Rewriting logic bibliography by topic: 1990–2011

Narciso Martí-Oliet *, Miguel Palomino, Alberto Verdejo

Dpto. de Sistemas Informáticos y Computación, Facultad de Informática, Universidad Complutense de Madrid, Spain

ARTICLE INFO

Article history:
Available online 26 June 2012

Keywords:
Rewriting logic
Concurrency
Logical frameworks
Temporal logics
Formal specification and verification
Programming language semantics
Networks and distributed systems
Real-time systems
Probabilistic systems
Security
Bioinformatics

ABSTRACT

This bibliography compiles, to the best of our knowledge, all the papers on rewriting logic and its applications which have been written during the more than 20 years that have passed since the introduction of rewriting logic in 1990. The papers are classified according to five main areas: foundations, logical and semantic framework, languages, tools, and applications.

© 2012 Elsevier Inc. All rights reserved.

Contents

1. Introduction ................................................................................ 783
2. Foundations ................................................................................ 783
  2.1. Rewriting logic ..................................................................... 784
  2.2. The $\rho$-calculus .................................................................. 786
  2.3. Coherence and computability ..................................................... 786
  2.4. Termination ........................................................................... 786
  2.5. Unification, narrowing, reachability analysis, and constraints ......... 787
  2.6. Reflection ............................................................................. 788
  2.7. Strategies ............................................................................. 788
  2.8. Modal and temporal logic properties ........................................... 789
  2.9. Simulation and abstraction ....................................................... 790
  2.10. Real-time rewrite theories ...................................................... 790
  2.11. Probabilistic rewrite theories ................................................... 791
  2.12. Tile logic ........................................................................... 791
3. Rewriting logic as a logical and semantic framework ...................... 792
  3.1. Representing logics ............................................................... 792
  3.2. Representing models of concurrency ......................................... 793
  3.3. Representing modeling languages ............................................ 793
  3.4. Rewriting logic semantics of programming languages ................. 794
  3.5. K framework ...................................................................... 795
4. Rewriting logic languages ............................................................ 796

* Work supported by MICINN Spanish project DESAFIOS10 (TIN2009-14599-C03-01) and Comunidad de Madrid program PROMETIDOS (S2009/TIC-1465).

* Corresponding author. Address: Facultad de Informática, Universidad Complutense de Madrid, C/Profesor José García Santesmases s/n, 28040 Madrid, Spain. Tel.: +34 913947557; fax: +34 913947529.

E-mail addresses: narciso@esi.ucm.es (N. Martí-Oliet), miguelpt@sip.ucm.es (M. Palomino), alberto@sip.ucm.es (A. Verdejo).

1567-8326/$ – see front matter © 2012 Elsevier Inc. All rights reserved.

http://dx.doi.org/10.1016/j.jlap.2012.06.001
1. Introduction

Rewriting logic was introduced in 1990 by Meseguer; since then, researchers around the world have contributed to the further development of its foundations, to the design and implementation of languages and of tools based on rewriting logic, to several extensions of the basic framework to include additional features such as time or probabilistic information, and, in general, to the application of this framework, its languages, and its associated tools to model, specify, program, verify, and analyze all kinds of systems, ranging from software systems and programming languages to bioinformatics and chemical systems.

All this work has given rise to an important number of papers, published mainly in journals and conference proceedings, but also as Master’s and Ph.D. theses, and technical reports. On the occasion of preparing the roadmap that introduced the 2002 *Theoretical Computer Science* special issue on rewriting logic and its applications, we compiled a bibliography on the subject consisting of more than 300 entries. The current bibliography has two goals with the intention of making it more useful for the interested community of researchers: first, to update that previous bibliography by taking into account all the work developed in the 10 years passed since then, and, second, to classify all the compiled papers. This classification has been organized according to five main areas: foundations, logical and semantic framework, languages, tools, and applications. Each of those areas is further divided into several subareas, as summarized in the table of contents above.

These are the criteria we have followed in classifying the bibliography:

- The subareas are not disjoint at all and, in some cases, one subarea could be included in another; for example, networks are distributed systems, and the K framework is devoted to developing and implementing semantics of programming languages. However, this division comes from previous surveys on this subject and is similar to the one used in the companion 20-year survey by Meseguer in this volume.
- Each paper appears only in one subarea.
- Since a paper can cover many subjects, it has been classified in the appropriate subarea according to the main contributions and the novelty of such.

The proposed classification has been validated by the rewriting logic community, since several drafts of this paper have been distributed along its preparation.

There are several factors who have greatly helped us in producing this bibliography: the DBLP computer science bibliography, the accuracy of Google’s search, and all the information provided by the authors of the papers themselves. We are very grateful to all of them for their willingness to include their work here.

The source bib file with all the entries in this bibliography is available at http://maude.sip.ucm.es/jlap-bibliography2012.bib.

2. Foundations

This section includes papers of a more theoretical nature on the foundations of rewriting logic, covering its proof theory and semantics; specification properties such as coherence, computability, and termination; extensions with additional features like time, probabilities, and strategies; and relations with associated logics such as the $\rho$-calculus, tile logic, and temporal logics.
2.1. Rewriting logic

The following papers address the theoretical foundations of rewriting logic and closely associated logics such as membership equational logic and order-sorted algebra, including denotational and categorical semantics, operational semantics, proof theory, extensions, and relationships with other logics.


J. Meseguer, Membership algebra as a logical framework for equational specification, in: [1096], pp. 18–61.


J. Meseguer, Twenty years of rewriting logic, in: [1093], pp. 15–17.


J. Meseguer, C.L. Talcott, A partial order event model for concurrent objects, in: [1025], pp. 415–430.

H. Miyoshi, Modelling conditional rewriting logic in structured categories, in: [1086], pp. 20–34.


G. Rosu, From conditional to unconditional rewriting, in: [1049], pp. 218–233.


W.M. Schorlemmer, Bi-rewriting rewriting logic, in: [1086], pp. 266–283.

W.M. Schorlemmer, Rewriting logic as a logic of special relations, in: [1071], pp. 197–217.


2.2. The $\rho$-calculus

The $\rho$-calculus is to rewriting what the $\lambda$-calculus is to functional programming; being another theoretical basis for rewriting, it is closely related to rewriting logic.


2.3. Coherence and computability

For rewrite theories to become executable specifications, they must satisfy some requirements so that the states and the rewrite relation become computable. The following papers address this issue in the form of coherence properties.


[114] G. Faure, C. Kirchner, Exceptions in the rewriting calculus, in: [1106], pp. 66–82.

2.4. Termination

The rich features of the rewriting logic framework and of the languages based on it have required the development of new methods to prove termination of rewrite theories in such framework and of programs in such languages. These new methods have also been integrated in appropriate tools, such as CARIBOO and MU-TERM.


2.5. Unification, narrowing, reachability analysis, and constraints

Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[146] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[147] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[148] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[149] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[150] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[151] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[152] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[153] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[154] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[155] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[156] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.

[157] M. Alpuente, D. Ballis, J. Espert, D. Romero, Narrowing generalizes rewriting by allowing unification (modulo equations) instead of matching (modulo equations) when applying a rule in a given step, thus instantiating variables both in the left-hand side of the rule being applied and in the term being reduced. In the context of rewriting logic, narrowing becomes a powerful method for symbolic reachability analysis with many important applications in automated deduction, constraint solving, model checking and security, among others.
Rewriting logic is reflective, that is, it can represent metalevel concepts, such as terms, theories, and deduction, at the object level, where they become usual data that can be used and manipulated by means of appropriate functions. All this is done in a so-called universal finitely presented rewrite theory. Logical reflection is a very powerful feature, related to computational reflection, and allows, among many other applications, the development of theorem proving and theory transformation tools, including theory composition operations that can be used to endow a language with a module algebra.

**2.6. Reflection**

Rewriting logic is reflective, that is, it can represent metalevel concepts, such as terms, theories, and deduction, at the object level, where they become usual data that can be used and manipulated by means of appropriate functions. All this is done in a so-called universal finitely presented rewrite theory. Logical reflection is a very powerful feature, related to computational reflection, and allows, among many other applications, the development of theorem proving and theory transformation tools, including theory composition operations that can be used to endow a language with a module algebra.

**References**


**2.7. Strategies**

Strategies are pervasive in computer science. In the context of rewriting logic, they can be used to control rewriting by equations (in the form of operator evaluation strategies), rewriting by rules, and more recently narrowing. A typical way of controlling the nondeterministic behavior of rules is to provide a strategy language which can be used to guide the application of rules. Several languages based on rewriting logic, such as ELAN, Maude, Stratego, and Tom, have developed this approach in different ways; in particular, Maude has made use of the reflective features of rewriting logic to define so-called internal strategies before providing an explicit strategy language.


2.8. Modal and temporal logic properties

Rewrite theories specify concurrent systems, whose properties can be specified in different logics, namely, appropriate modal and temporal logics. Indeed, on the one hand one can associate to a rewrite theory a transition system and, with suitable definitions of state predicates, a Kripke structure, and then one can use standard procedures to model check temporal properties in classical temporal logics, such as linear temporal logic (LTL). This path has led to the integration of an LTL model checker into Maude, for example. On the other hand, it is also possible to develop new modal and temporal logics, more
directly related to rewriting, such as the temporal logic of rewriting (TLR), and then it is necessary to develop new model-checking algorithms for such logics.


[250] M. Palomino, I. Pita, Proving VLRIL action properties with the Maude model checker, in: [1085], pp. 113–133.


2.9. Simulation and abstraction

A simulation relates two transition systems or two Kripke structures so that, under appropriate conditions, properties are translated from one system or structure to the other. This can be very useful, for example, to model check a temporal logic property of a system that in principle does not satisfy model-checking requirements such as being finite. The rewriting logic distinction between equations and rules provides direct abstraction methods consisting of turning rules into equations and of adding more equations, always trying to preserve the properties of interest and the executability conditions of the specifications.


[276] D.E. Rodríguez, Combining techniques to reduce state space and prove strong properties, in: [1099], pp. 267–280.

2.10. Real-time rewrite theories

The essential role played by real-time properties in the specification of many systems has led to the development of suitable methodologies to treat such properties in rewrite theories. Although it is possible to consider an extension of rewriting logic by adding explicit time to transitions, called timed rewriting logic, the most popular approach has been to consider a special kind of rewrite theories, called real-time rewrite theories, where time is taken care of by appropriate rules.
This second approach allows to represent different time domains as well as a wide range of models of real-time systems, and has been implemented in the Real-Time Maude tool, considered in Section 6.2.

2.11. Probabilistic rewrite theories

To take into account the nondeterministic and probabilistic aspects in many distributed systems, probabilistic rewrite rules have extra variables whose instantiation is controlled by a probability distribution. These rules constitute probabilistic rewrite theories, which can be used to represent different models of probabilistic systems. See Section 6.6 for applications of probabilistic rewrite theories.

2.12. Tile logic

Tile logic was designed as “a logic for modular descriptions of asynchronous and synchronized concurrent systems,” and it consists of an extension of (unconditional) rewriting logic obtained by adding state changes with side effects and synchronization. This allows the representation of process calculi, reactive systems, coordination languages, mobile calculi, etc.
3. Rewriting logic as a logical and semantic framework

Since rewriting logic was already introduced as "a general framework for unifying a wide variety of models of concurrency," it is not surprising that many different models of concurrent systems can be represented as rewrite theories; however, this also holds for many other models of computation, as well as very diverse logics (such as linear logic and type theories) and inference systems. For all these reasons, rewriting logic clearly deserves the designation of logical and semantic framework.

3.1. Representing logics

The following papers make use of rewriting logic as a logical framework by representing logics of a varied nature, including higher-order logic, linear logic, the connection calculus, inference systems, sequent calculi, explicit substitutions, pure type systems, the open calculus of constructions, and others.

References


3.1. Representing logics

The following papers make use of rewriting logic as a logical framework by representing logics of a varied nature, including higher-order logic, linear logic, the connection calculus, inference systems, sequent calculi, explicit substitutions, pure type systems, the open calculus of constructions, and others.

References


[325] O. Kahramanogullari, Maude as a platform for designing and implementing deep inference systems, in: [1107], pp. 35–50.


3.2. Representing models of concurrency

As the following papers show, in addition to the original papers by Meseguer, many researchers have devoted their attention to the representation in rewriting logic of many models of concurrent systems, including Petri nets of many kinds, actors, process calculi like Milner's CCS and π-calculus, LOTOS, mobile agents, and membrane and P systems.


3.3. Representing modeling languages

Since rewriting logic provides a good semantic framework to represent many different models of computation, it has been used to design, define, relate and give semantics to languages of all kinds, including agent languages, active networks languages, hardware and software architecture description languages, UML sublanguages, and domain-specific modeling languages. The particular case of operational semantics for concurrent programming languages is considered in the next section.

When a programming language is defined by means of a rewrite theory, if such a theory satisfies some requirements making it an executable specification, then the operational semantics of the programming language is obtained and programs in the language can be executed by means of rewriting. This methodology has been applied to many languages, as the following papers attest, but, more interesting than applying it to a given language, is to show that many techniques for defining operational semantics of concurrent programming languages can be mapped quite directly into rewriting logic. This has been shown for structural operational semantics (SOS) in its different variations (evaluation semantics or big step definition-based semantics, and the chemical abstract machine. A recent technique directly inspired by rewriting logic.

3.4. Rewriting logic semantics of programming languages

When a programming language is defined by means of a rewrite theory, if such a theory satisfies some requirements making it an executable specification, then the operational semantics of the programming language is obtained and programs in the language can be executed by means of rewriting. This methodology has been applied to many languages, as the following papers attest, but, more interesting than applying it to a given language, is to show that many techniques for defining operational semantics of concurrent programming languages can be mapped quite directly into rewriting logic. This has been shown for structural operational semantics (SOS) in its different variations (evaluation semantics or big step vs. computation semantics or small step), its extension to modular SOS, reduction semantics with evaluation contexts, continuation-based semantics, and the chemical abstract machine. A recent technique directly inspired by rewriting logic is the K framework, considered in the following section.


3.5. K framework

K is a recent and ambitious new technique for defining operational semantics of concurrent programming languages which tries to combine the advantages of previous techniques (evaluation, computation, and reduction semantics, modular SOS, etc.) and at the same time to overcome their limitations such as the difficulties to deal with control, non-modularity, and the lack of support for true concurrency. Furthermore, it is also expected that the executable semantics provide prototype interpreters even for complex and big languages. The K framework is based on configurations, computations, and rules, and has been inspired by continuation-based semantics, the chemical abstract machine, and rewriting logic. The ongoing
work on K includes an implementation based on the rewriting logic language Maude, and the development of semantics for different real-world languages.

4. Rewriting logic languages

Although there are many rule-based languages, we have singled out those whose designs have been directly inspired by rewriting logic, supporting in this way specification and programming based on rewrite theories. From the mid-nineties on there have been three main implementation efforts directed towards such languages: CafeOBJ in Japan, ELAN in France, and Maude in the USA. More recently, the developers of ELAN have started another project called Tom.
4.2. ELAN

ELAN was developed at LORIA, France, with a special emphasis on using strategies to guide rewriting. Together with the implementation of an interpreter and a compiler supporting new techniques for associative-commutative rewriting, the ELAN team and their collaborators developed the notion of computational system, consisting of a rewrite theory and a collection of strategies, and used it to represent many case studies in logic programming, constraint solving, higher-order unification, and equational theorem-proving, thus greatly contributing to the idea of rewriting logic as a logical and semantic framework. The techniques developed in this context have been more recently transferred to the new language Tom, discussed in the next section.


[490] P. Borovanský, C. Kirchner, H. Kirchner, Strategies and rewriting in ELAN, in: [1061], pp. 13–24.


[492] P. Borovanský, C. Kirchner, H. Kirchner, Rewriting as a unified specification tool for logic and control: The ELAN language, in: [1103].


[494] P. Borovanský, C. Kirchner, H. Kirchner, P.E. Moreau, C. Ringeissen, An overview of ELAN, in: [1071], pp. 55–70.

[495] P. Borovanský, C. Kirchner, H. Kirchner, P.E. Moreau, Elan: A logical framework based on computational systems, in: [1086], pp. 35–50.


[500] E. Deplagne, C. Kirchner, H. Kirchner, Q.H. Nguyen, Proof search and proof check for equational and inductive theorems, in: [1022], pp. 297–316.


[508] C. Kirchner, H. Kirchner, Rule-based constraint programming, Fundamenta Informaticae 34 (1998) 255–262.


[510] H. Kirchner, P.E. Moreau, Protoptyping completion with constraints using computational systems, in: [1066], pp. 438–443.

[511] H. Kirchner, P.E. Moreau, Non-deterministic computations in ELAN, in: [1047], pp. 168–182.


[517] P.E. Moreau, H. Kirchner, Compilation techniques for associative-commutative normalisation, in: [1103].
4.3. Tom

The Tom language is an extension of Java with capabilities based on matching and rewriting, thus sitting in an intermediate level between lower-level programming and higher-level specification. These rewriting capabilities are inherited from the ELAN language, commented in the previous section, and include support for strategies to guide non-deterministic computations and pattern matching modulo associativity and commutativity; a newer addition is anti-pattern matching, which is like a negative version of the more usual pattern matching.


4.4. Maude

The Maude language is developed at SRI International and the University of Illinois at Urbana-Champaign, USA, together with some collaborators in Spain. Maude supports functional modules, which are executable membership equational logic theories; system modules, which are executable rewrite theories; and object-oriented modules. Moreover, both equations and rules can be conditional, and rewriting can be modulo axioms such as associativity, commutativity, and/or identity. Maude also supports quite efficiently the reflective properties of rewriting logic discussed in Section 2.6, thus providing powerful metaprogramming features. In addition, the Maude system includes analysis capabilities based on breadth-first search and LTL model-checking. Some recently added features include order-sorted unification modulo axioms and narrowing.

[531] C. Ringeissen, Handling relations over finite domains in the rule-based system ELAN , in: [1052], pp. 194–211.

4.3. Tom

The Tom language is an extension of Java with capabilities based on matching and rewriting, thus sitting in an intermediate level between lower-level programming and higher-level specification. These rewriting capabilities are inherited from the ELAN language, commented in the previous section, and include support for strategies to guide non-deterministic computations and pattern matching modulo associativity and commutativity; a newer addition is anti-pattern matching, which is like a negative version of the more usual pattern matching.


4.4. Maude

The Maude language is developed at SRI International and the University of Illinois at Urbana-Champaign, USA, together with some collaborators in Spain. Maude supports functional modules, which are executable membership equational logic theories; system modules, which are executable rewrite theories; and object-oriented modules. Moreover, both equations and rules can be conditional, and rewriting can be modulo axioms such as associativity, commutativity, and/or identity. Maude also supports quite efficiently the reflective properties of rewriting logic discussed in Section 2.6, thus providing powerful metaprogramming features. In addition, the Maude system includes analysis capabilities based on breadth-first search and LTL model-checking. Some recently added features include order-sorted unification modulo axioms and narrowing.

[531] C. Ringeissen, Handling relations over finite domains in the rule-based system ELAN , in: [1052], pp. 194–211.
5. Tools

This section collects papers describing tools mostly written in Maude. We distinguish between Maude tools whose goal is to reason about Maude modules themselves, and other tools designed with the objective of using rewriting logic-based methods to specify and analyze systems in several application domains, supporting the corresponding domain-specific notations.

5.1. Maude tools

As discussed in Section 2.3, in order to become executable, rewrite theories must satisfy some requirements, such as coherence between rules and equations, for example. Moreover, the user might also be interested in proving properties satisfied by the specified system; as seen in Section 2.8, modal and temporal logics are suitable for this. The tools described in the following papers provide support for the verification of properties of these two kinds, including a Church-Rosser checker, a coherence checker, a sufficient completeness checker, a termination tool, a declarative debugger, an inductive theorem prover, and model checkers for LTL and fragments of TLR. The Maude Formal Environment is a recent effort to integrate and interoperate most of these tools. There are a couple of tools which could be included here, but we have decided instead to consider them in their respective application areas, namely, Real-Time Maude for the specification and analysis of real-time systems in Section 6.2, and PMaude for probabilistic systems in Section 6.6.
5.2. Other tools

Rewriting logic in general, and Maude in particular, have been used to define and implement a considerable number of tools for describing and analyzing systems of varied nature; to name just a couple of examples, JavaFAN is a tool for the formal analysis of Java programs, while CIRC implements and automates the theorem-proving circular coinduction technique. Other tools, like the InterOperability Platform, are designed with the goal of making easier the interaction of Maude with external tools. There are many more tools that could have been included in this section; however, whenever we have found another section devoted to an application area where the tool in question is particularly relevant, we have preferred to include the corresponding papers there; for example, the open calculus of constructions is included in Section 3.1 on rewriting logic as a logical framework, the K system plays an essential role in Section 3.5 on the K framework, the Maude-NPA protocol analyzer appears in Section 6.5 on security applications, and the Pathway Logic Assistant is the main tool in Section 6.7 on bioinformatics applications, to name a few.

[592] F. Durán, P.C. Olveczky, A guide to extending Full Maude illustrated with the implementation of Real-Time Maude, in: [1099], pp. 83–102.
6. Applications

As discussed in Section 3, rewriting logic has very good properties as a logical and semantic framework; this implies, in particular, that many kinds of systems have very natural representations as ( executable ) rewrite theories. Moreover, the rewriting logic-based languages are indeed general-purpose specification and programming languages, and therefore there is no limit to the applications that can be developed using them. However, the experience accumulated along these 20 years of work and compiled in this bibliography shows that they have been preferred in some areas, such as distributed architectures and networks, and security ( of course, another reason for this is the fact that these areas are fundamental in the current developments in Computer Science ). As in other occasions, both the named areas and the classification of papers in them follow somewhat arbitrary conventions, so, for example, automated deduction applications do not appear in this section, because they have been included in previous sections. Finally, it is clear that some extensions of Maude with particular goals have important applications in the intended areas: Real-Time Maude for real-time systems, PMaude for probabilistic systems, Maude-NPA for security, and the Pathway Logic Assistant for bioinformatics.

6.1. Network systems

The explicit emphasis on rewriting logic as a unifying framework for concurrency has endowed it with a considerable flexibility to model distributed objects and the many different modes of communication and interaction between them; for this reason, it is very well suited for the specification and analysis of network architectures and of communication protocols, including active networks and wireless sensor networks. Although most of the case studies included in this section are applied to existing network protocols, some others have also been used to design and study new protocols before their implementation. Moreover, the work reported in this section is complemented by the work on real-time systems in Section 6.2, because time is paramount in some protocols, and by the work on security in Section 6.5, because many protocols are, for example, cryptographic protocols or have essential security concerns. Finally, also included here is work on Mobile Maude, an extension to handle mobile agents deployed on networks.


6.2. Real-time and hybrid systems

Most of the following papers put into practice the theory on real-time systems described in Section 2.10, typically by means of the Real-Time Maude Tool. Indeed, Real-Time Maude is a specification language and a tool written in Maude itself which provides special syntax to specify real-time systems and suitable commands to analyze the timed properties by means of rewriting, search, and model checking. As this section evidences, there has been a substantial body of work in this area, applying these techniques to several protocols, distributed services, embedded systems, synchronous systems, and visual modeling, among others. More recent work on Real-Time Maude focuses on extending its implementation to handle other continuous variables, like temperature, thus broadening its applicability to a wide range of hybrid systems.

6.3. Distributed architectures and components

For the same reason explained at the beginning of Section 6.1, rewriting logic is also very well suited for the specification and analysis of distributed architectures and components, and this is explicitly confirmed by all the relevant work reported in the following papers. These include specifications and appropriate analyses for distributed software architectures, object-oriented distributed systems and frameworks, middleware architectures for composable services, reference models for Open Distributed Systems and frameworks, among others.

Distributed architectures and components

For the same reason explained at the beginning of Section 6.1, rewriting logic is also very well suited for the specification and analysis of distributed architectures and components, and this is explicitly confirmed by all the relevant work reported in the following papers. These include specifications and appropriate analyses for distributed software architectures, object-oriented distributed systems and frameworks, middleware architectures for composable services, reference models for Open Distributed Systems and frameworks, among others.

6.3. Distributed architectures and components

For the same reason explained at the beginning of Section 6.1, rewriting logic is also very well suited for the specification and analysis of distributed architectures and components, and this is explicitly confirmed by all the relevant work reported in the following papers. These include specifications and appropriate analyses for distributed software architectures, object-oriented distributed systems and frameworks, middleware architectures for composable services, reference models for Open Distributed Systems and frameworks, among others.

6.3. Distributed architectures and components

For the same reason explained at the beginning of Section 6.1, rewriting logic is also very well suited for the specification and analysis of distributed architectures and components, and this is explicitly confirmed by all the relevant work reported in the following papers. These include specifications and appropriate analyses for distributed software architectures, object-oriented distributed systems and frameworks, middleware architectures for composable services, reference models for Open Distributed Systems and frameworks, among others.

6.3. Distributed architectures and components

For the same reason explained at the beginning of Section 6.1, rewriting logic is also very well suited for the specification and analysis of distributed architectures and components, and this is explicitly confirmed by all the relevant work reported in the following papers. These include specifications and appropriate analyses for distributed software architectures, object-oriented distributed systems and frameworks, middleware architectures for composable services, reference models for Open Distributed Systems and frameworks, among others.


6.4. Software/hardware modeling and verification

This is the most general area, since it goes without saying that the vast majority of computational specifications will be developed for software or hardware systems of various kinds. Furthermore, the areas considered in previous sections on networks, real-time systems, and distributed components can very well be viewed as subareas of the present one, but their special and distinctive characteristics justify a separated treatment. This makes the collection of papers compiled here a miscellaneous assortment of rewriting logic applications, including web modeling and verification, prototyping of reconﬁgurable systems, static checking of units, asynchronous hardware designs, protocols for electronic payments, test-case generation, soft constraints, and certification of domain-speciﬁc properties, among many others.
Rewriting logic and its associated languages have been successfully applied to analyze security properties, such as secrecy, authenticity, availability, non-interference, etc., of many systems, including cryptographic protocols, network security, browser security, access control, and code security. Maude has been used to define the semantics of the cryptographic protocol specification language CAPSL and of the MSR security specification formalism, providing in this way execution and formal analysis environments for them. Using narrowing as a semi-decision procedure for symbolic reachability analysis, the Maude-NPA protocol analysis tool has been developed as an updated and greatly improved version of the NRL Protocol Analyzer. Maude-NPA takes into account the algebraic properties of the protocol, such as Diffie-Hellman exponentiation and exclusive or, which an attacker could use to break the protocol. Work on network security includes dynamically adaptive secure group communication protocols as well as the design of sophisticated protocols resilient to denial-of-service (DoS) attacks. Maude has also provided the basis for a formal specification of internet browsers whose model checking uncovered new attack scenarios, and then to design secure browsers. Concerning access control, Tom has been used to generate Java monitoring code for access control policies, which are previously analyzed by means of narrowing-based techniques. Finally, rewriting logic semantic of programming languages (see Section 3.4) have been used to study security properties of code at different levels of abstraction.
Probabilistic systems

Probabilistic rewrite theories (see Section 2.11) and statistical model-checking have been used to model and formally analyze adaptive designs of energy-constrained distributed embedded systems that can provide desired quality-of-service guarantees, such as multimedia applications and wireless sensor networks. The same techniques have also been applied to the specification and verification of distributed object-based stochastic hybrid systems. Another important application of probabilistic systems is the analysis of DoS-resistant network protocols. Finally, PMaude is a rewrite-based specification language for probabilistic object systems, while PVEStA is a parallel statistical model checking and quantitative analysis tool; both of them have been used in applications in this area.

6.6. Bioinformatics and chemical systems

On the bioinformatics side, biological cells can be modeled as concurrent systems whose transitions are precisely their biochemical reactions; this gives rise to rewriting-based symbolic models of biological systems which can be analyzed like any other rewrite theory. Following this basic idea, the Pathway Logic researchers at SRI International have used rewriting logic to develop sophisticated analyses of cell behavior in biological pathways, and have built useful notations and visualization tools, such as the Pathway Logic Assistant to represent the Maude-based analyses in forms more familiar to biologists. A related recent development is the use of rewriting logic in bioinformatics, describing models of neurons and their interconnections. Concerning chemical systems, ELAN and Tom have been applied to the development of symbolic representations of chemical graphs and to the modeling of chemical reactions as term rewriting modulo associativity and commutativity, guided by appropriate strategies to identify the dynamic constraints satisfied by chemical processes.
This section collects the proceedings in which two or more of the previous papers have been published. These include the proceedings of the biennial Workshop on Rewriting Logic and its Applications, whose first meeting took place in Asilomar, California, in 1996.

Despite the generality of the previous areas, it has been impossible to classify in them the following two papers, which apply rewriting logic in areas outside the more typical system specification, like cohomology or game theory.

6.8. Others

7. Proceedings

This section collects the proceedings in which two or more of the previous papers have been published. These include the proceedings of the Sixth International Workshop on Rewriting Logic and its Applications, whose first meeting took place in Asilomar, California, in 1996.