on Conditional Term Rewriting Systems, Orsay (France)*, volume 308 of *Lecture Notes in Computer Science*, pages 31–44. Springer-Verlag, July 1987. 7.1, 7.5, 20.1, 20.4, 20.3

[Dou90] D. Dougherty. Higher-order unification via combinators. Technical report, Wesleyan University, May 1990. Presented at UNIF'90, Leeds (UK). III

[DP88] N. Dershowitz and D. A. Plaisted. Equational programming. In J. E. Hayes, D. Michie, and J. Richards, editors, *Machine Intelligence 11: The logic and acquisition of knowledge*, chapter 2, pages 21–56. Oxford Press, Oxford, 1988. 7.1, 20.1, 20.1

[Dro83] K. Drosten. Towards executable specifications using conditional axioms. Technical report, universität Braunschwig, 1983. 7.1, 7.5

[Dro92] N. Drost. Unification in the algebra of sets with union and empty set. Technical Report P9213, University of Amsterdam, jul, 1992. 10.1

[DS88] N. Dershowitz and G. Sivakumar. Goal-directed equation solving. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 166–170, St. Paul, (MN, USA), August 1988. 14.2, 14.2.4

[DSvH87] M. Dincbas, H. Simonis, and P. van Hentenryck. Extending equation solving and constraint handling in logic programming. In *Proceedings of the Colloquium on Resolution of Equations in Algebraic Structures*, Austin (Texas), May 1987. 13.2.5

[dV91] R. C. de Vrijer. Unique normal forms for combinatory logic with parallel conditional, a case study in conditional rewriting, 1991. Submitted. 4.4

[Ede85] E. Eder. Properties of substitutions and unifications. *Journal of Symbolic Computation*, 1(1):31–46, 1985. 2.3.3

[Eke93] Steven Eker. Improving the efficiency of AC matching and unification. Research report 2104, INRIA, Inria Lorraine & Crin, November 1993. 13.1.5

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1. Equations and initial semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985. 23.3.2, 23.3, 23.3.3, 23.13, 23.3, 23.4, 23.9

[Eva51] T. Evans. On multiplicative systems defined by generators and relations. In *Proceedings of the Cambridge Philosophical Society*, pages 637–649, 1951. 16.1

[Fag83] F. Fages. *Formes Canoniques dans les Algèbres Booléennes et Application à la Démonstration Automatique en Logique du Premier Ordre.* Thèse de Doctorat de Troisième Cycle, Université de Paris 7 (France), 1983. 3.4, 10.1

[Fag84] F. Fages. Associative-commutative unification. In R. Shostak, editor, *Proceedings 7th International Conference on Automated Deduction, Napa Valley (Calif., USA)*, volume 170 of *Lecture Notes in Computer Science*, pages 194–208. Springer-Verlag, 1984. III, 10.1, 13.1.1

[Far88] W. Farmer. A unification algorithm for second order monadic terms. *Annals of Pure and Applied Logic*, 39:131–174, 1988. III

[Fay79] M. Fay. First order unification in equational theories. In *Proceedings 4th Workshop on Automated Deduction, Austin (Tex., USA)*, pages 161–167, 1979. III, 14.2

[Fer92] M. Fernández. Narrowing based procedures for equational disunification. *Applicable Algebra in Engineering, Communication and Computation*, 3:1–26, 1992. 14.2.5

[FH83] F. Fages and G. Huet. Unification and matching in equational theories. In *Proceedings Fifth Colloquium on Automata, Algebra and Programming, L'Aquila (Italy)*, volume 159 of *Lecture Notes in Computer Science*, pages 205–220. Springer-Verlag, 1983. 10.2.3

[FH86] F. Fages and G. Huet. Complete sets of unifiers and matchers in equational theories. *Theoretical Computer Science*, 43(1):189–200, 1986. III, 10.2.2, 10.3

[Fil78] R. Filman. Personal communication in [Der87], 1978. 6.5

[FJN93] F. Freese, J. Jezek, and J.B. Nation. Term rewrite systems for lattice theory. *Journal of Symbolic Computation*, 16(3):279–288, 1993. 18.8

[For83] A. Fortenbacher. Algebraische unifikation. Diplomarbeit, Institut für Informatik, Universität Karlsruhe, 1983. III, 13.1.5

[Fos53] A. L. Foster. Generalized "boolean" theory of universal algebras. *Math. Zeitschr.*, Bd. 59:191–199, 1953. 13.2.5

[Fri85a] L. Fribourg. SLOG: A logic programming language intepreter based on clausal superposition and rewriting. In *Proceedings of the IEEE Symposium on Logic Programming*, pages 172–184, Boston, MA, July 1985. 7.1

[Fri85b] L. Fribourg. SLOG: A logic programming language interpreter based on clausal superposition and rewriting. In *IEEE Symposium on Logic Programming*, Boston (MA), 1985. 14.2, 14.2.5

[Fri86] L. Fribourg. A strong restriction of the inductive completion procedure. In *Proceedings 13th International Colloquium on Automata, Languages and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 105–115. Springer-Verlag, 1986. 22.1, 22.6, 22.6.1

[Gal86] Jean H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*, volume 5 of *Computer Science and Technology Series.* Harper & Row, New York, 1986. 10.1

[Gan83] H. Ganzinger. Parameterized specifications: parameter passing and implementation with respect to observability. *ACM Transactions on Programming Languages and Systems*, 5(3):318–354, 1983. 23.3.2, 23.7, 23.4

[Gan87] H. Ganzinger. Ground term confluence in parametric conditional equational specifications. In F. J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, editors, *Proceedings STACS 87*, volume 247 of *Lecture Notes in Computer Science*, pages 286–298. Springer-Verlag, 1987. 23.3.2, 23.4

[Gan91] H. Ganzinger. A completion procedure for conditional equations. *Journal of Symbolic Computation*, 11:51–81, 1991. 20.1, 20.10

[GBT89] J. Gallier and V. Breazu-Tannen. Polymorphic rewriting conserves algebraic strong normalization and confluence. In *16th Colloquium Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 137–150. Springer-Verlag, 1989. 8.6

[Ges90] A. Geser. *Relative Termination.* PhD thesis, Universität Passau (Germany), 1990. 8.6

[GL86] I. Gnaedig and P. Lescanne. Proving termination of associative rewriting systems by rewriting. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 52–61. Springer-Verlag, 1986. 7.3.3

[GM86] J. A. Goguen and J. Meseguer. EQLOG: Equality, types, and generic modules for logic programming. In Douglas De Groot and Gary Lindstrom, editors, *Functional and Logic Programming*, pages 295–363. Prentice Hall, Inc., 1986. An earlier version appears in *Journal of Logic Programming*, Volume 1, Number 2, pages 179–210, September 1984. 7.1, III, 14.2, 14.2.5

[GM87] E. Giovannetti and C. Moiso. A completeness result for conditional narrowing. In *Proceedings 1st International Workshop on Conditional Term Rewriting Systems, Orsay (France)*, volume 308 of *Lecture Notes in Computer Science*. Springer-Verlag, July 1987. 7.5.1

[GMP83] J. A. Goguen, J. Meseguer, and D. Plaisted. Programming with parameterized abstract objects in OBJ. In Domenico Ferrari, Mario Bolognani, and J. A. Goguen, editors, *Theory and Practice of Software Technology*, pages 163–193. Elsevier Science Publishers B. V. (North-Holland), 1983. 23.3.2

[GNP$^+$88] J. Gallier, P. Narendran, D. Plaisted, S. Raatz, and W. Snyder. Finding canonical rewriting systems equivalent to a finite set of ground equations in polynomial time. In E. Lusk and R. Overbeek, editors, *Proceedings 9th International Conference on Automated Deduction, Argonne (Ill., USA)*, volume 310 of *Lecture Notes in Computer Science*, pages 182–196. Springer-Verlag, 1988. 16.5

[Gog80] Joseph Goguen. How to prove algebraic inductive hypotheses without induction, with applications to the correctness of data type implementation. In W. Bibel and R. Kowalski, editors, *Proceedings 5th International Conference on Automated Deduction, Les Arcs (France)*, volume 87 of *Lecture Notes in Computer Science*, pages 356–373. Springer-Verlag, 1980. 22.1

[Gog84] J. A. Goguen. Parameterized programming. *Transactions on Software Engineering*, SE-10(5):528–543, September 1984. 23.3.2

[Gog89] J. A. Goguen. What is unification? In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 1, pages 217–262. Academic Press inc., New York, 1989. 10.1

[Gol81] D. Goldfarb. The undecidability of the second order unification problem. *Theoretical Computer Science*, 13:225–230, 1981. III

[Grä79] G. Grätzer. *Universal Algebra*. Springer-Verlag, second edition, 1979. 2, 2.4.7, 24.7

[Gra88] B. Gramlich. Unification of term schemes - theory and applications. Technical Report SR-88.18, SEKI, University of Kaiserslautern, Germany, 1988. 16.6

[Gra89] B. Gramlich. Inductive theorem proving using refined unfailing completion techniques. Technical Report SR-89-14, SEKI, Universität Kaiserslautern, Germany, 1989. 22.1, 22.6.1

[Gra92] Bernhard Gramlich. Generalized sufficient conditions for modular termination of rewriting. In Levi and Kirchner [LK92], pages 53–68. 8.5, 8.3.3

[GRS89] J. Gallier, S. Raatz, and W. Snyder. Rigid E-unification and its applications to equational matings. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 1, pages 151–216. Academic Press inc., New York, 1989. III

[GS87] J. Gallier and W. Snyder. A general complete E-unification procedure. In P. Lescanne, editor, *Proceedings 2nd Conference on Rewriting Techniques and Applications, Bordeaux (France)*, volume 256 of *Lecture Notes in Computer Science*, pages 216–227, Bordeaux (France), 1987. Springer-Verlag. 14.1.1

[GS89] J. Gallier and W. Snyder. Complete sets of transformations for general E-unification. *Theoretical Computer Science*, 67(2-3):203–260, October 1989. III, 14.1.1

[GS90] J. Gallier and W. Snyder. Designing unification procesures using transformations: A survey. Technical report, Boston University, 1990. III

[GTW78] J. A. Goguen, J. W. Thatcher, and E. G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming methodology IV: Data structuring*, pages 80–144. Prentice Hall, Inc., 1978. 2.6.3

[Gut78] John V. Guttag. Abstract data types and software validation. *Communications of the ACM*, 21:1048–1064, 1978. 23.2.1

[Han90] M. Hanus. Compiling logic programs with equality. In *Proceedings of PLILP'90*, volume 456 of *Lecture Notes in Computer Science*, pages 387–401. Springer-Verlag, 1990. 14.2.5

[Han94] Michael Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994. 14.2.5

[HD83] J. Hsiang and N. Dershowitz. Rewrite methods for clausal and non-clausal theorem proving. In *Proceedings of 10th International Colloquium on Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 331–346, Barcelona (Spain), 1983. Springer-Verlag. 13.2.2, 17.8

[Hee86] J. Heering. Partial evaluation and ω-completeness of algebraic specifications. *Theoretical Computer Science*, 43:149–167, 1986. 22.3

[Hen77] L. Henkin. The logic of equality. *The American Mathematical Monthly*, 84:597–612, October 1977. 2, 22.3

[Hen89] F. Henglein. *Polymorphic Type Inference and Semi-Unification.* PhD thesis, Courant Institute of Mathematical Sciences, New York University, May 1989. Appears as Technical report 443. III

[Her30] J. Herbrand. Recherches sur la théorie de la démonstration. *Travaux de la Soc. des Sciences et des Lettres de Varsovie, Classe III*, 33(128), 1930. III, 10.1, 10.5.1

[Her86] A. Herold. Combination of unification algorithms. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 450–469. Springer-Verlag, 1986. 11

[Her87] A. Herold. *Combination of Unification Algorithms in Equational Theories.* PhD thesis, Universität Kaiserslautern (Germany), 1987. 10.1, 13.1.7

[Her88] M. Hermann. Vademecum of divergent term rewriting systems. Research report 88–R–082, Centre de Recherche en Informatique de Nancy, 1988. Presented at *Term Rewriting Workshop, Bristol (UK).* 16.6

[Her89] M. Hermann. Crossed term rewriting systems. Research report 89-R-003, Centre de Recherche en Informatique de Nancy, 1989. 16.6

[Her91] M. Hermann. On proving properties of completion strategies. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 398–410. Springer-Verlag, April 1991. 15.3

[HH82] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, October 1982. Preliminary version in Proceedings 21st Symposium on Foundations of Computer Science, IEEE, 1980. 22.1

[Hig52] G. Higman. Ordering by divisibility in abstract algebra. *Proceedings of the London Mathematical Society*, 2(7):326–336, September 1952. 4.5, 6.4.1

[Hin64] J. R. Hindley. *The Church-Rosser property and a result in combinatory logic.* PhD thesis, University of Newcastle-upon-Tyne, 1964. 8.5.1, 8.1, 8.5.1

[Hin94] C. Hintermeier. A transformation of canonical conditional trs's into equivalent canonical trs's. In *Proceedings of the 4th International Workshop on Conditional Rewriting Systems, Jerusalem (Israel)*, June 1994. 7.5.4

[HK88] D. Hofbauer and R. D. Kutsche. Proving inductive theorems based on term rewriting systems. In J. Grabowski, P. Lescanne, and W. Wechler, editors, *Proceedings 1st International Workshop on Algebraic and Logic Programming*, pages 180–190. Akademie Verlag, 1988. 22.1

[HKK89] M. Hermann, Claude Kirchner, and Hélène Kirchner. Implementations of term rewriting systems. Technical Report 89-R-218, Centre de Recherche en Informatique de Nancy, 1989. To appear in the *Computer Journal*, British Computer Society. 16.1

[HL78a] G. Huet and B. Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978. III

[HL78b] G. Huet and D. S. Lankford. On the uniform halting problem for term rewriting systems. Technical Report 283, Laboria, France, 1978. 5.7, 6.2

[HL89] D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 167–177. Springer-Verlag, April 1989. 6.1

[HM88] J. Hannan and D. Miller. Uses of higher-order unification for implementing program transformers. In R. Kowalski and K. Bowen, editors, *Proceedings of the Logic Programming Conference*, Seattle (USA), 1988. The MIT press. III

[HO80] G. Huet and D. Oppen. Equations and rewrite rules: A survey. In R. V. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic Press inc., 1980. 1, 5.1, 6.3.2

[Höl88] S. Hölldobler. From paramodulation to narrowing. In *5th International Conference/Symposium on Logic Programming*, 1988. 14.2.5

[Höl89] S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1989. 14.1.1

[HR86] J. Hsiang and M. Rusinowitch. A new method for establishing refutational completeness in theorem proving. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 141–152. Springer-Verlag, 1986. 17.7

[HR87] J. Hsiang and M. Rusinowitch. On word problem in equational theories. In Th. Ottmann, editor, *Proceedings of 14th International Colloquium on Automata, Languages and Programming, Karlsruhe (Germany)*, volume 267 of *Lecture Notes in Computer Science*, pages 54–71. Springer-Verlag, 1987. 17.2, 17.7

[HRS87] J. Hsiang, M. Rusinowitch, and K. Sakai. Complete inference rules for the cancellation laws. In *Int. Joint Conf. on Artificial Intelligence*, 1987. 19.2.4

[HS85] A. Herold and J. Siekmann. Unification in abelian semigroups. Memo SEKI 85-III-KL, Universität Kaiserslautern, 1985. 10.1, 13.1.1, 13.1.7

[Hue72] G. Huet. *Constrained Resolution: A Complete Method for Higher Order Logic.* PhD thesis, Case Western Reserve University, 1972. 7.6, III

[Hue73a] G. Huet. A mechanization of type theory. In *Proceeding of the third international joint conference on artificial intelligence*, pages 139–146, 1973. III

[Hue73b] G. Huet. The undecidability of unification in third order logic. *Information and Control*, 22:257–267, 1973. III

[Hue75] G. Huet. A unification algorithm for typed lambda calculus. *Theoretical Computer Science*, 1(1):27–57, 1975. III

[Hue76] G. Huet. *Résolution d'equations dans les langages d'ordre 1,2, ...,ω.* Thèse de Doctorat d'Etat, Université de Paris 7 (France), 1976. 2.3.3, 2.3.3, 3.2, 3.1.2, 3.1.3, 3.1.4, 3.1.5, 3.2.5, 3.3, 3.4, III, 10.1, 10.1, 10.2.1, 10.1

[Hue78] G. Huet. An algorithm to generate the basis of solutions to homogenous linear diophantine equations. *Information Processing Letters*, 7(3):144–147, 1978. III, 13.1.5, 1

[Hue80] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, October 1980. 4.6.1, 5.6, 7.1, 7.3, 7.3.1, 7.3.2, 8.7, 16.1, 16.1, 16.1, 18.1, 18.1, 18.2.3, 18.1

[Hue81] G. Huet. A complete proof of correctness of the Knuth and Bendix completion algorithm. *Journal of Computer and System Sciences*, 23:11–21, 1981. 12.4.3, 15.3, 16.6, 16.6

[Hul79] J.-M. Hullot. Associative-commutative pattern matching. In *Proceedings 9th International Joint Conference on Artificial Intelligence*, 1979. 12.3

[Hul80a] J.-M. Hullot. Canonical forms and unification. In W. Bibel and R. Kowalski, editors, *Proceedings 5th International Conference on Automated Deduction, Les Arcs (France)*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer-Verlag, July 1980. III, 10.1, 14.2, 14.2.1, 14.2.1, 14.2.2, 18.1

[Hul80b] J.-M. Hullot. A catalog of canonical term rewriting systems. Technical report, Stanford Research Institute, 1980. 7.9, 16.4

[Hul80c] J.-M. Hullot. *Compilation de Formes Canoniques dans les Théories équationelles.* Thèse de Doctorat de Troisième Cycle, Université de Paris Sud, Orsay (France), 1980. 7.9, 10.1, 10.2.1, 10.1, 13.1.1, 13.1.5, 13.1.5, 14.2.1, 14.2.2, 16.4

[Hus85] H. Hussmann. Unification in conditional equational theories. In B. Buchberger, editor, *Proceedings of the EUROCAL Conference, Linz (Austria)*, volume 204 of *Lecture Notes in Computer Science*, pages 543–553. Springer-Verlag, 1985. 14.2.4

[Hus88] H. Hussmann. *Nondeterministic Algebraic Specifications.* PhD thesis, Universität Passau (Germany), September 1988. English literal translation, Nov. 1990. 5.3

[Jaf84] J. Jaffar. Efficient unification over infinite terms. *New Generation Computing*, 2:207–219, 1984. 3.4

[Jaf90] J. Jaffar. Minimal and complete word unification. *Journal of the ACM*, April 1990. 10.1

[JD86] N. Alan Josephson and Nachum Dershowitz. An implementation of narrowing: The RITE way. In *Proceedings of the Symposium on Logic Programming*, pages 187–197, Salt Lake City, UT, September 1986. 14.2.5

[Jea80] J. Jeanrond. Deciding unique termination of permutative rewrite systems: Choose your term algebra carefully. In W. Bibel and R. Kowalski, editors, *Proceedings 5th International Conference on Automated Deduction, Les Arcs (France)*, volume 87 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980. 10.1

[JK84] J.-P. Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. In *Proceedings 11th ACM Symposium on Principles of Programming Languages, Salt Lake City*, 1984. 12.4.3

[JK86a] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in theories without constructors. Technical Report 295, Université de Paris-Sud, Centre d'Orsay (France), September 1986. 22.4, 22.4

[JK86b] J.-P. Jouannaud and E. Kounalis. Proof by induction in equational theories without constructors. In *Proceedings 1st IEEE Symposium on Logic in Computer Science, Cambridge (Mass., USA)*, pages 358–366, 1986. 22.1, 22.4, 22.1, 22.5.2, 23.3.7

[JK86c] Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986. 7.3, 7.3.1, 7.4, 16.6, 18.1, 18.4.3, 18.5, 18.5, 18.6, 21.1

[JK89] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82:1–33, 1989. 22.4, 23.3.7

[JK91] J.-P. Jouannaud and Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In Jean-Louis Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. The MIT press, Cambridge (MA, USA), 1991. 3.4, III, 10.5.1

[JKK83] J.-P. Jouannaud, Claude Kirchner, and Hélène Kirchner. Incremental construction of unification algorithms in equational theories. In *Proceedings International Colloquium on Automata, Languages and Programming, Barcelona (Spain)*, volume 154 of *Lecture Notes in Computer Science*, pages 361–373. Springer-Verlag, 1983. 10.1, 14.2

[JL87] J. Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th Annual ACM Symposium on Principles Of Programming Languages, Munich (Germany)*, pages 111–119, 1987. 7.6, III

[JM84] J.-P. Jouannaud and M. Muñoz. Termination of a set of rules modulo a set of equations. In *Proceedings 7th International Conference on Automated Deduction, Napa Valley (Calif., USA)*, volume 170 of *Lecture Notes in Computer Science*. Springer-Verlag, 1984. 7.3.3

[JM90] J.-P. Jouannaud and C. Marché. Completion modulo associativity, commutativity and identity (AC1). In A. Miola, editor, *Proceedings of DISCO'90*, volume 429 of *Lecture Notes in Computer Science*, pages 111–120. Springer-Verlag, April 1990. 21.1

[Jou83] J.-P. Jouannaud. Confluent and coherent equational term rewriting systems. Applications to proofs in abstract data types. In G. Ausiello and M. Protasi, editors, *Proceedings of the 8th Colloquium on Trees in Algebra and Programming, L'Aquila (Italy)*, volume 159 of *Lecture Notes in Computer Science*, pages 269–283. Springer-Verlag, 1983. 7.3, 18.1, 18.3.1, 18.4.1

[Jou87] J.-P. Jouannaud. Proof algebras. *Invited lecture at the second Rewriting Techniques and Applications Conference, Bordeaux (France)*, 1987. 16.3

[JP76] D. Jensen and T. Pietrzykowski. Mechanizing ω-order type theory through unification. *Theoretical Computer Science*, 3(1):123–171, 1976. III

[JW86] J.-P. Jouannaud and B. Waldmann. Reductive conditional term rewriting systems. In M. Wirsing, editor, *3rd IFIP Conf. on Formal Description of Programming Concepts*, Ebberup, (Denmark), 1986. Elsevier Science Publishers B. V. (North-Holland). 7.1, 7.5, 7.5.2, 7.27, 20.1, 20.5

[Kap83] S. Kaplan. *Un langage de spécification de types abstraits algébriques*. Thèse de Doctorat de Troisième Cycle, Université d'Orsay, France, 1983. 2.6.3, 2.7, 7.5.2

[Kap84] S. Kaplan. Conditional rewrite rules. *Theoretical Computer Science*, 33:175–193, 1984. 7.1, 7.5, 7.13, 7.5.2, 20.1

[Kap87] S. Kaplan. Simplifying conditional term rewriting systems: Unification, termination and confluence. *Journal of Symbolic Computation*, 4(3):295–334, December 1987. 7.1, 7.5, 7.5.2, 7.5.2, 20.1, 20.2, 20.3, 20.2

[KB70] Donald E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970. 5.1, 12.4.3, 16.1, 16.1, 16.1, 16.2

[KB91] S. Krischer and A. Bockmayr. Detecting Redundant Narrowing Derivations by the LSE-SL Reducability Test. In Book [Boo91], pages 74–85. 14.2, 14.2.4

[KdV90] J. W. Klop and R. de Vrijer. Extended term rewriting systems. In S. Kaplan and M. Okada, editors, *Proceedings of the 2nd International Workshop on Conditional and Typed Rewriting Systems*, volume 516 of *Lecture Notes in Computer Science*, pages 26–50. Springer-Verlag, 1990. 20.6, 20.1, 20.5

[Ken89] J. R. Kennaway. Sequential evaluation strategies for parallel-or and related reduction systems. *Annals of Pure and Applied Logic*, 43:31–56, 1989. 5.3

[KH90] Hélène Kirchner and M. Hermann. Meta-rule synthesis from crossed rewrite systems. In S. Kaplan and M. Okada, editors, *Proceedings 2nd International Workshop on Conditional and Typed Rewriting Systems, Montreal (Canada)*, volume 516 of *Lecture Notes in Computer Science*, pages 143–154. Springer-Verlag, June 1990. 16.6

[Kir84a] Claude Kirchner. A new equational unification method: A generalization of Martelli-Montanari algorithm. In R. Shostak, editor, *Proceedings 7th International Conference on Automated Deduction, Napa Valley (Calif., USA)*, volume 170 of *Lecture Notes in Computer Science*. Springer-Verlag, 1984. 10.1, 14.1.1

[Kir84b] Hélène Kirchner. A general inductive completion algorithm and application to abstract data types. In R. Shostak, editor, *Proceedings 7th International Conference on Automated Deduction, Napa Valley (Calif., USA)*, volume 170 of *Lecture Notes in Computer Science*, pages 282–302. Springer-Verlag, 1984. 22.5.2

[Kir85a] Claude Kirchner. *Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles*. Thèse de Doctorat d'Etat, Université Henri Poincaré – Nancy 1, 1985. 10.1, 10.1, 11, 11.2, 12.1.1, 12.4, 12.8

[Kir85b] Hélène Kirchner. *Preuves par complétion dans les variétés d'algèbres*. Thèse de Doctorat d'Etat, Université Henri Poincaré – Nancy 1, 1985. 18.1

[Kir86] C. Kirchner. Computing unification algorithms. In *Proceedings 1st IEEE Symposium on Logic in Computer Science, Cambridge (Mass., USA)*, pages 206–216, 1986. III, 10.1, 12.4

[Kir88] Claude Kirchner. Order-sorted equational unification. Presented at the fifth International Conference on Logic Programming (Seattle, USA), August 1988. Also as rapport de recherche INRIA 954, Dec. 88. III

[Kir89a] Claude Kirchner. From unification in combination of equational theories to a new AC-unification algorithm. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 171–210. Academic Press inc., New York, 1989. III, 10.1

[Kir89b] Hélène Kirchner. Schematization of infinite sets of rewrite rules generated by divergent completion processes. *Theoretical Computer Science*, 67(2-3):303–332, October 1989. 16.6

[Kir90] Claude Kirchner, editor. *Unification.* Academic Press inc., London, 1990. III

[KK82] Claude Kirchner and Hélène Kirchner. *Résolution d'équations dans les algèbres libres et les variétés équationnelles d'algèbres.* Thèse de Doctorat de Troisième Cycle, Université Henri Poincaré – Nancy 1, 1982. 10.1

[KK86] Claude Kirchner and Hélène Kirchner. Reveur-3: Implementation of a general completion procedure parametrized by built-in theories and strategies. *Science of Computer Programming*, 20(8):69–86, 1986. 18.1, 18.6

[KK89] Claude Kirchner and Hélène Kirchner. Constrained equational reasoning. In *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation, Portland (Oregon)*, pages 382–389. ACM Press, July 1989. Report CRIN 89-R-220. III, 14.2.1, 21.1

[KK90] C. Kirchner and F. Klay. Syntactic theories and unification. In *Proceedings 5th IEEE Symposium on Logic in Computer Science, Philadelphia (Pa., USA)*, pages 270–277, June 1990. 12.3.3, 12.3.4

[KKM88] Claude Kirchner, Hélène Kirchner, and J. Meseguer. Operational semantics of OBJ-3. In *Proceedings of 15th International Colloquium on Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 287–301. Springer-Verlag, 1988. 7.7

[KKR90] Claude Kirchner, Hélène Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on Automatic Deduction. 7.6.1, 14.2.1, 21.1, 21.3, 21.5

[KL80] S. Kamin and J.-J. Lévy. Attempts for generalizing the recursive path ordering. Unpublished manuscript, 1980. 6.4

[KL87] Claude Kirchner and P. Lescanne. Solving disequations. In D. Gries, editor, *Proceedings 2nd IEEE Symposium on Logic in Computer Science, Ithaca (N.Y., USA)*, pages 347–352. IEEE, 1987. 10.1

[Kla92] F. Klay. *Unification dans les Théories Syntaxiques.* Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, 1992. 12.1.2, 12.1, 12.2, 12.5, 12.6, 12.3.3, 12.4, 12.7

[KLMR90] H. Kuchen, R. Loogen, J. J. Moreno, and M. Rodriguez. Graph-based implementation of a functional language. In *Proceedings of ESOP'90*, volume 432 of *Lecture Notes in Computer Science*, pages 279–290. Springer-Verlag, 1990. 14.2.5

[Klo80] J. W. Klop. *Combinatory Reduction Systems.* PhD thesis, CWI, 1980. 5.6, 20.6

[Klo90a] J. W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter 6. Oxford University Press, 1990. 1

[Klo90b] J. W. Klop. Term rewriting systems. Technical Report CS-R9073, Centre for Mathematics and Computer Science, 1990. 20.2

[KM87] D. Kapur and D. R. Musser. Proof by consistency. *Artificial Intelligence*, 31(2):125–157, February 1987. 22.1

[KMN85] D. Kapur, D. R. Musser, and P. Narendran. Only prime superpositions need be considered in the Knuth-Bendix procedure, 1985. Computer Science Branch, Corporate Research and Development, General Electric, Schenectady, New York. 19.3

[KMN88] D. Kapur, D. R. Musser, and P. Narendran. Only prime superpositions need to be considered in the Knuth-Bendix completion procedure. *Journal of Symbolic Computation*, 6(2):19–36, 1988. 19.4

[KN85] D. Kapur and P. Narendran. A finite Thue system with decidable word problem and without equivalent finite canonical system. *Theoretical Computer Science*, 35:337–344, 1985. 5.4

[KN86] D. Kapur and P. Narendran. NP-completeness of the set unification and matching problems. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*. Springer-Verlag, 1986. 13.1.1

[Kni89] K. Knight. Unification: A multidisciplinary survey. *ACM Computing Surveys*, 21(1):93–124, March 1989. 3.4, 1, III

[KNO90] D. Kapur, P. Narendan, and F. Otto. On ground confluence of term rewriting systems. *Information and Computation*, May 1990. 5.7

[KNZ87] D. Kapur, P. Narendran, and H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica*, 24:395–415, 1987. 22.3, 22.4, 23.2.2, 23.3.8, 23.3.8

[KO90] M. Kurihara and A. Ohuchi. Modularity of simple termination of term rewriting systems. *Journal of IPS Japan*, 31(5):633–642, 1990. 8.5, 8.3.3

[KO92] M. Kurihara and A. Ohuchi. Modularity of simple termination of term rewriting systems with shared constructors. *Theoretical Computer Science*, 103(2):273–282, 1992. 8.9, 8.4, 8.10, 11.3

[Kou85] E. Kounalis. Completeness in data type specifications. In B. Buchberger, editor, *Proceedings EUROCAL Conference, Linz (Austria)*, volume 204 of *Lecture Notes in Computer Science*, pages 348–362. Springer-Verlag, 1985. 22.4

[Kou90] E. Kounalis. Testing for inductive (co)-reducibility. In A. Arnold, editor, *Proceedings 15th CAAP, Copenhagen (Denmark)*, volume 431 of *Lecture Notes in Computer Science*, pages 221–238. Springer-Verlag, May 1990. 22.4, 22.7.1

[KR87] E. Kounalis and M. Rusinowitch. On word problem in Horn logic. In J.-P. Jouannaud and S. Kaplan, editors, *Proceedings 1st International Workshop on Conditional Term Rewriting Systems, Orsay (France)*, volume 308 of *Lecture Notes in Computer Science*, pages 144–160. Springer-Verlag, July 1987. See also the extended version published in Journal of Symbolic Computation, 11(1 & 2), 1991. 20.1, 20.4

[KR89a] P. Kanellakis and P. Revesz. On the relationship of congruence closure and unification. *Journal of Symbolic Computation*, 7(3 & 4):427–444, 1989. Special issue on unification. Part one. 3.4

[KR89b] S. Kaplan and Jean-Luc Rémy. Completion algorithms for conditional rewriting systems. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 141–170. Academic Press inc., 1989. 7.6.4, 20.1, 20.5

[KR90] E. Kounalis and M. Rusinowitch. Mechanizing inductive reasoning. In *Proceedings of the American Association for Artificial Intelligence Conference, Boston*, pages 240–245. AAAI Press and MIT Press, July 1990. 22.1, 22.7, 22.7.1, 23.4

[KR92] Hélène Kirchner and Ch. Ringeissen. A constraint solver in finite algebras and its combination with unification algorithms. In K. Apt, editor, *Proc. Joint International Conference and Symposium on Logic Programming*, pages 225–239. The MIT press, 1992. 7.6.1, 11.4

[KR93] H. Kirchner and Ch. Ringeissen. Constraint solving by narrowing in combined algebraic domains (extended version). research report, CRIN-CNRS and INRIA-Lorraine, 1993. 7.6.3, 14.2.4, 21.3

[KR94a] H. Kirchner and C. Ringeissen. Combining symbolic constraint solvers on algebraic domains. *Journal of Symbolic Computation*, 18(2):113–155, 1994. 11.4, 13.4, 13.5

[KR94b] H. Kirchner and C. Ringeissen. Constraint solving by narrowing in combined algebraic domains. In P. Van Hentenryck, editor, *Proc. 11th International Conference on Logic Programming*, pages 617–631. The MIT press, 1994. 14.2.5

[Kru54] J. B. Kruskal. *The theory of well-partially-ordered sets.* PhD thesis, Princeton University, Princeton, N.J., 1954. 4.2.3

[Kru60] J. B. Kruskal. Well-quasi ordering, the tree theorem and Vazsonyi's conjecture. *Trans. Amer. Math. Soc.*, 95:210–225, 1960. 4.9, 4.2.3, 6.4.1

[Kru72] J. B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *J. Combinatorial Theory Ser. A*, 13(3):297–305, November 1972. 4.2.3

[Küc85] W. Küchlin. A confluence criterion based on the generalised Knuth-Bendix algorithm. In B. Buchberger, editor, *Proceedings of the EUROCAL Conference, Linz (Austria)*, volume 204 of *Lecture Notes in Computer Science*, pages 390–399. Springer-Verlag, 1985. 19.3

[Kuc88] G. A. Kucherov. A new quasi-reducibility testing algorithm and its application to proofs by induction. In J. Grabowski, P. Lescanne, and W. Wechler, editors, *Proceedings 1st International Workshop on Algebraic and Logic Programming*, pages 204–213. Akademie Verlag, 1988. Also in Springer-Verlag, Lecture Notes in Computer Science, volume 343. 22.4

[Küc89] W. Küchlin. Inductive completion by ground proof transformation. In H. Aït-Kaci and M. Nivat, editors, *Colloquium on the Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 211–244. Academic Press inc., 1989. 22.1, 22.6.2

[KZ91] D. Kapur and H. Zhang. A case study of the completion procedure: ring commutativity problems. In Jean-Louis Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 10, pages 360–394. The MIT press, Cambridge (MA, USA), 1991. 19.2, 19.3, 19.3

[Lam87a] J.-L. Lambert. Une borne pour les générateurs des solutions entières positives d'une équation diophantienne linéaire. *Compte-rendus de l'Académie des Sciences de Paris*, 305(1):39–40, 1987. III

[Lam87b] J.-L. Lambert. Une borne pour les générateurs des solutions entières positives d'une équation diophantienne linéaire. Technical Report 334, Université de Paris-Sud, Centre d'Orsay (France), February 1987. 13.1.5, 1

[Lan75a] D. S. Lankford. Canonical algebraic simplifications. Technical report, Louisiana Tech. University, 1975. 6.3.2

[Lan75b] Dallas Lankford. Canonical inference. Technical report, Louisiana Tech. University, 1975. 7.2.2, 16.5, 17.7

[Lan77] D. S. Lankford. Some approaches to equality for computational logic: A survey and assessment. Memo ATP-36, Automatic Theorem Proving Project, University of Texas, Austin (Texas, USA), 1977. 6.1

[Lan79a] D. S. Lankford. On proving term rewriting systems are noetherian. Technical report, Louisiana Tech. University, Mathematics Dept., Ruston LA, 1979. 7.2.2

[Lan79b] D. S. Lankford. A unification algorithm for abelian group theory. Technical report, Louisiana Tech. University, 1979. 6.3.2, 10.1

[Lan87] D. S. Lankford. Non-negative integer basis algorithms for linear equations with integer coefficients. Technical report, Louisiana Tech University (USA), Ruston, LA 71272, 1987. 13.1.5

[Lau88] C. Lautemann. A note on polynomial interpretation. *Bulletin of European Association for Theoretical Computer Science*, 1(36):129–131, October 1988. 6.1

[LB77a] D. S. Lankford and A. Ballantyne. Decision procedures for simple equational theories with associative commutative axioms: complete sets of associative commutative reductions. Technical report, Univ. of Texas at Austin, Dept. of Mathematics and Computer Science, 1977. 7.1, 18.1, 19.3

[LB77b] D. S. Lankford and A. Ballantyne. Decision procedures for simple equational theories with commutative axioms: complete sets of commutative reductions. Technical report, University of Texas at Austin, Dept. of Mathematics and Computer Science, 1977. 7.1, 18.1

[LB77c] D. S. Lankford and A. Ballantyne. Decision procedures for simple equational theories with permutative axioms: complete sets of permutative reductions. Technical report, Univ. of Texas at Austin, Dept. of Mathematics and Computer Science, 1977. 7.1, 18.1

[LBB84] D. S. Lankford, G. Butler, and B. Brady. Abelian group unification algorithms for elementary terms. In W. Bledsoe and W. Loveland, editors, *Automated Theorem Proving: After 25 Years*. AMS, 1984. 10.1

[LC89] P. Le Chenadec. On the logic of unification. *Journal of Symbolic Computation*, 8(1 & 2):141–200, 1989. Special issue on unification. Part two. 3.4, III

[Les84] P. Lescanne. Uniform termination of term rewriting systems. Recursive decomposition ordering with status. In B. Courcelle, editor, *Proceedings 9th Colloque sur les Arbres en Algèbre et en Programmation*, pages 182–194, Bordeaux (France), 1984. Cambridge University Press. 6.4.3

[Les86] P. Lescanne. Divergence of the Knuth-Bendix completion procedure and termination orderings. *Bulletin of European Association for Theoretical Computer Science*, 30:80–83, October 1986. 16.6

[Les89] P. Lescanne. Well quasi-orderings in a paper by Maurice Janet. *Bulletin of European Association for Theoretical Computer Science*, 39:185–188, October 1989. 4.2.3

[Lin89] N. Lindenstrauss. A parallel implementation of rewriting and narrowing. In *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 35 of *Lecture Notes in Computer Science*, pages 569–573. Springer-Verlag, April 1989. 14.2.5

[LK92] Giorgio Levi and Hélène Kirchner, editors. *Algebraic and Logic Programming, Third International Conference*, LNCS 632, Volterra, Italy, September 2–4, 1992. Springer-Verlag. 24.7

[LLT90] A. Lazrek, P. Lescanne, and J.-J. Thiel. Tools for proving inductive equalities, relative completeness and ω-completeness. *Information and Computation*, 84(1):47–70, January 1990. 22.3

[LMM88] Jean-Louis Lassez, M. J. Maher, and K. Marriot. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgan-Kaufman, 1988. 3.2.2, 3.2.2, 3.4, III

[Löw08] L. Löwenheim. Uber das auflösungsproblem im logischen klassenkalkül. *Sitzungsberg. Berl. Math. Gesell*, 7:89–94, 1908. 13.2.1, 13.2.3

[LPB$^+$87] G. Levi, C. Palamidessi, P. G. Bosco, E. Giovannetti, and C. Moiso. A complete semantic characterization of k-leaf, a logic language with partial functions. In *Proc. IEEE Symposium on Logic Programming*, pages 318–327, 1987. 14.2.5

[LS75] M. Livesey and J. Siekmann. Termination and decidability results for string unification. Technical report memo CSM-12, University of Essex, 1975. 10.1

[LS76] M. Livesey and J. Siekmann. Unification of bags and sets. Technical report, Institut für Informatik I, Universität Karlsruhe, 1976. 10.1, 13.1.1

[LS93] C. Lynch and W. Snyder. Redundancy criteria for constrained completion. In C. Kirchner, editor, *Proceedings 5th Conference on Rewriting Techniques and Applications, Montreal (Canada)*, volume 690 of *Lecture Notes in Computer Science*, pages 2–16. Springer-Verlag, 1993. 21.4.1

[Mak77] G. S. Makanin. The problem of solvability of equations in a free semigroup. *Math. USSR Sbornik*, 32(2):129–198, 1977. III, 10.1

[Mar91] Marché. On ground AC-completion. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 411–422. Springer-Verlag, April 1991. 18.7, 1

[Mar93] C. Marché. *Réécriture modulo une théorie présentée par un système convergent et décidabilité du problème du mot dans certaines classes de théories équationnelles.* Thèse de Doctorat d'Université, Université de Paris-Sud, Orsay (France), October 1993. 18.8

[Mat70] Y. Matijasevič. Diophantine representation of recursively enumerable predicates. In *Actes du Congrès International des Mathématiciens*, volume 1, pages 235–238, Nice (France), 1970. III, 10.1, 10.3

[Mes92] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992. 5.3, 7.6.5

[Met83] Y. Metivier. About the rewriting systems produced by the Knuth-Bendix completion algorithm. *Information Processing Letters*, 16(1):31–34, January 1983. 5.5, 16.4, 16.5

[MG85] J. Meseguer and J. A. Goguen. Initiality, induction and computability. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*. Cambridge University Press, 1985. 2.5.2, 22.2

[MGS87] J. Meseguer, J. A. Goguen, and G. Smolka. Order-sorted unification. In *Proceedings of the Colloquium on Resolution of Equations in Algebraic Structures*, Austin (Texas), May 1987. III

[MH92] Aart Middeldorp and Erik Hamoen. Counterexamples to completeness results for basic narrowing (extended abstract). In Levi and Kirchner [LK92], pages 244–258. 14.2, 14.2.3

[Mid89a] A. Middeldorp. Confluence of the disjoint union of conditional term rewriting systems. Technical Report CS-R8944, CWI, Amsterdam, October 1989. 8.1

[Mid89b] A. Middeldorp. Modular aspects of properties of term rewriting systems related to normal forms. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 263–277. Springer-Verlag, April 1989. 8.6

[Mid89c] A. Middeldorp. A sufficient condition for the termination of the direct sum of term rewriting systems. In *Proceedings 4th IEEE Symposium on Logic in Computer Science, Pacific Grove*, pages 396–401, 1989. 8.1, 8.3.2

[Mid89d] A. Middeldorp. Termination of disjoint unions of conditional term rewriting systems. Technical Report CS-R8959, CWI, Amsterdam, December 1989. 8.1

[Mid90] A. Middeldorp. *Modular Properties of Term Rewriting Systems*. PhD thesis, Vrije Universiteit, Amsterdam, 1990. 8.1, 8.2, 8.3, 8.3.1, 8.6

[Mid91] A. Middeldorp. Completeness of combinations of constructor systems, 1991. Submitted. 8.4, 8.8, 8.4, 8.5

[ML92] U. Martin and M. Lai. Some experiments with a completion theorem prover. *Journal of Symbolic Computation*, 13(1):81–100, January 1992. 18.2

[MM76] A. Martelli and U. Montanari. Unification in linear time and space: A structured presentation. Technical Report B76-16, University of Pisa, July 1976. 3.2.5

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982. 2, 3.2.5, III, 10.1, 10.5.1

[MN70] Z. Manna and S. Ness. On the termination of Markov algorithms. In *Proceedings of the Third Hawaii International Conference on System Science*, pages 789–792, Honolulu, Hawaii, 1970. 4.7

[MN86] D. A. Miller and G. Nadathur. Higher-order logic programming. In E. Shapiro, editor, *Proceedings of the Third International Logic Programming Conference*, volume 225 of *Lecture Notes in Computer Science*, pages 448–462. Springer-Verlag, 1986. III

[MN89] U. Martin and T. Nipkow. Boolean unification — the story so far. *Journal of Symbolic Computation*, 7(3 & 4):275–294, 1989. Special issue on unification. Part one. 10.4, 10.1, 13.2.1, 13.2.3, 13.2.3

[MN90] U. Martin and T. Nipkow. Ordered rewriting and confluence. In M. E. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, volume 449 of *Lecture Notes in Computer Science*, pages 366–380. Springer-Verlag, 1990. 7.2, 7.3, 7.15, 17.6, 21.1

[MNRA92] J. Moreno-Navarro and M. Rodriguez-Artalejo. Logic programming with functions and predicates: the language BABEL. *Journal of Logic Programming*, 12(3):191–223, February 1992. 14.2

[Moh89] C. K. Mohan. Priority rewriting: Semantics, confluence, and conditionals. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 278–291. Springer-Verlag, April 1989. 7.7

[MP88] Chao-Tai Mong and Paul W. Purdom. Divergence in the completion of rewriting systems. Internal report, Indiana University, Bloomington, IN 47405, 1988. 16.6

[MS92] A. Middeldorp and M. Starčević. A rewrite approach to polynomial ideal theory. Internal report, Vrieje Universiteit, Amsterdam, 1992. 24.3, 24.3, 24.4, 24.4, 24.4, 24.6

[MT91] A. Middeldorp and Y. Toyama. Completeness of combinations of constructor systems. In *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, 1991. also Report CS-R9058, CWI, 1990. 8.1

[Mus80] D. R. Musser. On proving inductive properties of abstract data types. In *Proceedings 7th ACM Symp. on Principles of Programming Languages*, pages 154–162. ACM, 1980. 22.1

[Mza85] J. Mzali. Filtrage associatif, commutatif ou idempotent. In *Matériels et logiciels pour la 5ième génération*, pages 243–258, Paris, 1985. AFCET. 14.1.1

[Mza86] J. Mzali. *Méthodes de filtrage équationnel et de preuve automatique de théorèmes.* Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, 1986. 14.1.1

[Nar96] Paliath Narendran. Solving linear equations over semirings. In Edmund Clarke, editor, *Proceedings of LICS'96*. IEEE Computer Society Press, 1996. 10.1

[New42] M. H. A. Newman. On theories with a combinatorial definition of equivalence. In *Annals of Math*, volume 43, pages 223–243, 1942. 4.1, 4.6.2

[Nip88] T. Nipkow. Unification in primal algebras. In M. Dauchet and M. Nivat, editors, *Proceedings of the 13th Colloquium on Trees in Algebra and Programming*, volume 299 of *Lecture Notes in Computer Science*, pages 117–131, Nancy (France), 1988. Springer-Verlag. 2.4.7

[Nip90a] T. Nipkow. Proof transformations for equational theories. In *Proceedings 5th IEEE Symposium on Logic in Computer Science, Philadelphia (Pa., USA)*, pages 278–288, June 1990. 12.3.4, 12.5

[Nip90b] T. Nipkow. Unification in primal algebras, their powers and their varieties. *Journal of the ACM*, 37(1):742–776, October 1990. 13.2.1, 13.6, 13.2.5, 13.2.5

[NO87] M. Navarro and F. Orejas. Parameterized Horn clause specifications: proof theory and correctness. In *Proceedings TAPSOFT Conference*, volume 249 of *Lecture Notes in Computer Science*. Springer-Verlag, 1987. 20.1, 23.3.2, 23.4

[NO90] P. Narendran and F. Otto. Some results on equational unification. In M. E. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, volume 449 of *Lecture Notes in Computer Science*, pages 276–291, July 1990. 10.3

[NOR85] P. Narendran, C. O'Dúlaing, and H. Rolletschek. Complexity of certain decision problems about congruential languages. *Journal of Computer and System Sciences*, 30:343–358, 1985. 2.4.8

[NR91a] P. Narendran and M. Rusinowitch. Any Gound Associative-Commutative Theory Has a Finite Canonical System. In Book [Boo91], pages 423–434. 18.7

[NR91b] P. Narendran and M. Rusinowitch. Any ground associative-commutative theory has a finite canonical system. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*. Springer-Verlag, 1991. 7.3.3, 19.2

[NR92a] R. Nieuwenhuis and A. Rubio. Basic superposition is complete. In B. Krieg-Brückner, editor, *Proceedings of ESOP'92*, volume 582 of *Lecture Notes in Computer Science*, pages 371–389. Springer-Verlag, 1992. 21.1, 21.1

[NR92b] R. Nieuwenhuis and A. Rubio. Theorem proving with ordering constrained clauses. In D. Kapur, editor, *Proceedings 11th International Conference on Automated Deduction, Saratoga Springs (N.Y., USA)*, volume 607 of *Lecture Notes in Computer Science*, pages 477–491. Springer-Verlag, 1992. 21.1, 21.3, 21.1

[NR94] R. Nieuwenhuis and A. Rubio. AC-superposition with constraints: no AC-unifiers needed. In A. Bundy, editor, *Proceedings 12th International Conference on Automated Deduction, Nancy (France)*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 545–559. Springer-Verlag, June 1994. 21.4, 21.5

[NRS89] W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *Journal of Symbolic Computation*, 7(3 & 4):295–318, 1989. Special issue on unification. Part one. III, 14.2

[Nut89] W. Nutt. The unification hierarchy is undecidable. In H.-J. Bürckert and W. Nutt, editors, *Proceedings 3rd International Workshop on Unification, Lambrecht (Germany)*, June 1989. 10.4

[NW63] C. St. J. A. Nash-Williams. On well-quasi-ordering finite trees. *Proc. Cambridge Phil. Soc.*, 59(4):833–835, 1963. 6.4.1

[NW83] T. Nipkow and G. Weikum. A decidability result about sufficient completeness of axiomatically specified abstract data types. In *6th GI Conference*, volume 145 of *Lecture Notes in Computer Science*, pages 257–268. Springer-Verlag, 1983. 22.4

[O'D77] M. J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *Lecture Notes in Computer Science*. Springer-Verlag, 1977. 5.2

[O'D85] M. J. O'Donnell. *Equational Logic as a Programming Language*. Foundation of Computing. The MIT press, 1985. 5.6

[Oka89] M. Okada. Strong normalizability for the combined system of the typed Lambda-calculus and an arbitrary convergent term rewrite system. In *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation, Portland (Oregon)*, pages 357–363. ACM Press, July 1989. Report CRIN 89-R-220. 8.6

[Orc90a] I. Orci. A mathematics sampler. Technical Report UMINF-176.90, ISSN-0348-0542, University of Umea, Institute of Information Processing, S-901 87 Umea, Sweden, 1990. 2

[Orc90b] I. Orci. Universal algebra. Technical Report UMINF-180.90, ISSN-0348-0542, University of Umea, Institute of Information Processing, S-901 87 Umea, Sweden, 1990. 2

[Ore87] F. Orejas. A characterization of passing compatibility for parameterized specifications. *Theoretical Computer Science*, pages 205–214, 1987. 23.3.6, 23.3.6

[OS88] A. Ohsuga and K. Sakai. An efficient implementation method of reduction and narrowing in METIS. Technical report, ICOT Research Center Japan, June 1988. 14.2.5

[Pad88] P. Padawitz. The equational theory of parameterized specifications. *Information and Computation*, 76:121–137, 1988. 23.3.2, 23.3.6, 23.5

[Péc81] J.-P. Pécuchet. *Equations avec constantes et algorithme de Makanin*. Thèse de doctorat, Université de Rouen (France), 1981. III, 10.1

[Ped84] J. Pedersen. *Confluence methods and the word problem in universal algebra*. PhD thesis, Emory University, 1984. 7.3.1

[PEE81] U. Pletat, G. Engels, and H. D. Ehrich. Operational semantics of algebraic specifications with conditional axioms. Technical report, universität Dortmund, 1981. 7.1, 7.5

[Pet83] G. Peterson. A technique for establishing completeness results in theorem proving with equality. *SIAM Journal of Computing*, 12(1):82–100, 1983. 17.7

[Pet90] G. E. Peterson. Complete sets of reductions with constraints. In M. E. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, volume 449 of *Lecture Notes in Computer Science*, pages 381–395. Springer-Verlag, 1990. 21.1, 21.4.1, 21.3, 21.12

[Pfe88] F. Pfenning. Partial polymorphic type inference and higher-order unification. In *Proceeding of the 1988 ACM Conference on Lisp and Functional Programming*. ACM, July 1988. III

[Pie71] T. Pietrzykowski. A complete mechanization of second-order logic. Research report CSSR 2038, Dept. of Appl. Anal. and Comp. Sci., University of Waterloo, 1971. III

[Pla85] D. Plaisted. Semantic confluence and completion method. *Information and Control*, 65:182–215, 1985. 22.3

[Pla93] D. Plaisted. Equational reasoning and term rewriting systems. In D. Gabbay, C. Hogger, J. A. Robinson, and J. Siekmann, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1, pages 273–364. Oxford University Press, 1993. 1, 10.1

[Plo72] G. Plotkin. Building-in equational theories. *Machine Intelligence*, 7:73–90, 1972. III, 10.1, 10.2.1, 10.2.3, 10.4, 10.1

[Poi86] A. Poigné. Parameterisation for order-sorted algebraic specifications. Technical report, Dept. of Computing, Imperial College, London, 1986. 23.3.2

[Pot81] G. Pottinger. The church-rosser theorem for the typed lambda-calculus with surjective pairing. *Notre Dame Journal of Formal Logic*, 22(3):264–268, 1981. 4.6.3

[Pot91] L. Pottier. Minimal solutions of linear diophantine systems: Bounds and algorithms. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 162–173. Springer-Verlag, April 1991. 13.1.5

[PS81] Gerald Peterson and Mark Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28:233–264, 1981. 7.1, 7.3.1, 7.9, 18.1, 18.2.3, 18.3, 18.3.3, 18.7

[PW78] M. S. Paterson and M. N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16:158–167, 1978. 3.2.5, 2, 3.2.5, III, 10.1

[Qui52] W. V. Quine. The problem of simplifying truth functions. *American Math. Monthly*, 59:521–531, 1952. 7.9

[Qui59] W. V. Quine. On cores and prime implicants of truth functions. *American Math. Monthly*, 66:755–760, 1959. 7.9

[Rao81] J.-C. Raoult. Finiteness results in term rewriting systems. *RAIRO Informatique Théorique et applications*, 4:373–391, 1981. 2.4.8

[Rau90] A. Rauzy. Boolean unification: an efficient algorithm. Technical report, LABRI, University of Bordeaux 1, 1990. 13.2.1

[RB85] D. E. Rydeheard and R. M. Burstall. The unification of terms: A category-theoretic algorithm. Technical report, Department of Computer Science University of Manchester, 1985. 10.1

[RB86] D. E. Rydeheard and R. M. Burstall. A categorical unification algorithm. In *Proceedings of the Workshop on Category Theory and Computer Programming*, volume 240 of *Lecture Notes in Computer Science*. Springer-Verlag, 1986. 10.1

[Red85] U. S. Reddy. Narrowing as the operational semantics of functional languages. In *Proceedings of the IEEE Symposium on Logic Programming*, pages 138–151, Salt Lake City (Utah), July 1985. 14.2.5

[Red90] U. S. Reddy. Term rewriting induction. In M. E. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, volume 449 of *Lecture Notes in Computer Science*, pages 162–177. Springer-Verlag, 1990. 22.1, 22.7, 22.7.1, 22.7.2, 23.4

[Rém82] Jean-Luc Rémy. *Etude des systèmes de Réécriture Conditionnels et Applications aux Types Abstraits Algébriques.* Thèse de Doctorat d'Etat, Institut National Polytechnique de Lorraine, Nancy (France), 1982. 2.6.3, 7.1, 7.5, 22.1

[Rét87] P. Réty. Improving basic narrowing. In P. Lescanne, editor, *Proceedings 2nd Conference on Rewriting Techniques and Applications, Bordeaux (France)*, volume 256 of *Lecture Notes in Computer Science*, pages 228–241, Bordeaux (France), May 1987. Springer-Verlag. 14.2

[Rin90] Ch. Ringeissen. Etude et implantation d'un algorithme d'unification dans les algèbres finies. Rapport de DEA, Université Henri Poincaré – Nancy 1, 1990. 13.2.1

[Rin92] Ch. Ringeissen. Unification in a combination of equational theories with shared constants and its application to primal algebras. In *Proceedings of the 1st International Conference on Logic Programming and Automated Reasoning, St. Petersburg (Russia)*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 261–272. Springer-Verlag, 1992. 11.3, 11.4

[Rin93] Ch. Ringeissen. *Combinaison de Résolutions de Contraintes.* Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, December 1993. 11.3, 11.4

[RKKL85] P. Réty, Claude Kirchner, Hélène Kirchner, and P. Lescanne. Narrower: A new algorithm for unification and its application to logic programming. In J.-P. Jouannaud, editor, *Proceedings 1st Conference on Rewriting Techniques and Applications, Dijon (France)*, volume 202 of *Lecture Notes in Computer Science*, pages 141–157. Springer-Verlag, 1985. 14.2, 14.2.5

[RN93] A. Rubio and R. Nieuwenhuis. A precedence-based total ac-compatible ordering. In C. Kirchner, editor, *Proceedings 5th Conference on Rewriting Techniques and Applications, Montreal (Canada)*, volume 690 of *Lecture Notes in Computer Science*, pages 374–388. Springer-Verlag, 1993. 7.3.3, 7.3.3, 19.2

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965. III, 10.1, 17.7

[Rob71] J. A. Robinson. Computational logic: The unification computation. *Machine Intelligence*, 6:63–72, 1971. 10.1

[Rom88] J.-F. Romeuf. Solutions of a linear diophantine system, November 1988. LIR & Université de Rouen. 13.1.5

[Ros73] B. K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, 1973. 8.5.1, 8.1

[RP89] P. Ružička and I Privara. An almost linear robinson unification algorithm. *Acta Informatica*, 27:61–71, 1989. 3.2.5

[RS78] P. Raulefs and J. Siekmann. Unification of idempotent functions. Technical report, Institut für Informatik I, Universität Karlsruhe, 1978. 10.1

[RS86] D. E. Rydeheard and J. Stell. A categorical approach to solving equations. Draft Notes, 1986. 10.1

[RT90] O. Ridoux and H. Tonneau. Une mise en œuvre de l'unification d'expressions booléennes. In *Actes de SPLT'90, Trégastel*. CNET, 1990. 13.2.1

[Rus87a] M. Rusinowitch. *Démonstration automatique par des techniques de réécriture*. Thèse de Doctorat d'Etat, Université Henri Poincaré – Nancy 1, 1987. Also published by InterEditions, Collection Science Informatique, directed by G. Huet, 1989. 17.2, 17.3, 17.7, 17.7

[Rus87b] M. Rusinowitch. On termination of the direct sum of term rewriting systems. *Information Processing Letters*, 26(2):65–70, 1987. 8.1, 8.3.2

[Rus88] M. Rusinowitch. Theorem-proving with resolution and superposition: an extension of Knuth and Bendix procedure to a complete set of inference rules. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1988. See also the extended version published in Journal of Symbolic Computation, number 1&2, 1991. 17.7

[RV80] J.-C. Raoult and J. Vuillemin. Operational and semantic equivalence between recursive programs. *Journal of the ACM*, 27(4):772–796, 1980. 8.6, 1

[RW69] G. A. Robinson and L. T. Wos. Paramodulation and first-order theorem proving. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence 4*, pages 135–150. Edinburgh University Press, 1969. 17.7

[RZ84] Jean-Luc Rémy and H. Zhang. Reveur4 : a system for validating conditional algebraic specifications of abstract data types. In T. O'Shea, editor, *Proceedings of the 5th European Conference on Artificial Intelligence*, Pisa, Italy, 1984. ECAI, Elsevier Science Publishers B. V. (North-Holland). 23.4

[SA91] R. Socher-Ambrosius. Boolean Algebra Admits No Convergent Term Rewriting System. In Book [Boo91], pages 264–274. 7.9

[SAK89] G. Smolka and H. Aït-Kaci. Inheritance hierarchies: Semantics and unification. *Journal of Symbolic Computation*, 7(3 & 4):343–370, 1989. Special issue on unification. Part one. III

[Sal92] G. Salzer. The unification of infinite sets of terms and its applications. In A. Voronkov, editor, *Proceedings of the 1st International Conference on Logic Programming and Automated Reasoning, St. Petersburg (Russia)*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 409–420. Springer-Verlag, July 1992. 16.6

[SCL70] J. R. Slagle, C. L. Chang, and R. C. T. Lee. A new algorithm for generating prime implicants. *IEEE Transactions on Computing*, 19(4):304–310, 1970. 7.9

[Sel72] A. Selman. Completeness of calculii for axiomatically defined classes of algebras. *Algebra Universalis*, 2:20–32, 1972. 2.6.3

[SG89] W. Snyder and J. Gallier. Higher order unification revisited: Complete sets of tranformations. *Journal of Symbolic Computation*, 8(1 & 2):101–140, 1989. Special issue on unification. Part two. III

[Sie75] J. Siekmann. String-unification. Internal report memo CSM-7, University of Essex, 1975. 10.1

[Sie79] J. Siekmann. Unification of commutative terms. In *Proceedings of the Conference on Symbolic and Algebraic Manipulation*, volume 72 of *Lecture Notes in Computer Science*, pages 531–545, Marseille (France), June 1979. Springer-Verlag. Also internal report SEKI, 1976. 10.1, 10.5.5, 12.7

[Sie89] J. Siekmann. Unification theory. *Journal of Symbolic Computation*, 7(3 & 4):207–274, 1989. Special issue on unification. Part one. III

[SK92] A. Sattler-Klein. Infinite, canonical string rewriting systems generated by completion. In A. Voronkov, editor, *Proceedings of the 1st International Conference on Logic Programming and Automated Reasoning, St. Petersburg (Russia)*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 433–444. Springer-Verlag, July 1992. 16.6

[SL90] W. Snyder and C. Lynch. A note on the completeness of sld-resolution. submitted, 1990. 2.3.3

[Smo89] G. Smolka. *Logic Programming over Polymorphically Order-Sorted Types.* PhD thesis, FB Informatik, Universität Kaiserslautern, Germany, 1989. 7.6.1

[Sny88] W. Snyder. *Complete sets of transformations for general unification.* PhD thesis, University of Pennsylvania, 1988. 14.1.1

[Sny89] W. Snyder. Efficient ground completion: An $O(n\ log\ n)$ algorithm for generating reduced sets of ground rewrite rules equivalent to a set of ground equations E. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 419–433. Springer-Verlag, April 1989. 16.5

[SS82a] J. Siekmann and P. Szabó. A noetherian and confluent rewrite system for idempotent semigroups. *Semigroup Forum*, 25:83–110, 1982. 7.6.4

[SS82b] J. Siekmann and P. Szabó. Universal unification and classification of equational theories. In *Proceedings 6th International Conference on Automated Deduction, New York (N.Y., USA)*, volume 138 of *Lecture Notes in Computer Science*. Springer-Verlag, 1982. III, 10.4, 10.1

[SS84] J. Siekmann and P. Szabó. Universal unification. In R. Shostak, editor, *Proceedings 7th International Conference on Automated Deduction, Napa Valley (Calif., USA)*, volume 170 of *Lecture Notes in Computer Science*, pages 1–42, Napa Valley (California, USA), 1984. Springer-Verlag. 10.4

[SS86a] M. Schmidt-Schauß. Unification in many sorted equational theories. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*. Springer-Verlag, 1986. III

[SS86b] M. Schmidt-Schauß. Unification under associativity and idempotence is of type nullary. *Journal of Automated Reasoning*, 2(3):277–282, 1986. 10.2.3, 10.1

[SS88] K. Sakai and Y. Sato. Boolean gröbner bases. Technical report, ICOT Research Center Japan, June 1988. 24.7

[SS89] M. Schmidt-Schauß. Unification in a Combination of Arbitrary Disjoint Equational Theories. *Journal of Symbolic Computation*, 8(1 & 2):51–99, 1989. III, 11

[SS90a] K. Sakai and Y. Sato. Application of the ideal theory to boolean constraint solving. In *Proc. Pacific Rim International Conference on Artificial Intelligence*, pages 490–495, 1990. 24.7

[SS90b] M. Schmidt-Schauß. Unification in permutative equational theories is undecidable. In Claude Kirchner, editor, *Unification*, pages 117–124. Academic Press inc., London, 1990. 10.3

[SS98] Manfred Schmidt-Schauss. A decision algorithm for distributive unification. *Theoretical Computer Science*, 208(xxx):xxx, 1998. III

[SSS90] K. Sakai, Y. Sato, and Menju S. Solving constraints over sets by boolean Gröbner bases. Internal report, ICOT Research center, 1990. 24.7

[Sta75] J. Staples. Church-Rosser theorems for replacement systems. *Algebra and Logic, Lectures Notes in Mathematics*, 450:291–307, 1975. 8.5.1

[Sti75] M. E. Stickel. A complete unification algorithm for associative-commutative functions. In *Proceedings 4th International Joint Conference on Artificial Intelligence, Tbilissi (USSR)*, pages 71–76, 1975. 10.1

[Sti76] M. E. Stickel. *Unification Algorithms for Artificial Intelligence Languages*. PhD thesis, Carnegie-Mellon University, 1976. 10.1, 13.1.1

[Sti81] M. E. Stickel. A unification algorithm for associative-commutative functions. *Journal of the ACM*, 28:423–434, 1981. III, 10.1, 13.1.1

[Sti84] M. E. Stickel. A case study of theorem proving by the Knuth-Bendix method: Discovering that $x^3 = x$ implies ring commutativity. In R. Shostak, editor, *Proceedings 7th International Conference on Automated Deduction, Napa Valley (Calif., USA)*, volume 170 of *Lecture Notes in Computer Science*, pages 248–258. Springer-Verlag, 1984. 19.2

[Sza82] P. Szabó. *Unifikationstheorie erster Ordnung*. PhD thesis, Universität Karlsruhe, 1982. 2.4.8, III, 10.3, 10.4, 10.1

[“T02] “Terese” (M. Bezem, J. W. Klop and R. de Vrijer, eds.). *Term Rewriting Systems*. Cambridge University Press, 2002. 1, 6.4.3

[TA87] E. Tiden and S. Arnborg. Unification problems with one-sided distributivity. *Journal of Symbolic Computation*, 3(1 & 2):183–202, April 1987. III, 10.3, 10.3, 10.1

[Tar68] A. Tarski. Equational logic and equational theories of algebras. In K. Schütte, editor, *Contributions to Mathematical Logic*, pages 275–288. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1968. 22.3

[Tay79] W. Taylor. Equational logic. *Houston Journal of Mathematics*, 1979. Appears also in [Grä79], Appendix 4. 22.3, 2

[Tid86] E. Tidén. Unification in combinations of collapse-free theories with disjoint sets of functions symbols. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 431–449. Springer-Verlag, 1986. 11

[Tis89] S. Tison. Fair termination is decidable for ground systems. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 462–476. Springer-Verlag, April 1989. 6.2

[TKB89] Y. Toyama, J. W. Klop, and H. P. Barendregt. Termination for the direct sum of left-linear term rewriting systems. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 477–491. Springer-Verlag, April 1989. 8.7

[Toy86] Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, May 1986. 6.1, 8.1, 8.1

[Toy87] Y. Toyama. On the church-rosser property for the direct sum of term rewritig systens. *Journal of the ACM*, 34(1):128–143, January 1987. 8.1, 8.1

[Toy88] Y. Toyama. Commutativity of term rewriting systems. In K. Fuchi and L. Kott, editors, *Programming of Future Generation Computers II*, pages 393–407. Elsevier Science Publishers B. V. (North-Holland), 1988. 8.6

[Ver81] R. L. Veroff. Canonicalization and demodulation. Internal Report ANL-81-6, Argonne National Laboratory, Argonne,IL, 1981. 19.2

[Vig93] L. Vigneron. Associative-commutative deduction with constraints. Technical Report 93-R-196, CRIN, 1993. 21.4, 21.5

[Vog78] E. Vogel. Morphismenunifikation. Technical report, Universität Karlsruhe, 1978. Diplomarbeit. 10.1

[Wal84] C. Walther. Unification in many sorted theories. In T. O'Shea, editor, *Proceedings of the European Conference on Artificial Intelligence, Pisa, Italy*, pages 593–602. ECAI, 1984. III

[WB83] F. Winkler and B. Buchberger. A criterion for eliminating unnecessary reductions in the Knuth-Bendix algorithm. In *Processing of the Colloquium on Algebra, Combinatorics and Logic in Computer Science*, Györ, Hungary, 1983. 4.2

[Wec92] Wolfgang Wechler. *Universal Algebra for Computer Scientists*, volume 25 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1992. 2

[Wer94] Andreas Werner. Normalizing narrowing for weakly terminating and confluent systems. Technical report, Karlruhe University, October 1994. 14.2.4

[Whi41] Ph.M. Whitman. Free lattices. *Annals of Mathematics*, 42(2):325–330, 1941. 18.8

[Whi42] Ph.M. Whitman. Free lattices II. *Annals of Mathematics*, 43(2):104–115, 1942. 18.8

[Win83] F. Winkler. A criterion for eliminating unnecessary reductions in the Knuth-Bendix algorithm. Technical report, Universität Linz, Austria, 1983. 19.3

[Win89] Franz Winkler. Knuth-Bendix procedure and Buchberger algorithm - A synthesis. In *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, pages 55–67, Portland (Oregon, USA), 1989. ACM Press. 18.8, 24.6

[Wos88] L. Wos. *Automated Reasoning: 33 basic research problems*. Prentice Hall, Inc., Englewood Cliffs, NJ, 1988. 19.2

[Yel87] K. Yelick. Unification in combinations of collapse-free regular theories. *Journal of Symbolic Computation*, 3(1 & 2):153–182, April 1987. III, 11, 11.2.2

[You89] J.-H. You. Enumerating outer narrowing derivations for constructor-based term rewriting systems. *Journal of Symbolic Computation*, 7(3 & 4):319–342, 1989. Special issue on unification. Part one. 14.2, 14.2.5

[Zan92] H[ans] Zantema. Termination of term rewriting by interpretation. In Michaël Rusinowitch and Jean-Luc Rémy, editors, *Conditional Term Rewriting Systems, Third International Workshop*, LNCS 656, pages 155–167, Pont-à-Mousson, France, July 8–10, 1992. Springer-Verlag. Published in 1993. 6.3

[Zha92] H. Zhang. A linear robinson unification algorithm. Technical report, The University of IOWA, Iowa City, July 1992. 3.2.5

[ZK89] H. Zhang and D. Kapur. Consider only general superpositions in completion procedures. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 513–527. Springer-Verlag, April 1989. 19.3, 19.3

[ZKK88] H. Zhang, D. Kapur, and M. S. Krishnamoorthy. A mechanizable induction principle for equational specifications. In E. Lusk and R. Overbeek, editors, *Proceedings 9th International Conference on Automated Deduction, Argonne (Ill., USA)*, volume 310 of *Lecture Notes in Computer Science*, pages 162–181. Springer-Verlag, 1988. 22.7.1

[ZR85] H. T. Zhang and Jean-Luc Rémy. Contextual rewriting. In J.-P. Jouannaud, editor, *Proceedings 1st Conference on Rewriting Techniques and Applications, Dijon (France)*, volume 202 of *Lecture Notes in Computer Science*, pages 46–62, Dijon (France), 1985. Springer-Verlag. 7.1, 7.5