Invited Talk III

A LOGICAL FRAMEWORK FOR DISTRIBUTED SYSTEMS AND COMMUNICATION PROTOCOLS

José Meseguer

Computer Science Laboratory
SRI International, Menlo Park
California 94025, USA
meseguer@csl.sri.com

WHY REWRITING LOGIC

I will introduce the main ideas of rewriting logic [11]—a logic for the specification, prototyping and programming of concurrent systems in which concurrent computation exactly corresponds to logical deduction—and will discuss some promising directions for its use as a logical and semantic framework for distributed computing and communication protocols. An important aspect is its *wide-spectrum* character. Thus, on the one hand it connects smoothly with—and provides a formal foundation for—notations suitable for the early phases of software design, such as architectural description languages and object-oriented modeling languages [14, 19]; and on the other hand it also provides a natural implementation path through subsets of the logic that are efficiently implementable as distributed or mobile languages [9]. Similarly, rewriting logic specifications, when supported by appropriate tools, can be used in a wide range of specification, prototyping, code generation, testing, formal analysis, and formal verification efforts to reach high levels of assurance about a system's correctness. All these capabilities seem potentially useful for specifying, prototyping, testing, validating, and implementing distributed systems in general and communication protocols in particular; and they offer the promise of substantially narrowing the gap between specification and code, and of reaching high assurance about the correctness of implementations that can themselves be realized in efficient subsets of rewriting logic.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35394-4_29

S. Budkowski et al. (eds.), *Formal Description Techniques and Protocol Specification, Testing and Verification*

For such applications, an attractive feature of rewriting logic is its simple and smooth integration of three levels of description that ultimately need to be addressed in a distributed system's formal specification, namely, the *data* level of data structures and functions, the *dynamic* level of processes and concurrent computation, and the *requirements* level of formal properties that must be satisfied by the system.

Indeed, a rewrite theory is a triple $\mathcal{R} = (\Sigma, E, R)$, with (Σ, E) an equational theory, and R a collection of labeled rewrite rules that are applied *modulo* the equations E; they axiomatize the basic *concurrent transitions* of the specified system. The two main kinds of axioms in a rewrite theory, namely, *equations* $t = t'$, and *rewrite rules* $r : t \longrightarrow t'$, specify, respectively, the data and dynamic levels. That is, the data types and the overall state space of the system are axiomatized as the initial algebra $T_{\Sigma,E}$ associated to the equational specification (Σ, E), and the rewrite rules[1] in R are understood as local rules of concurrent change in the intended system, so that rewriting logic deduction with such rules describes exactly those concurrent computations possible in the system. This one-to-one correspondence between concurrent computation and logical deduction is of course of great practical usefulness, because it means that, given an adequate implementation of rewriting logic, a rewrite theory $\mathcal{R}$ becomes an *executable specification* of the concurrent system that it formalizes. Such a specification can then be used for rapid prototyping, symbolic simulation, formal analysis, and formal verification, to uncover design errors very early in the design process; and can also be used to generate correct code by means of adequate semantics-preserving transformations.

Since a rewrite theory $\mathcal{R}$ has an *initial model* $\mathcal{T}_{\mathcal{R}}$ [11] that mathematically characterizes the system as a category whose objects are the system's states and whose morphisms are the system's concurrent computations, formal system properties are in essence *inductive properties* of such a model. They can be expressed either in a natural extension of rewriting logic allowing arbitrary quantification, or in a temporal or modal logic providing a convenient notation for such properties. In this way, the three levels of data, dynamics and properties are naturally and seamlessly addressed.

A REFLECTIVE LOGICAL AND SEMANTIC FRAMEWORK

Being a *semantic framework* means that rewriting logic, instead of building in a particular model of concurrency or distribution such as, for example, process algebras, allows a wide range of such models—including indeed process algebras and languages such as LOTOS, but including also many other models such as Petri nets, Actors, the Unity language, dataflow, concurrent objects, concurrent graph rewriting, the π-calculus, and real-time systems [12, 13]. Being a *logical framework* means that rewriting logic can be used as a *metalogic* in which many other logics can be naturally defined and executed [10]. Natural and simple faithful such representations have been defined for a wide range of logics including classical, intuitionistic and linear logic, temporal and modal logics, inductive equational logic, and any logics with a sequent calculus presentation.

In particular, logical representations of this kind have been used to define and build *theorem proving tools* in a rewriting logic language such as Maude [3]. The way in which a model of computation, a programming language, a design or architectural language, or a logic are represented in the rewriting logic framework is by means of *representation maps* of the form

$$\Phi : \mathcal{L} \longrightarrow \mathit{RWLogic}.$$

Such maps give a rewriting logic semantics to the language $\mathcal{L}$ in question.

A key property of rewriting logic is that it is *reflective* [6, 2], in the sense that rewriting logic can represent its own metalevel as well as those of other logics. In fact, there is a finitely presented rewrite theory $\mathcal{U}$ that is *universal* in the sense that for any finitely presented rewrite theory $\mathcal{R}$ (including $\mathcal{U}$ itself) we have the following equivalence

$$\mathcal{R} \vdash t \rightarrow t' \;\Leftrightarrow\; \mathcal{U} \vdash \langle \overline{\mathcal{R}}, \overline{t} \rangle \rightarrow \langle \overline{\mathcal{R}}, \overline{t'} \rangle,$$

where $\overline{\mathcal{R}}$ and $\overline{t}$ are terms representing $\mathcal{R}$ and t as data elements of $\mathcal{U}$. Specifically, $\overline{\mathcal{R}}$ is a term of sort `Module`, and $\overline{t}$ is a term of sort `Term` in $\mathcal{U}$. Since $\mathcal{U}$ is representable in itself as a term of sort `Module`, we can achieve a "reflective tower" with an arbitrary number of levels of reflection, since we have

$$\mathcal{R} \vdash t \rightarrow t' \;\Leftrightarrow\; \mathcal{U} \vdash \langle \overline{\mathcal{R}}, \overline{t} \rangle \rightarrow \langle \overline{\mathcal{R}}, \overline{t'} \rangle \;\Leftrightarrow\; \mathcal{U} \vdash \langle \overline{\mathcal{U}}, \overline{\langle \overline{\mathcal{R}}, \overline{t} \rangle} \rangle \rightarrow \langle \overline{\mathcal{U}}, \overline{\langle \overline{\mathcal{R}}, \overline{t'} \rangle} \rangle \ldots$$

For efficiency reasons, the Maude language provides key features of the universal theory $\mathcal{U}$ in a built-in module called `META-LEVEL` that supports an arbitrary number of levels of reflection [4].

An important use of reflection is defining and executing *within* rewriting logic itself representation maps $\Phi : \mathcal{L} \longrightarrow \mathit{RWLogic}$. Such maps associate to each module M in the language $\mathcal{L}$ a rewrite theory $\Phi(M)$ in *RWLogic*. In semantic framework applications the language $\mathcal{L}$ can be a model of computation, a programming language, or an architectural description language; in logical framework applications $\mathcal{L}$ is typically a logic. In all cases, both $\mathcal{L}$ and Φ are metalevel entities that are, in principle, outside rewriting logic; however, thanks to reflection, they can be *internalized*—or as it is sometimes said *reified*—within rewriting logic. The idea is that in the universal theory $\mathcal{U}$ rewrite theories are already reified in an algebraic data type `Module`; we can then define another such data type $\texttt{Module}_{\mathcal{L}}$ reifying the modules in $\mathcal{L}$, and can reify Φ as a function

$$\overline{\Phi} : \texttt{Module}_{\mathcal{L}} \longrightarrow \texttt{Module}.$$

Since typically Φ is a total computable function, by general results of Bergstra and Tucker [1] it can always be specified by Church-Rosser and terminating rewrite rules, and therefore can be defined and executed in a reflective rewriting logic language such as Maude.

Yet another important application of reflection is that rewriting logic computations can be controlled at the metalevel by means of *internal strategies* defined with rewrite rules [6, 2]. This is of great practical use for the formal analysis of highly concurrent systems such a communication protocols, because, once they have been specified in rewriting logic, we can define strategies that explore all the possible concurrent computations from some given initial state or states, and that check whether some key property is satisfied or violated in the new states. In this way, formal specifications can be analyzed and corrected very early in the design process by what amounts to symbolic model checking and symbolic testing.

RECENT AND FUTURE DEVELOPMENTS

Rewriting logic is at present a young international research program with over a hundred papers by authors in Europe, the US and Japan, and three language implementations (see the survey [13] and the upcoming Second Workshop Proceedings [8]). We are still in an early phase in the task of applying rewriting logic to distributed computing and communication protocols. However, in addition to the work on foundations, on models of concurrent computation, and on languages, some recent research focusing specifically on this area seems quite promising. For example, the paper [7] shows how cryptographic communication protocols and attackers can be specified, executed, tested, and analyzed in Maude, and how adequate execution strategies can search and find security violations. Similarly, Appendix B of [4] discusses a reliable broadcast protocol Maude specification currently being jointly developed by researchers at the University of California, Santa Cruz, Stanford University, and SRI International (Denker, García-Luna, Meseguer, Smith, Ölveczky, and Talcott); using executable specifications very early in the design process has quickly exposed a number of mistakes and deadlocks in the initial design. In the same vein, several fault-tolerant communication protocols have also been specified in Maude, and *active network algorithms* written in the PLAN language are currently being specified and formally analyzed in Maude as part of a joint collaboration between researchers at the University of Pennsylvania and at SRI International (Gunter, Meseguer, and Wang). Other very promising developments include the work of Najm and Stefani using rewriting logic to specify computational models for open distributed systems [15], Talcott's work on open distributed components [18], Pita's and Martí-Oliet's work specifying a network management system in rewriting logic [17], Nakajima's work on the semantics of the calculus of mobile ambients and on specifying a Java/ORB implementation of a network management system [16], and Wirsing's and Knapp's work on a systematic translation from object-oriented design notations to Maude specifications [19].

Much more work remains ahead. More experience should be gained through examples and case studies; compositional aspects—so that protocols and other distributed systems and services can be specified, reasoned about, and built in a much more modular way—should be systematically studied; specification and proof techniques for system properties at the requirements level—in adequate temporal or modal logics above rewriting logic—also need to be much further advanced; and efficient implementations of distributed and mobile rewriting logic languages need to be developed. All this will require a lively international cooperation between individual researchers and entire research teams that is already taking place and whose prospects seem very encouraging.

Acknowledgments

I wish to thank the organizers of FORTE/PSTV'98 for kindly suggesting the possibility of giving this talk. I also wish to thank all the researchers at SRI, Stanford University, UC Santa Cruz, the University of Pennsylvania and other research institutions with whom I am collaborating on these ideas for their many contributions. Some have been already mentioned by name above or in the references, and many other contributions are surveyed in [13]; I would like to thank also Jonathan Millen at SRI, who has given us many good suggestions about cryptographic protocols. This work has been supported by DARPA through Rome Laboratories Contract F30602-97-C-0312, by Office of Naval Research Contracts N00014-95-C-0225 and N00014-96-C-0114, and by National Science Foundation Grant CCR-9633363.

Notes

1. Both equations and rewrite rules can be *conditional*, but for simplicity I discuss the unconditional case.

References

[1] J. Bergstra and J. Tucker. Characterization of computable data types by means of a finite equational specification method. In J. W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming, Seventh Colloquium*, pages 76–90. Springer-Verlag, 1980. LNCS, Volume 81.

[2] M. Clavel. Reflection in general logics and in rewriting logic, with applications to the Maude language. Ph.D. Thesis, University of Navarre, 1998.

[3] M. Clavel, F. Durán, S. Eker, and J. Meseguer. Building equational logic tools by reflection in rewriting logic. In *Proc. of the CafeOBJ Symposium '98, Numazu, Japan.* CafeOBJ Project, April 1998.

[4] M. Clavel, F. Durán, S. Eker, J. Meseguer, and P. Lincoln. An introduction to Maude (beta version). Manuscript, SRI International, March 1998.

[5] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science.* Elsevier, 1996. `http://www1.elsevier.nl/mcs/tcs/pc/volume4.htm`.

[6] M. Clavel and J. Meseguer. Reflection and strategies in rewriting logic. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science.* Elsevier, 1996. `http://www1.elsevier.nl/mcs/tcs/pc/volume4.htm`.

[7] G. Denker, J. Meseguer, and C. Talcott. Protocol Specification and Analysis in Maude. In N. Heintze and J. Wing, editors, *Proc. of Workshop on Formal Methods and Security Protocols, 25 June 1998, Indianapolis, Indiana*, 1998.

[8] C. Kirchner and H. Kirchner (eds.). *Proc. 2nd Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, North Holland, 1998.

[9] P. Lincoln, N. Martí-Oliet, and J. Meseguer. Specification, transformation, and programming of concurrent systems in rewriting logic. In G. Blelloch, K. Chandy, and S. Jagannathan, editors, *Specification of Parallel Algorithms*, pages 309–339. DIMACS Series, Vol. 18, American Mathematical Society, 1994.

[10] N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science.* Elsevier, 1996. `http://www1.elsevier.nl/mcs/tcs/pc/volume4.htm`.

[11] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[12] J. Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In *Proc. CONCUR'96, Pisa, August 1996*, pages 331–372. Springer LNCS 1119, 1996.

[13] J. Meseguer. Research directions in rewriting logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, NATO Advanced Study Institute, Marktoberdorf, Germany, July 29 – August 6, 1997.* Springer-Verlag, 1998.

[14] J. Meseguer and C. Talcott. Using rewriting logic to interoperate architectural description languages (I and II). Lectures at the Santa Fe and Seattle DARPA-EDCS Workshops, March and July 1997. `http://www-formal.stanford.edu/clt/ArpaNsf/adl-interop.html`.

[15] E. Najm and J.-B. Stefani. Computational models for open distributed systems. In H. Bowman and J. Derrick, editors, *Formal Methods for Open Object-based Distributed Systems, Vol. 2*, pages 157–176. Chapman & Hall, 1997.

[16] S. Nakajima. Encoding mobility in CafeOBJ: an exercise of describing mobile code-based software architecture. In *Proc. of the CafeOBJ Symposium '98, Numazu, Japan.* CafeOBJ Project, April 1998.

[17] I. Pita and N. Martí-Oliet. A Maude specification of an object oriented database model for telecommunication networks. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science.* Elsevier, 1996. `http://www1.elsevier.nl/mcs/tcs/pc/volume4.htm`.

[18] C. L. Talcott. An actor rewrite theory. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of

Electronic Notes in Theoretical Computer Science. Elsevier, 1996. `http://www1.elsevier.nl/mcs/tcs/pc/volume4.htm`.

[19] M. Wirsing and A. Knapp. A formal approach to object-oriented software engineering. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996. `http://www1.elsevier.nl/mcs/tcs/pc/volume4.htm`.