A rewriting point of view on strategies

Hélène Kirchner
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex France
e-mail:Helene.Kirchner@inria.fr

This paper is an expository contribution reporting on published work. It focusses on an approach followed in the rewriting community to formalize the concept of strategy. Based on rewriting concepts, several definitions of strategy are reviewed and connected: in order to catch the higher-order nature of strategies, a strategy is defined as a proof term expressed in the rewriting logic or in the rewriting calculus; to address in a coherent way deduction and computation, a strategy is seen as a subset of derivations; and to recover the definition of strategy in sequential path-building games or in functional programs, a strategy is considered as a partial function that associates to a reduction-in-progress, the possible next steps in the reduction sequence.

1 Introduction

Strategies frequently occur in automated deduction and reasoning systems and more generally are used to express complex designs for control in modeling, proof search, program transformation, SAT solving or security policies. In these domains, deterministic rule-based computations or deductions are often not sufficient to capture complex computations or proof developments. A formal mechanism is needed, for instance, to sequentialize the search for different solutions, to check context conditions, to request user input to instantiate variables, to process subgoals in a particular order, etc. This is the place where the notion of strategy comes in.

This paper deliberately focusses on an approach followed in the rewriting community to formalize a notion of strategy relying on rewriting logic [17] and rewriting calculus [7] that are powerful formalisms to express and study uniformly computations and deductions in automated deduction and reasoning systems. Briefly speaking, rules describe local transformations and strategies describe the control of rule application. Most often, it is useful to distinguish between rules for computations, where a unique normal form is required and where the strategy is fixed, and rules for deductions, in which case no confluence nor termination is required but an application strategy is necessary. Regarding rewriting as a relation and considering abstract rewrite systems leads to consider derivation tree exploration: derivations are computations and strategies describe selected computations.

Based on rewriting concepts, that are briefly recalled in Section 2, several definitions of strategy are reviewed and connected. In order to catch the higher-order nature of strategies, a strategy is first defined as a proof term expressed in rewriting logic in Section 3 then in rewriting calculus in Section 4. In Section 5, a strategy is seen as a set of paths in a derivation tree; then to recover the definition of strategy in sequential path-building games or in functional programs, a strategy is considered as a partial function that associates to a reduction-in-progress, the

F. Mogavero, A. Murano, and M.Y. Vardi (Eds.):
1st Workshop on Strategic Reasoning 2013 (SR'13)
EPTCS ??, 2013, pp. 1–7, doi:10.4204/EPTCS.??.??

possible next steps in the reduction sequence. In this paper, the goal is to show the progression of ideas and definitions of the concept, as well as their correlations.

2 Rewriting

Since the 80s, many aspects of rewriting have been studied in automated deduction, programming languages, equational theory decidability, program or proof transformation, but also in various domains such as chemical or biological computing, plant growth modelling, etc. In all these applications, rewriting definitions have the same basic ingredients. Rewriting transforms syntactic structures that may be words, terms, propositions, dags, graphs, geometric objects like segments, and in general any kind of structured objects. Transformations are expressed with patterns or rules. Rules are buit on the same syntax but with an additional set of variables, say $\mathcal{X}$, and with a binder $\Rightarrow$, relating the left-hand side and the right-hand side of the rule, and optionally with a condition or constraint that restricts the set of values allowed for the variables. Performing the transformation of a syntactic structure t is applying the rule labelled ℓ on t, which is basically done in three steps: (1) match to select a redex of t at position p denoted $t_{|p}$ (possibly modulo some axioms, constraints,...); (2) instantiate the rule variables by the result(s) of the matching substitution σ; (3) replace the redex by the instantiated right-hand side. Formally: t rewrites to t' using the rule $\ell : l \Rightarrow r$ if $t_{|p} = \sigma(l)$ and $t' = t[\sigma(r)]_p$. This is denoted $t \longrightarrow_{p,\ell,\sigma} t'$.

In this process, there are many possible choices: the rule itself, the position(s) in the structure, the matching substitution(s). For instance, one may choose to apply a rule concurrently at all disjoint positions where it matches, or using matching modulo an equational theory like associativity-commutativity, or also according to some probability.

3 Rewriting logic

The Rewriting Logic is due to J. Meseguer and N. Martí-Oliet [17].
As claimed on `http://wrla2012.lcc.uma.es/`:

Rewriting logic (RL) is a natural model of computation and an expressive semantic framework for concurrency, parallelism, communication, and interaction. It can be used for specifying a wide range of systems and languages in various application fields. It also has good properties as a metalogical framework for representing logics. In recent years, several languages based on RL (ASF+SDF, CafeOBJ, ELAN, Maude) have been designed and implemented.

In Rewriting Logic, the syntax is based on a set of terms $\mathcal{T}(\mathcal{F})$ built with an alphabet $\mathcal{F}$ of function symbols with arities, a theory is given by a set $\mathcal{R}$ of labeled rewrite rules denoted $\ell(x_1,\ldots,x_n) : l \Rightarrow r$, where labels $\ell(x_1,\ldots,x_n)$ record the set of variables occurring in the rewrite rule. Formulas are sequents of the form $\pi : t \to t'$, where π is a *proof term* recording the proof of the sequent: $\mathcal{R} \vdash \pi : t \to t'$ if $\pi : t \to t'$ can be obtained by finite application of equational deduction rules given below. In this context, a proof term π encodes a sequence of rewriting steps called a derivation.

Reflexivity For any $t \in \mathcal{T}(\mathcal{F})$:

$$\mathbf{t} : t \to t$$

Congruence For any $f \in \mathcal{F}$ with $arity(f) = n$:

$$\frac{\pi_\mathbf{1} : t_1 \to t'_1 \quad \ldots \quad \pi_\mathbf{n} : t_n \to t'_n}{\mathbf{f}(\pi_\mathbf{1}, \ldots, \pi_\mathbf{n}) : f(t_1, \ldots, t_n) \to f(t'_1, \ldots, t'_n)}$$

Transitivity

$$\frac{\pi_\mathbf{1} : t_1 \to t_2 \quad \pi_\mathbf{2} : t_2 \to t_3}{\pi_\mathbf{1}; \pi_\mathbf{2} \ : \ t_1 \to t_3}$$

Replacement For any $\ell(x_1, \ldots, x_n) : l \Rightarrow r \in \mathcal{R}$,

$$\frac{\pi_\mathbf{1} : t_1 \to t'_1 \quad \ldots \quad \pi_\mathbf{n} : t_n \to t'_n}{\ell(\pi_\mathbf{1}, \ldots, \pi_\mathbf{n}) : l(t_1, \ldots, t_n) \to r(t'_1, \ldots, t'_n)}$$

The ELAN language, designed in 1997, introduced the concept of strategy by giving explicit constructs for expressing control on the rule application [5]. Beyond labeled rules and concatenation denoted ";", other constructs for deterministic or non-determinitic choice, failure, iteration, were also defined in ELAN. A strategy is there defined as a set of proof terms in rewriting logic and can be seen as a higher-order function : if the strategy ζ is a set of proof terms π, applying ζ to the term t means finding all terms t' such that $\pi : t \to t'$ with $\pi \in \zeta$. Since rewriting logic is reflective, strategy semantics can be defined inside the rewriting logic by rewrite rules at the meta-level. This is the approach followed by Maude in [8, 18].

4 Rewriting Calculus

The rewriting calculus, also called ρ-calculus, has been introduced in 1998 by Horatiu Cirstea and Claude Kirchner [7]. As claimed on `http://rho.loria.fr/index.html`:

The rho-calculus has been introduced as a general means to uniformly integrate rewriting and λ-calculus. This calculus makes explicit and first-class all of its components: matching (possibly modulo given theories), abstraction, application and substitutions.

The rho-calculus is designed and used for logical and semantical purposes. It could be used with powerful type systems and for expressing the semantics of rule based as well as object oriented paradigms. It allows one to naturally express exceptions and imperative features as well as expressing elaborated rewriting strategies.

Some features of the rewriting calculus are worth emphasizing here: first-order terms and λ-terms are ρ-terms ($\lambda x.t$ is $(x \Rightarrow t)$); a rule is a ρ-term as well as a strategy, so rules and strategies are abstractions of the same nature and "first-class concepts"; application generalizes $\beta-$reduction; composition of strategies is like function composition; recursion is expressed as in λ calculus with a recursion operator μ.

In order to illustrate the use of ρ-calculus, let us consider the Abstract Biochemical Calculus (or ρ_{Bio}-calculus) [3]. This rewriting calculus models autonomous systems as *biochemical programs* which consist of the following components: collections of molecules (objects and rewrite rules), higher-order rewrite rules over molecules (that may introduce new rewrite rules in the behaviour of the system) and strategies for modelling the system's evolution. A visual representation via *port graphs* and an implementation are provided by the PORGY environment described in [1]. In this calculus, strategies are abstract molecules, expressed with an arrow constructor ($\Rightarrow$ for rule abstraction), an application operator $\bullet$ and a constant operator stk for explicit failure.

Below are examples of useful strategies in ρ_{Bio}-calculus:

$$
\begin{array}{rcl}
\mathsf{id} & \triangleq & X \Rightarrow X \\
\mathtt{fail} & \triangleq & X \Rightarrow \mathsf{stk} \\
\mathtt{seq}(S_1,S_2) & \triangleq & X \Rightarrow S_2\bullet(S_1\bullet X) \\
\mathtt{first}(S_1,S_2) & \triangleq & X \Rightarrow (S_1\bullet X)\ \ (\mathsf{stk} \Rightarrow (S_2\bullet X))\bullet(S_1\bullet X) \\
\mathtt{try}(S) & \triangleq & \mathtt{first}(S,\mathsf{id}) \\
\mathtt{not}(S) & \triangleq & X \Rightarrow \mathtt{first}(\mathsf{stk} \Rightarrow X, X' \Rightarrow \mathsf{stk})\bullet(S\bullet X) \\
\mathtt{ifTE}(S_1,S_2,S_3) & \triangleq & X \Rightarrow \mathtt{first}(\mathsf{stk} \Rightarrow S_3\bullet X, X' \Rightarrow S_2\bullet X)\bullet(S_1\bullet X) \\
\mathtt{repeat}(S) & \triangleq & \mu X.\mathtt{try}(\mathtt{seq}(S,X))
\end{array}
$$

Based on such constructions, the ρ_{Bio}-calculus allows failure handling, repair instructions, persistent application of rules or strategies, and more generally strategies for autonomic computing, as described in [2]. In [3], it is shown how to do invariant verification in biochemical programs. Thanks to ρ_{Bio}-calculus, an invariant property can in many cases, be encoded as a special rule in the biochemical program modelling the system and this rule is dynamically checked at each execution step. For instance, an invariant of the system is encoded by a rule $G \Rightarrow G$ and the strategy verifying such an invariant is encoded with a persistent strategy $\mathtt{first}(G \Rightarrow G, X \Rightarrow \mathsf{stk})$. In a similar way, an unwanted occurrence of a concrete molecule G in the system can be modeled with the rule ($G \Rightarrow \mathsf{stk}$). And instead of yielding failure stk, the problem can be "repaired" by associating to each property the necessary rules or strategies to be inserted in the system in case of failure.

5 Abstract Reduction Systems

Another view of rewriting is to consider it as an abstract relation on structural objects. An *Abstract Reduction System (ARS)* [19, 15, 6] is a labelled oriented graph $(\mathcal{O},\mathcal{S})$ with a set of labels $\mathcal{L}$. The nodes in $\mathcal{O}$ are called *objects*. The oriented labelled edges in $\mathcal{S}$ are called *steps*: $a \xrightarrow{\phi} b$ or (a,ϕ,b), with *source* a, *target* b and *label* ϕ. Derivations are composition of steps.

For a given ARS $\mathcal{A}$, an $\mathcal{A}$*-derivation* is denoted $\pi : a_0 \xrightarrow{\phi_0} a_1 \xrightarrow{\phi_1} a_2 \ldots \xrightarrow{\phi_{n-1}} a_n$ or $a_0 \xrightarrow{\pi} a_n$, where $n \in \mathbb{N}$. The *source* of π is a_0 and its domain $Dom(\pi) = \{a_0\}$. The *target* of π is a_n and applying π to a_0 gives the singleton set $\{a_n\}$, which is denoted $\pi\bullet a_0 = \{a_n\}$.

Abstract strategies are defined in [15] and in [6] as follows: for a given ARS $\mathcal{A}$, an *abstract strategy* ζ is a subset of the set of all derivations (finite or not) of $\mathcal{A}$. The notions of domain and application are generalized as follows: $Dom(\zeta) = \bigcup_{\pi\in\zeta} Dom(\pi)$ and $\zeta\bullet a = \{b \mid \exists\pi \in \zeta$ such that $a \xrightarrow{\pi} b\} = \{\pi\bullet a \mid \pi \in \zeta\}$. Playing with these definitions, [6] explored adequate definitions of termination, normal form and confluence under strategy.

Since abstract reduction systems may involve infinite sets of objects, of reduction steps and of derivations, we can schematize them with constraints at different levels: (i) to describe the objects occurring in a derivation (ii) to describe, via the labels, requirements on the steps of reductions (iii) to describe the structure of the derivation itself (iv) to express requirements on the histories. The framework developed in [16] defines a strategy ζ as all instances $\sigma(D)$ of a derivation schema D such that σ is solution of a constraint C involving derivation variables, object variables and label variables. As a simple example, the infinite set of derivations of length one that transform a into $f(a^n)$ for all $n \in \mathbb{N}$, where $a^n = a * \ldots * a$ (n times), is simply described by: $(a \to f(X) \mid X * a =_A a * X)$, where $=_A$ indicates that the constraint is solved modulo associativity of the operator $*$. This very general definition of abstract strategies is called *extensional* in [6] in the sense that a strategy is defined explicitly as a set of derivations of an abstract reduction system. The concept is useful to understand and unify reduction systems and deduction systems as explored in [15].

But abstract strategies do not capture another point of view, also frequently adopted in rewriting: a strategy is a partial function that associates to a reduction-in-progress, the possible next steps in the reduction sequence. Here, the strategy as a function depends only on the object and the derivation so far. This notion of strategy coincides with the definition of strategy in sequential path-building games, with applications to planning, verification and synthesis of concurrent systems [9]. This remark leads to the following *intensional* definition given in [6]. The essence of the idea is that strategies are considered as a way of constraining and guiding the steps of a reduction. So at any step in a derivation, it should be possible to say whether a contemplated next step obeys the strategy ζ. In order to take into account the past derivation steps to decide the next possible ones, the history of a derivation has to be memorized and available at each step. Through the notion of traced-object $[\alpha]a = \Big[(a_0,\phi_0),\ldots,(a_n,\phi_n)\Big]a$ in $\mathcal{O}^{[\mathcal{A}]}$, each object a memorizes how it has been reached with the trace α.

An *intensional strategy* for $\mathcal{A} = (\mathcal{O},\mathcal{S})$ is a partial function λ from $\mathcal{O}^{[\mathcal{A}]}$ to $2^{\mathcal{S}}$ such that for every traced object $[\alpha]a$, $\lambda([\alpha]a) \subseteq \{\pi \in \mathcal{S} \mid Dom(\pi) = a\}$. If $\lambda([\alpha]a)$ is a singleton, then the reduction step under λ is deterministic.

As described in [6], an intensional strategy λ naturally generates an abstract strategy, called its *extension*: this is the abstract strategy ζ_λ consisting of the following set of derivations:

$\forall n \in \mathbb{N}, \pi : a_0 \xrightarrow{\phi_0} a_1 \xrightarrow{\phi_1} a_2 \ldots \xrightarrow{\phi_{n-1}} a_n \in \zeta_\lambda \qquad \text{iff} \qquad \forall j \in [0,n], \quad (a_j \xrightarrow{\phi_j} a_{j+1}) \in \lambda([\alpha]a_j).$

This extension may obviously contain infinite derivations; in such a case it also contains all the finite derivations that are prefixes of the infinite ones, and so is closed under taking prefixes.

A special case are memoryless strategies, where the function λ does not depend on the history of the objects. This is the case of many strategies used in rewriting systems, as shown in the next example. Let us consider an abstract reduction system $\mathcal{A}$ where objects are terms, reduction is term rewriting with a rewrite rule in the rewrite system, and labels are positions where the rewrite rules are applied. Let us consider an order $<$ on the labels which is the prefix order on positions. Then the intensional strategy that corresponds to innermost rewriting is $\lambda_{inn}(t) = \{\pi : t \xrightarrow{p} t' \mid p = max(\{p' \mid t \xrightarrow{p'} t' \in \mathcal{S}\})\}$. When a lexicographic order is used, the classical *rightmost-innermost* strategy is obtained.

Another example, to illustrate the interest of traced objects, is the intensional strategy that restricts the derivations to be of bounded length k. Its definition makes use of the size of the trace α, denoted $|\alpha|$: $\lambda_{ltk}([\alpha]a) = \{\pi \mid \pi \in \mathcal{S},\ Dom(\pi) = a,\ |\alpha| < k-1\}$. However, as noticed in [6], the fact that intensional strategies generate only prefix closed abstract strategies prevents us from computing abstract strategies that look straightforward: there is no intensional strategy that can generate a set of derivations of length exactly k. Other solutions are provided in [6].

6 Conclusion

A lot of interesting questions about strategies are yet open, going from the definition of this concept and the interesting properties we may expect to prove, up to the definition of domain specific strategy languages. As further research topics, two directions seem really interesting to explore:

- The connection with Game theory strategies. In the fields of system design and verification, *games* have emerged as a key tool. Such games have been studied since the first half of 20th century in descriptive set theory [14], and they have been adapted and generalized for applications in formal verification; introductions can be found in [13, 20]. It is worth wondering whether the coincidence of the term "strategy" in the domains of rewriting and games is more than a pun. It should be fruitful to explore the connection and to be guided in the study of the foundations of strategies by some of the insights in the literature of games.

- Proving properties of strategies and strategic reductions. A lot of work has already begun in the rewriting community and have been presented in journals, workshops or conferences of this domain. For instance, properties of confluence, termination, or completeness for rewriting under strategies have been addressed, either based on schematization of derivation trees, as in [12], or by tuning proof methods to handle specific strategies (innermost, outermost, lazy strategies) as in [10, 11]. Other approaches as [4] use strategies transformation to equivalent rewrite systems to be able to reuse well-known methods. Finally, properties of strategies such as fairness or loop-freeness could be worthfully explored by making connections between different communities (functional programming, proof theory, verification, game theory,...).

Acknowledgements The results presented here are based on pioneer work in the ELAN language designed in the Protheo team from 1997 to 2002. They rely on joint work with many people, in particular Marian Vittek and Peter Borovanský, Claude Kirchner and Florent Kirchner, Dan Dougherty, Horatiu Cirstea and Tony Bourdier, Oana Andrei, Maribel Fernandez and Olivier Namet. I am grateful to José Meseguer and to the members of the PROTHEO and the PORGY teams, for many inspiring discussions on the topics of this talk.

References

[1] O. Andrei, M. Fernández, H. Kirchner, G. Melançon, O. Namet & B. Pinaud (2011): *PORGY: Strategy Driven Interactive Transformation of Graphs*. In: *6th International Workshop on Computing with Terms and Graphs (TERMGRAPH 2011), Saarbrucken, April 2011*, Electronic Proceedings In Theoretical Computer Science 48, pp. 54–68.

[2] Oana Andrei & Hélène Kirchner (2009): *A Higher-Order Graph Calculus for Autonomic Computing*. In: *Graph Theory, Computational Intelligence and Thought. Golumbic Festschrift, Lecture Notes in Computer Science* 5420, Springer, pp. 15–26.

[3] Oana Andrei & Hélène Kirchner (2009): *A Port Graph Calculus for Autonomic Computing and Invariant Verification*. *Electronic Notes In Theoretical Computer Science* 253(4), pp. 17–38.

[4] Emilie Balland, Pierre-Etienne Moreau & Antoine Reilles (2012): *Effective strategic programming for Java developers*. *Software: Practice and Experience*.

[5] Peter Borovanský, Claude Kirchner, Hélène Kirchner & Pierre-Etienne Moreau (2002): *ELAN from a rewriting logic point of view*. *Theoretical Computer Science* 2(285), pp. 155–185.

[6] Tony Bourdier, Horatiu Cirstea, Daniel J. Dougherty & Hélène Kirchner (2009): *Extensional and Intensional Strategies*. In: *Proceedings Ninth International Workshop on Reduction Strategies in Rewriting and Programming, Electronic Proceedings In Theoretical Computer Science* 15, pp. 1–19.

[7] Horatiu Cirstea & Claude Kirchner (2001): *The rewriting calculus — Part I* and *II*. *Logic Journal of the Interest Group in Pure and Applied Logics* 9(3), pp. 427–498.

[8] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer & Carolyn Talcott (2007): *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Reriting Logic*. 4350, Springer.

[9] Daniel J. Dougherty (2008): *Rewriting strategies and game strategies*. Internal report.

[10] J. Giesl & A Middeldorp (2003): *Innermost Termination of Context-Sensitive Rewriting*. In: *Proceedings of the 6th International Conference on Developments in Language Theory (DLT 2002), LNCS* 2450, Springer, Kyoto, Japan, pp. 231–244.

[11] Jürgen Giesl, Matthias Raffelsieper, Peter Schneider-Kamp, Stephan Swiderski & René Thiemann (2011): *Automated termination proofs for Haskell by term rewriting*. *ACM Trans. Program. Lang. Syst.* 33(2), pp. 7:1–7:39.

[12] Isabelle Gnaedig & Hélène Kirchner (2009): *Termination of rewriting under strategies*. *ACM Trans. Comput. Logic* 10(2), pp. 1–52.

[13] Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors (2002): *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*. *Lecture Notes in Computer Science* 2500, Springer.

[14] Alexander S. Kechris (1995): *Classical Descriptive Set Theory*. *Graduate Texts in Mathematics* 156, Springer.

[15] Claude Kirchner, Florent Kirchner & Hélène Kirchner (2008): *Strategic Computations and Deductions*. In Christoph Benzmüller, Chad E. Brown, Jörg Siekmann & Richard Statman, editors: *Reasoning in Simple Type Theory. Festchrift in Honour of Peter B. Andrews on His 70th Birthday, Studies in Logic and the Foundations of Mathematics* 17, College Publications, pp. 339–364.

[16] Claude Kirchner, Florent Kirchner & Hélène Kirchner (2010): *Constraint Based Strategies*. In: *Proceedings 18th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2009), Brasilia, LNCS* 5979, pp. 13–26.

[17] Narciso Martí-Oliet & José Meseguer (2000): *Rewriting logic as a logical and semantic framework*. In J. Meseguer, editor: *Electronic Notes in Theoretical Computer Science*, 4, Elsevier Science Publishers.

[18] Narciso Martí-Oliet, José Meseguer & Alberto Verdejo (2008): *A rewriting semantics for Maude strategies*. *Electronic Notes in Theoretical Computer Science* 238(3), pp. 227–247.

[19] Vincent van Oostrom & Roel de Vrijer (2003): *Term Rewriting Systems*, chapter 9: Strategies. *Cambridge Tracts in Theoretical Computer Science* 2, Cambridge University Press.

[20] Igor Walukiewicz (2004): *A Landscape with Games in the Background*. In: *19th IEEE Symposium on Logic in Computer Science (LICS 2004)*, pp. 356–366.