Rewriting Logic and its Applications

Narciso Martí-Oliet
Escuela Superior de Informática
Universidad Complutense de Madrid, Spain
narciso@eucmos.sim.ucm.es

Rewriting logic is a logic to specify, reason about, and program concurrent and distributed systems. It was first proposed by Meseguer as a unifying framework for concurrency in 1990 [7, 8]. Since then a large body of work by researchers around the world has contributed to the development of several aspects of the logic and its applications in different areas of computer science. The recent paper [10] by Meseguer surveys the main contributions to the use of rewriting logic as a semantic framework for concurrency, and the proceedings of the *First International Workshop on Rewriting Logic and its Applications* that took place in September 1996 in Asilomar, California [11] can give a good idea about current research directions in this field.

In this tutorial, we will summarize some of the key developments in the use of rewriting logic as a logical and semantic framework [6, 10], as well as the more recent work on the reflective properties of rewriting logic [2, 3, 4].

A *signature* in (order-sorted) rewriting logic is an (order-sorted) equational theory (Σ, E), where Σ is an equational signature and E is a set of Σ-equations. Rewriting will operate on equivalence classes of terms modulo the set of Σ-equations E. Given a signature (Σ, E), *sentences* of the logic are sequents of the form $[t]_E \longrightarrow [t']_E$, where t and t' are Σ-terms possibly involving some variables. A *rewrite theory* $\mathcal{R} = (\Sigma, E, L, R)$ consists of a signature (Σ, E), a set of labels L, and a set R of labelled *rewrite rules* of the form $r : [t] \longrightarrow [t']$, where r is a label and $[t]$ and $[t']$ are equivalence classes of terms in $T_{\Sigma,E}(X)$ for a set of variables X.

The results can be extended to more expressive conditional rules of the form

$$r : [t] \longrightarrow [t'] \ \textit{if} \ [u_1] \longrightarrow [v_1] \ \wedge \ldots \wedge \ [u_k] \longrightarrow [v_k].$$

Given a rewrite theory $\mathcal{R}$, we say that $\mathcal{R}$ *entails* a sentence $[t] \longrightarrow [t']$ and write $\mathcal{R} \vdash [t] \longrightarrow [t']$ if and only if $[t] \longrightarrow [t']$ can be obtained by finite application of four simple *rules of deduction*: Reflexivity, Congruence, Replacement, and Transitivity.

The main idea is that given a concurrent system $\mathcal{S}$ axiomatized by a rewrite theory $\mathcal{R}$, a concurrent computation from a state S to a state S' is possible in $\mathcal{S}$ if and only if $\mathcal{R} \vdash S \longrightarrow S'$ using the deduction rules of rewriting logic.

Meseguer [8] has also developed a categorical model theory for rewriting logic, and proved corresponding initiality, freeness, soundness, and completeness theorems. In particular, the initial model of a rewrite theory $\mathcal{R}$ provides a precise mathematical model of the concurrent system axiomatized by such a theory.

Rewriting logic admits a double interpretation, which is summarized in the following table, obtained by adopting either a computational point of view (semantic framework), or a logical one (logical framework).

State	$\leftrightarrow$	*Term*	$\leftrightarrow$	*Proposition*
Transition	$\leftrightarrow$	*Rewriting*	$\leftrightarrow$	*Deduction*
Distributed structure	$\leftrightarrow$	*Algebraic structure*	$\leftrightarrow$	*Propositional structure*

Rewriting logic can be used directly as a wide spectrum language supporting specification, rapid prototyping, and programming of concurrent systems [5]. This is realized in the multiparadigm Maude language in which modules are theories in rewriting logic [9]. As a special case of modules, Maude provides syntactic support for the specification and programming of concurrent object-oriented systems.

In this way, we can use rewriting logic to specify the semantics of a wide variety of programming languages (including parallel languages), models of computation (including concurrent ones), and software systems. Besides, using a Maude interpreter, such specifications can be executed and used for fast and convenient development of interpreters and software prototypes. In particular, most structural operational semantics specifications can be regarded as special cases of rewriting logic specifications. Among the many models of computation that have a simple and natural expression as rewrite theories [8, 10], we would like to point out

- Petri nets,
- Concurrent objects and actors,
- Milner's CCS,
- Objects with concurrent access,
- Graph rewriting,
- Real time systems.

It is important to emphasize that all models appear as special cases, without any encoding, by appropriate choices of syntax, and structural axioms.

From the other point of view, rewriting logic also seems a good candidate for a logical framework because syntax is *user-definable* and can be very abstract thanks to user-definable *structural axioms*. Any order-sorted data type of formulas can thus be defined. Moreover, there are only four "metarules" of deduction which are applied relative to chosen, user-definable rewrite rules. Typically, formulas or proof-theoretic structures such as sequents are represented as *terms* in an order-sorted equational data type whose equations express structural axioms natural to the logic in question. Then, the rules of deduction of a logic are represented as rewrite rules that transform certain patterns of formulas into other patterns modulo the given structural axioms. In this way, we have developed in [6] representations for:

- Equational logic,
- Horn logic with equality,
- Linear logic,
- Logics with quantifiers,
- Any logic describable with a sequent calculus.

Some of the most recent and interesting research on rewriting logic by Clavel and Meseguer has focused on its *reflective* properties [2, 3, 4]. Reflection allows a computational or logical system to access its own metalevel, and in this way the system can

be much more powerful, flexible and adaptable. Rewriting logic is reflective in a precise mathematical sense, namely, in that there is a *universal rewrite theory* $\mathcal{U}$ in which any other rewrite theory $\mathcal{R}$ (including itself) can be represented as a term $\overline{\mathcal{R}}$ in such a way that

$$\mathcal{R} \vdash t \longrightarrow t' \iff \mathcal{U} \vdash \langle \overline{\mathcal{R}}, \overline{t} \rangle \longrightarrow \langle \overline{\mathcal{R}}, \overline{t'} \rangle.$$

Some applications of these reflective properties include the internal representation of maps between logics as terms in rewriting logic, which can be very useful for its already mentioned use as logical framework, as well as giving mathematical semantics to the ideas of object-oriented and actor reflection, which expands the semantic framework point of view.

The use of reflection that will be emphasized in this tutorial is the definition of *internal strategy languages* to define adequate strategies for controlling the rewriting process, which in principle could go in many undesired directions. The important issue to point out in this context is that, thanks to reflection, these languages are also based on rewriting, and their semantics and implementation are described in the same logic. This allows us to define the strategies by rewriting rules and to implement them in a reflective rewriting logic language like Maude [3, 4]. In this way, internal strategies can be given a declarative semantics within the same logic that they control, that is, control is not an extra-logical addition to the language but remains declaratively inside the logic.

In fact there is great freedom for defining different strategy languages inside Maude. This can be done in a completely user-definable way, so the users are not limited by a fixed and closed strategy language. Clavel and Meseguer propose the following methodology for defining a strategy language. First, a kernel is defined stating how rewriting in the object level is accomplished at the metalevel. Then, the strategy language of choice extends the kernel with the desired additional strategy expressions and corresponding semantic definitions and rules.

The internal strategy language thus defined is used in [4] to illustrate the advantages of this approach by showing how the rules of inference for Knuth-Bendix completion can be given strategies corresponding to completion procedures in a completely modular way, not requiring any change to the inference rules themselves, and cleanly separating the logical or proof-theoretical component of the completion method from issues pertaining to control. Indeed, this example requires the use of three levels of reflection, and keeping the separation between the different levels is the key to the modularity thus obtained.

The overall goal of the study of rewriting logic and its applications is to serve as a guide for the design and implementation of a theoretically-based high-level system and associated tools in which it can be easy to define logics and to perform deductions in them, and in which a very wide variety of systems, languages, and models of computation can similarly be specified and prototyped. The following features seem particularly useful:

- Executability,
- User-definable abstract syntax,
- Modularity and parameterization,
- Simple and general logical semantics,
- Reflection.

All these features are supported by a Maude interpreter, under development at SRI International by Meseguer and his collaborators.

References

[1] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer, Principles of Maude, in [11].

[2] M.Clavel and J. Meseguer, Axiomatizing reflective logics and languages, in: G. Kiczales, ed., *Proc. Reflection'96, San Francisco, CA*, April 1996, 263–288.

[3] M. Clavel and J. Meseguer, Reflection and strategies in rewriting logic, in [11].

[4] M. Clavel and J. Meseguer, *Internal strategies in a reflective logic*, manuscript, Computer Science Laboratory, SRI International, April 1997.

[5] P. Lincoln, N. Martí-Oliet, and J. Meseguer, Specification, transformation, and programming of concurrent systems in rewriting logic, in: G. E. Blelloch et al. (eds.), *Specification of Parallel Algorithms, DIMACS Workshop, May 1994*, American Mathematical Society, 1994, 309–339.

[6] N. Martí-Oliet and J. Meseguer, Rewriting logic as a logical and semantic framework, Technical report SRI-CSL-93-05, SRI International, August 1993. To appear in D. M. Gabbay, ed., *Handbook of Philosophical Logic*, Kluwer Academic Publishers. Short version in [11].

[7] J. Meseguer, Rewriting as a unified model of concurrency, in: J.C.M.Baeten and J.W.Klop, eds., *Proc. CONCUR'90*, LNCS 458, Springer-Verlag, 1990, 384–400.

[8] J. Meseguer, Conditional rewriting logic as a unified model of concurrency, *Theoretical Computer Science* **96**, 1992, 73–155.

[9] J. Meseguer, A logical theory of concurrent objects and its realization in the Maude language, in: G. Agha, P. Wegner, and A. Yonezawa, eds., *Research Directions in Concurrent Object-Oriented Programming*, The MIT Press, 1993, 314–390.

[10] J. Meseguer, Rewriting logic as a semantic framework for concurrency: A progress report, in: U. Montanari and V. Sassone, eds., *Proc. CONCUR'96*, LNCS 1119, Springer-Verlag, 1996, 331–372.

[11] J. Meseguer, editor, *Proc. First Int. Workshop on Rewriting Logic and its Applications, Asilomar, CA*, Electronic Notes in Theoretical Computer Science 4, Elsevier, Sept. 1996. URL `http://www1.elsevier.nl/mcs/tcs/pc/volume4.htm`