ISR 2014
Strategies

Hélène KIRCHNER
Inria

August 2014

Topics, Objectives, Contents

Computation, Deduction and Strategies

Series of workshops since 1997

Strategies CADE-IJCAR *Reduction Strategies RTA-RDP*

Strategies in Rewriting, Proving, and Programming FLoC

This lecture is based on joint work with many people, in particular the members of the PROTHEO and the PORGY teams. Thanks to all of them!

Rules... and Strategies

Rules describe local transformations

- **Rules for computations:** unique normal form required
  the strategy is fixed
- **Rules for deductions:** no confluence nor termination required
  an application strategy is required

Strategies describe the control of rule application

Derivation tree exploration:
strategies are needed to express choices
Strategies describe selected computations

Rules... and Strategies

Rules describe local transformations

- **Rules for computations:** unique normal form required
  the strategy is fixed
- **Rules for deductions:** no confluence nor termination required
  an application strategy is required

Strategies describe the control of rule application

Derivation tree exploration:
strategies are needed to express choices
Strategies describe selected computations

Strategies are ALWAYS needed

1- In Functional or Logic Programming, Theorem Proving, Constraint Solving, Formal Specifications, ...

2- To describe the way computation or deduction should be done

    Lazy evaluation
    Search plans
    Action plans
    Tactics
    Priorities ...

3- This requires in general to *search* for a particular derivation corresponding to the desired strategy.

Example - HO rewriting

A non-deterministic strategy for higher-order rewriting: choose an outermost redex and skip redexes that do no contribute to the normal form because they are in a cycle [Klop,vOostrom,vRaamsdonk07]

Example - in theorem proving

Given two tactics A and B, apply tactic B only if the application of tactic A has either failed or did not modify the proof (definition of `orelse` in LCF)

Example - constraint programming

/* Strategy : Forward Checking with Choice Point */
/* Value selection : Value enumeration first to last */
/* Number of solutions : All */
("FCChoicePointFirstToLastAll" [CastroPhd])

Example - program transformation strategies

In program transformation [VisserJSC05], a strategy can be provided by a transformation engine or can be user-definable.
Transformation strategies are the control part of transformation systems that determine the order of application of basic transformation steps.

Example - in game theory

Two players W and B with respective rules R_W and R_B play a game by rewriting terms in the combined signature. Is there a winning strategy to reach the normal form? [Dougherty-WRS09]

Our approach

We are at International School on Rewriting, so we adopt a rewriting point of view !

This is a meaningful “parti-pris” !

Rules and strategies provide powerful formalism to express and study uniformly computations and deductions in automated deduction and reasoning systems.

Strategic rewriting and strategic programs

Objectives of the lecture

1. Define strategies, strategic rewriting, strategic programs
2. Define semantics of strategies and strategic programs
3. Provide examples of strategy languages and how to write strategies
4. Explore some properties of strategic programs
5. Identify research topics
6. Provide bibliography

Contents of the whole lecture

1. Objectives and contents
2. What are strategic rewriting and strategic programming?
   - Rewriting point of view: different uses of rules
   - Strategic point of view: different uses of strategies
   - Focus on term rewriting strategies
3. Strategy Semantics: different points of view
   - Rewriting logic
   - Rewriting calculus
   - Abstract reduction systems
   - Properties of strategic rewriting
4. Strategy languages
   - Examples of languages
   - Common constructs in strategy languages
   - Operational semantics of strategic programs
   - PORGY
5. Further work: verification techniques,...

ISR 2014
Strategies

Hélène KIRCHNER
Inria

August 2014

Which is the rewriting point of view?

Ingredients of rewriting

The syntactic structure

Words, Terms, Propositions, Logic formulas, Dags, Graphs, Structured Objects, Segments ...

The pattern : rule

Expressed with $\Rightarrow$, variables, left-hand side, right-hand side, condition or constraint

The application mode

- match to select a redex (possibly modulo some axioms, constraints,...)
- instantiate variables
- replace

Formally

t rewrites to t' using the rule $\ell : l \Rightarrow r$ if

$$t_{|p} = \sigma(l) \quad \text{and} \quad t' = t[\sigma(r)]_p$$

This is denoted

$$t \longrightarrow_{p,\ell,\sigma} t'$$

The choices: position(s), rule, matching substitution(s).
Rewriting may be concurrent, probabilistic, modulo...

Matching modulo associativity-commutativity

$\cup$ is assumed to be an associative commutative (AC) symbol:
$\forall x, y, z, \;\; x \cup (y \cup z) = (x \cup y) \cup z$ and $\forall x, y, \; x \cup y = y \cup x$.

$$\{i\} \cup s \Rightarrow i$$

$$\{i\} \cup s \ll_{AC} \{1\} \cup \{2\} \cup \{3\} \cup \{4\} \cup \{5\}$$

$$\begin{array}{ll} \{1\} \cup \{2\} \cup \{3\} \cup \{4\} \cup \{5\} & =_{AC} \\ \{2\} \cup \{3\} \cup \{4\} \cup \{5\} \cup \{1\} & =_{AC} \\ \ldots & \\ \{5\} \cup \{1\} \cup \{2\} \cup \{3\} \cup \{4\} & \end{array}$$

5 different and non *AC*-equivalent matches.
The rewrite rule applies in 5 different ways and gives 5 different results : $1, 2, 3, 4, 5$.

Matching modulo associativity-commutativity

$\cup$ is assumed to be an associative commutative (AC) symbol:
$\forall x, y, z, \;\; x \cup (y \cup z) = (x \cup y) \cup z$ and $\forall x, y, \, x \cup y = y \cup x$.

$$\{i\} \cup s \Rightarrow i$$

$$\{i\} \cup s \ll_{AC} \{1\} \cup \{2\} \cup \{3\} \cup \{4\} \cup \{5\}$$

$$\begin{array}{ll} \{1\} \cup \{2\} \cup \{3\} \cup \{4\} \cup \{5\} & =_{AC} \\ \{2\} \cup \{3\} \cup \{4\} \cup \{5\} \cup \{1\} & =_{AC} \\ \ldots & \\ \{5\} \cup \{1\} \cup \{2\} \cup \{3\} \cup \{4\} & \end{array}$$

5 different and non AC-equivalent matches.
The rewrite rule applies in 5 different ways and gives 5 different results : $1, 2, 3, 4, 5$.

Example: Sorting by rewriting

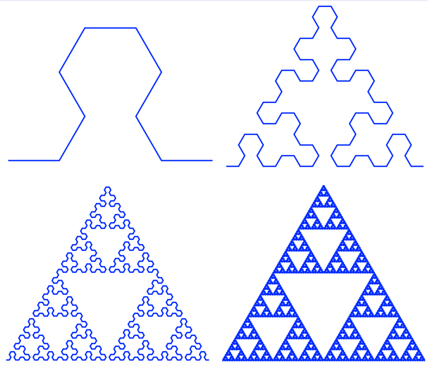
```
rules for List
  X, Y : Nat ; L L' L'' : List;
   sort (L X L' Y L'') => sort (L Y L' X L'') if Y < X.
   sort L => L .
end

sort (6 5 4 3 2 1) -> ... -> (1 2 3 4 5 6)

sorts NeList List ;   subsorts Nat < NeList < List ;
operators
  nil : List ;
  @ @ : (List List) List       [associative id: nil] ;
  @ @ : (NeList List) NeList   [associative] ;
  sort @ : (List) List ;
end
```

Example: Lindenmayer's systems

E.g. http://en.wikipedia.org/wiki/L-system: The Sierpinski triangle



```
start: A
rules: A=>B-A-B, B=>A+B+A
angle: 60
```

A $\rightarrow$

B−A−B $\rightarrow$

A+B+A−B−A−B−A+B+A $\rightarrow$

...

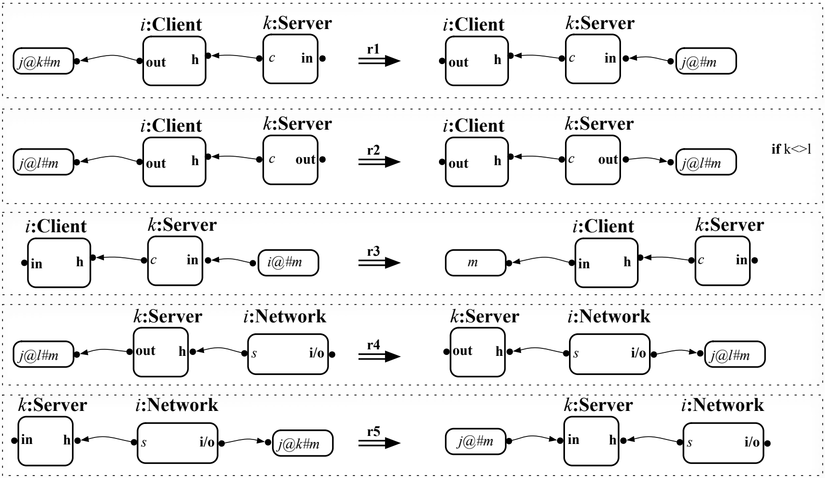
Example: Program Transformation

Refactoring rules in Stratego

```
rules
 InlineF:
    let f(xs) = e in e'[f(es)] =>
    let f(xs) = e in e'[e[es/xs]]
 InlineV:
    let x = e in e'[x] => let x = e in e'[e]
 Dead:
    let x = e in e' => e' where <not(in)> (x,e')
 Extract(f,xs):
    e => let f(xs) = e in f(xs)
 Hoist:
    let x = e1 in let f(xs) = e2 in e3 =>
    let f(xs) = e2 in let x = e1 in e3
    where <not(in)> (x, <free-vars> e2)
```

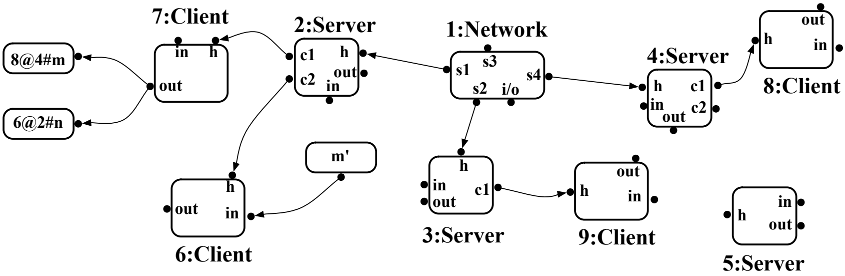
Biochemical rules

Basic rules for the mail delivery system



Example: Biochemical program

A mail system configuration



ISR 2014
Strategies

Hélène KIRCHNER
Inria

August 2014

Which is the strategic point of view?

Rewrite rules and Strategies

Rewrite rules describe local transformations

Rewrite derivations are computations
Normal forms are the results

Strategies describe the control of rewrite rule application

Strategic rewriting derivations are selected computations

But the strategy is often implicite in algebraic languages: ASF+SDF, OBJ, Maude,...
and in functional languages: ML, Haskell,...

Derivation tree

Given a set of rewrite rules $\mathcal{R}$, a *derivation*, or computation from G is a sequence of rewriting steps

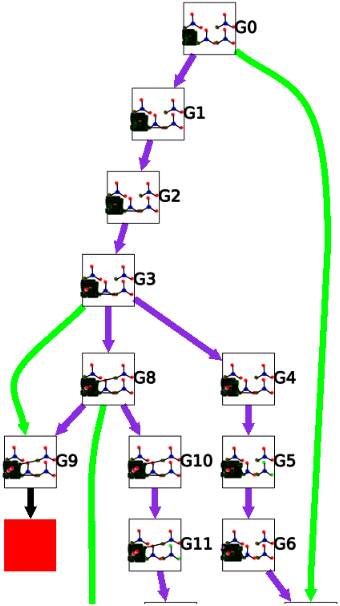
$$G \to_{\mathcal{R}} G' \to_{\mathcal{R}} G'' \to_{\mathcal{R}} \dots$$

The *derivation tree* of G, written $DT(G, \mathcal{R})$, is a labelled tree
- whose root is labelled by G,
- its children are all the derivation trees $DT(G_i, \mathcal{R})$ such that $G \to_{\mathcal{R}} G_i$.

The edges of the derivation tree are labelled with the rewrite rule and the morphism used in the corresponding rewrite step.
A derivation tree may be infinite, if there is an infinite reduction sequence out of G.

A derivation tree

$$
\begin{tikzcd}
 & G_1^1 & & & \\
 & G_1^2 \arrow[r, "\phi_1^2"] & G_2^2 & \cdots & \\
G_0^0 \arrow[uur, "\phi_0^1"] \arrow[ur, "\phi_0^2"] \arrow[dr, "\phi_0^n"] & \vdots & & & \\
 & G_1^n \arrow[r, "\phi_1^n"] & \cdots \arrow[r, "\phi_{n-2}^n"] & G_{n-1}^n \arrow[r, "\phi_{n-1}^n"] & G_n^n
\end{tikzcd}
$$


G0
G1
G2
G3
G8
G4
G9
G10
G5
G11
G6

Strategic programming

Intuitively,

- a strategic program consists of an initial structure G (or t when it is a term), together with a set of rules $\mathcal{R}$ that will be used to reduce it, following the given strategy expression S.
- in general, there is more than one way of rewriting a structure, so the set of rewrite derivations can be organised as a derivation tree. We need to identify the branches in G's derivation tree that satisfy the strategy S.
- the strategy expression S is used to decide which rewrite steps should be performed on G.

Strategic programming

A *strategic rewrite program* consists of a finite set of rewrite rules $\mathcal{R}$, a strategy expression S (built from $\mathcal{R}$ using a strategy language) and a given structure G.

We denote it $[S_{\mathcal{R}}, G]$, or simply $[S, G]$ when $\mathcal{R}$ is clear from the context.

ISR 2014
Strategies

Hélène KIRCHNER
Inria

August 2014

Focus on term rewriting strategies

Term rewriting (positional) strategies

In first-order term rewriting,

- Terms may have normal forms or admit infinite reductions. Rewriting strategies are efficient ways to compute normal forms.
- Strategies are used to determine at each step of a derivation which is the next redex. Rather than exploring all possible rewrite sequences from a given term, a rewrite strategy dictates which sequences must be computed.

Which (computable) strategies are guaranteed to find a normal form for any term whenever it exists?

[Terese]

Rewriting strategies on terms

- Innermost and outermost reduction
- Needed and standard reduction
- Reduction under local strategies
- Context-sensitive reduction

Definitions

A *strategic term rewriting reduction* is a relation $\xrightarrow{S}$ such that $\xrightarrow{S} \subseteq \xrightarrow{+}_{\mathcal{R}}$
and $NF(\xrightarrow{S}) = NF(\mathcal{R})$ (additional hypothesis for positional term rewriting strategies).

- A strategic term rewriting reduction *normalizes the term* t if there is no infinite $\xrightarrow{S}$-rewrite sequence starting from t.
- A strategic rewriting reduction is *normalizing* or *complete* if it normalizes every term that has a $\mathcal{R}$-normal form

Maximal strategies

A ==*maximal multi-step relation*== $\xrightarrow{\parallel}$ is inductively defined as follows:

1. $x \xrightarrow{\parallel} x$ for all variables x
2. $f(s_1, ..., s_n) \xrightarrow{\parallel} f(t_1, ..., t_n)$ if $s_i \xrightarrow{\parallel} t_i, \forall i, 1 \leq i \leq n$ and $f(s_1, ..., s_n)$ is no redex
3. $\sigma(l) \xrightarrow{\parallel} \tau(r)$ if $l \to r \in \mathcal{R}$ and $\sigma \xrightarrow{\parallel} \tau$

Innermost and outermost reduction

$$\mathcal{R} = \{f(x, b) \Rightarrow b; a \Rightarrow b; c \Rightarrow c\}$$

- leftmost outermost $f(\mathbf{c}, a) \longrightarrow f(c, a)$
- leftmost innermost $f(\mathbf{c}, a) \longrightarrow f(c, a)$
- maximal outermost $f(\mathbf{c}, \mathbf{a}) \xrightarrow{\|} f(c, b)$
- maximal innermost $f(\mathbf{c}, \mathbf{a}) \xrightarrow{\|} f(c, b)$

Innermost and outermost reduction

Reminder - definitions

- orthogonal: left-linear and no critical pair
- overlay: no critical pairs with respect to a non-root position
- left-normal:
  A term t is left-normal if $q \in Pos_V(t)$ for all positions $p, q \in Pos(t)$ such that $p \in Pos_V(t)$ and $p <_{left} q$
  $\mathcal{R}$ is left-normal if all left-hand sides of rules in $\mathcal{R}$ are left-normal.

Exercise : Combinatory Logic is left-normal and orthogonal

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

Innermost and outermost reduction

(see [Terese] and A. Middledorp's slides)

- Leftmost outermost strategy is normalizing for orthogonal left-normal TRS
- Leftmost outermost strategy is not normalizing in general ([HuetLevy1991])

$$\mathcal{R} = \{f(x, b) \Rightarrow b; a \Rightarrow b; c \Rightarrow c\}$$

  Exercise : $f(c, a)$ has a normal form which is not found by the leftmost outermost strategy.
- Innermost strategy is complete for terminating and non ambiguous TRS.
- Innermost strategy is complete for terminating, right-linear and overlay TRS [Okamoto&all2003]

Needed reduction

Needed reduction is interesting for orthogonal term rewriting systems: combinatory logic, λ-calculus, functional programming ...

Intuition: reductions to normal form contract same redexes (only order of contraction differs). A needed strategy performs needed steps: do not contract other redexes to reach result

Let $\mathcal{R}$ be a TRS. Let $\bullet$ be a fresh constant and consider the extension $\mathcal{R}^\bullet = \mathcal{R} \cup \{\bullet \to \bullet\}$. A redex Δ in a term $t = C[\Delta]$ is needed if $t = C[\bullet]$ has no normal form in $\mathcal{R}^\bullet$.

Needed reduction

Let $\mathcal{R}$ be an orthogonal TRS.

- Every reducible term contains a needed redex.
- Repeated contraction of needed redexes results in a normal form, if the term under consideration has a normal form.

Unfortunately

- Neededness of a redex is not decidable.
  (cf. [Terese, ch.9] for a counter-example)

Needed reduction

Let $\mathcal{R}$ be an orthogonal TRS.

- Every reducible term contains a needed redex.
- Repeated contraction of needed redexes results in a normal form, if the term under consideration has a normal form.

Unfortunately

- Neededness of a redex is not decidable.
  (cf. [Terese, ch.9] for a counter-example)

Needed redexes

But needed redexes can be computed for some classes of rewrite systems:

- In *sequential* TRS, every term which is not in normal form contains a needed redex ([HuetLevy91],[Terese]).
  Strong sequentiality is decidable for left-linear TRS.
- External redexes (outermost until contracted) are needed.

  - Combinatory logic: leftmost-outermost redex external
  - λ-calculus: idem

Reduction under local strategies

Historically, local strategies were used

- in eager languages such as Lisp (lazy cons)
- in the OBJ family of languages (OBJ, CafeOBJ, Maude,...) to guide the evaluation (local strategies of functions)
- in (lazy) functional programming, via different kinds of syntactic annotations on the program (strictness annotations, or global and local annotations).
  Haskell allows for syntactic annotations on the arguments of datatype constructors.

Strategy annotations

Informally, a *strategy annotation* is a list of argument positions and rule names.
The argument positions indicate the next argument to evaluate and the rule names indicate a rule to apply.

The innermost strategy for a function symbol C corresponds to an annotation

$$strat(C) = [1, 2, , .., k, R_1, R_2, ...R_n]$$

that indicates that all its arguments should be evaluated from left to right and that the rules R_i should be tried.

Example

Rewrite system with non-terminating reduction [Visser01]

```
imports integers
signature
  sorts Int
  constructors
    Fac : Int -> Int
If :Bool*Int*Int->Int rules
  Fac : Fac(x) -> If(Eq(x,0), 1, Mul(x,Fac(Sub(x,1))))
  IfT : If(True, x, y) -> x
  IfF : If(False, x, y) -> y
  IfE : If(p, x, x) -> x

strat(If) = [1,IfT,IfF,2,3,IfE]
```

declares that only the first argument should be evaluated before applying rules `IfT` and `IfF`.

Strategy annotation definitions

- A *strategy annotation* for function symbol f is a finite list $A(f)$ containing
  - argument positions of f
  - (labels of) rewrite rules for f.
- A strategy annotation $A(f)$ for function symbol f is *full* if $A(f)$ contains all argument positions of f and all rewrite rules for f.
- A strategy annotation $A(f)$ for function symbol f is *in-time* if argument positions are listed in $A(f)$ before rewrite rules that need them.
- A rewrite rule $f(s_1, ..., s_n) \to t$ needs argument position i if
  - s_i is non-variable
  - s_i is variable that appears in $s_1, ..., s_{i-1}, s_{i+1}, ..., s_n$.

Just-in-time strategies

Strategy annotations for functions:

- For any full and in-time strategy annotation A, any term t, if leftmost innermost strategy normalizes t then the corresponding $\xrightarrow{S_A}$ normalizes t.
- If all argument positions and all rules for a function are mentioned in the annotation, it can be guaranteed that a normal form is reached.

The *just-in-time strategy* is a permutation of argument positions and rules in which rules are applied as early as possible.
See JITty: `http://www.cwi.nl/~vdpol/jitty/`

Context-sensitive reduction

Context-sensitive rewriting (CSR) is a rewriting restriction which can be associated to every term rewriting system (TRS) [Lucas98].

Given a signature $\mathcal{F}$, a mapping $\mu : \mathcal{F} \mapsto \mathcal{P}(N)$, called the *replacement map*, discriminates some argument positions $\mu(f) \subseteq \{1, ..., k\}$ for each k-ary symbol f.
Given a function call $f(t_1, ..., t_k)$, the replacements are allowed on arguments t_i such that $i \in \mu(f)$ and are forbidden for the other argument positions.

Example

[Lucas2001]

$$
\begin{aligned}
&sel(0, x : y) \to x\\
&sel(s(x), y : z) \to sel(x, z)\\
&from(x) \to x : from(s(x))\\
&first(0, x) \to []\\
&first(s(x), y : z) \to y : first(x, z)
\end{aligned}
$$

$$
\begin{aligned}
&\mu(s) = \mu(:) = \mu(from) = 1\\
&\mu(sel) = \mu(first) = \{1, 2\}
\end{aligned}
$$

Context-sensitive rewriting derivation:

$$
sel(s(0), \text{from(s(0)))} \to \text{sel(s(0),s(0):from(s(s(0))))} \to
$$
$$
sel(0, \text{from(s(s(0))))} \to \text{sel(0,s(s(0)):from(s(s(0))))} \to s(s(0))
$$

Infinite derivation avoided:

$$
sel(s(0), \text{from(s(0)))} \to sel(s(0), s(0) : \text{from(s(s(0))))}
$$
$$
\to sel(s(0), s(0) : s(s(0)) : \text{from(s(s(s(0)))))} \to \ldots
$$

Context-sensitive reduction

Sufficient conditions to ensure that CSR is still able to compute head-normal forms and values have been established in [[Lucas98].
The *canonical replacement map* (denoted by μ_{can}) specifies the most restrictive replacement map which can be (automatically) associated to a TRS $\mathcal{R}$ in order to achieve completeness of context-sensitive computations.

- Left-linear, confluent, and μ_{can}-terminating TRS admit a computable normalizing strategy.

Context-sensitive reduction

Exercise : With the replacement map μ in previous Example, show that every term t having a value $s^n(0)$ for some $n \geq 0$ can be evaluated using CSR.
Compare with:

$$\mu_{can}(\mathit{first}) = \mu_{can}(\mathit{sel}) = 1, 2$$
$$\mu_{can}(s) = \mu_{can}(:) = \mu_{can}(\mathit{from}) = \varnothing$$