Reflection in Rewriting Logic and its Applications in the Maude Language

Manuel Clavel and José Meseguer
Computer Science Laboratory
SRI International, Menlo Park CA 94025

1 Introduction

Reflection is a system's ability to represent and control its own metalevel. In this way, very powerful techniques to extend, adapt, and modify a system become available. Many researchers have recognized the great importance and usefulness of reflection in programming languages [39, 38, 41, 40, 18, 14, 29], in theorem-proving [42, 5, 36, 15, 1, 28, 11, 13], in concurrent and distributed computation [27, 33, 35], and in many other areas such as compilation, programming environments, operating systems, fault-tolerance, and databases (see [37, 17] for recent snapshots of research in reflection).

In spite of being a very important topic, the semantic foundations of reflection are not yet well understood. For the case of reflection in logical languages we have recently proposed a general axiomatic notion of reflective logic [8]. Our axioms are based on the theory of general logics [30]; they are formulated in a logic-independent way that makes the reflective logic an explicit parameter and does not depend on the particular details of each logic.

The key concept in our axiomatization of reflective logics is the notion of a *universal theory*, that is, a theory U that can simulate the deductions of all other theories in a class $\mathcal{C}$ of theories of interest. In particular, if U is one of the theories in the class $\mathcal{C}$, then U can simulate its own metalevel at the object level, and this process can be iterated *ad infinitum*, giving rise to a "reflective tower."

In [8] several examples of reflective logics, including the lambda calculus and Turing machines, are discussed. However, our original motivation for studying reflective logics has come from our work on rewriting logic and our desire of endowing the Maude language with reflective features in a rigorous way. In fact, we have proposed a specific universal theory for rewriting logic in [8] and have proved it to be correct [9], i.e., that it satisfies all the formal requirements that we have stated for a reflective logic in [8]. We explain below what has been shown.

Reflection is an enormously powerful and useful property in practice. Therefore, a key aspect of our research, along with the development of the theoretical foundations, has been exploiting reflection in a wide range of applications, using the Maude language [7], which implements rewriting logic and supports its reflective capabilities, as a tool for developing those applications.

In particular, we survey in this paper applications of reflection in rewriting logic and Maude to the following areas:

- *Logical framework applications*, to make executable the representations of many other logics within the logical framework of rewriting logic.

- *Modularity applications*, allowing the semantic definition and implementation of an open and extensible *module algebra* to combine program modules in very powerful ways.
- *Internal strategies.* Thanks to reflection, execution strategies can be expressed *inside* rewriting logic, where they are formally defined and executed by rewrite rules that operate on metalevel entities.
- *Formal environments.* Using reflection, it becomes very easy to specify and build formal environments, providing theorem-proving and formal analysis support for a logically-based specification or programming language. The idea is that the inference systems that these tools implement are themselves rewrite theories; and the fact that such tools operate on metalevel entities —including the object theories that they verify or formally analize— is naturally supported by the reflective features.
- *Executable specification of reflective languages*, that can be formally specified in rewriting logic and can then be executed as interpreters.

2 A Universal Theory for Rewriting Logic

A reflective logic is a logic in which important aspects of its metatheory can be represented at the object level in a consistent way, so that the object-level representation correctly simulates the relevant metatheoretic aspects. Two obvious metatheoretic notions that can be so reflected are theories and the entailment relation $\vdash$. This leads us to the notion of a universal theory. However, universality may not be absolute, but only relative to a class $\mathcal{C}$ of *representable* theories. Typically, for a theory to be representable at the object level, it must have a finitary description in some way—say, being recursively enumerable—so that it can be represented as a piece of language.

In rewriting logic sentences are sequents of the form $t \longrightarrow t'$. Therefore, entailmenst are of the form

$$T \vdash t \longrightarrow t'$$

where T is a rewrite theory and t, t' terms in such a theory. We denote by $sen(T)$ the set of all sequents of the form $t \longrightarrow t'$ with t, t' terms in T.

The class $\mathcal{C}$ of theories of interest will be the class of all *finitely presentable* rewrite theories. That is, theories whose signature, set of equations, and set of rewrite rules are all finite. Our proof treat the case of unsorted and unconditional rewrite theories. However, the general techniques can be extended to typed versions and to conditional rules.

We specify a finitely presentable rewrite theory U and prove that it is $\mathcal{C}$-*universal* for $\mathcal{C}$ the class of finitely presentable rewrite theories. By our general axioms for reflection this makes rewriting logic $\mathcal{C}$-*reflective*.

Specifically, $\mathcal{C}$-universality means that there is a function, called a *representation function*,

$$\overline{(\_ \vdash \_)} : \bigcup_{T \in \mathcal{C}} \{T\} \times sen(T) \longrightarrow sen(U)$$

such that for each $T \in \mathcal{C}$, $\varphi \in sen(T)$,

$$T \vdash \varphi \iff U \vdash \overline{T \vdash \varphi}.$$

We can expand this equivalence further. For φ the sentence $t \longrightarrow t'$, then the U-sentence $\overline{T \vdash \varphi}$ takes the form $\overline{T}@\overline{t} \longrightarrow \overline{T}@\overline{t'}$, where $\overline{T}$ and $\overline{t}$ are the U-terms representing T and t, and where @ is a binary operator in U. Therefore, we get the expanded equivalence

$$T \vdash t \longrightarrow t' \iff U \vdash \overline{T}@\overline{t} \longrightarrow \overline{T}@\overline{t'}.$$

In [9] we have provided a detailed proof of this universality, showing that indeed a sentence is provable in an object theory T if and only if the corresponding representation of the entailment is provable in the universal theory U.

Note that, since U itself is finitely presentable and therefore belongs to the class $\mathcal{C}$ of representable theories, the representation process can be iterated, so that we immediately have a "reflective tower"

$$T \vdash \varphi \iff U \vdash \overline{T \vdash \varphi} \iff U \vdash \overline{U \vdash \overline{T \vdash \varphi}} \ \ldots$$

3 Exploiting Reflection

The fact that rewriting logic is reflective opens up promising possibilities both conceptually and as a guide to powerful and extensible language implementations. For example, since the Maude system is precisely a way to implement any finitely specifiable rewrite theory, this means that the Maude system is in an adequate sense an implementation of the universal theory. Making this explicit in the implementation through a module META-LEVEL [10] provides powerful ways of accessing metalevel entities—for example modules and proofs—at the object level, that is, as first class citizens within the logic itself.

The following areas seem particularly promising for exploiting the power of reflection in rewriting logic:

Reflective logical and semantic framework uses Rewriting logic is a logical framework in which many different logics, languages, and systems can be naturally represented. This is very useful for executable specification, rapid prototyping, and formal reasoning. A substantial amount of evidence, advocating the suitability of rewriting logic for these purposes already exists [31, 24, 19, 25].

In [26] a very general method for reifying inside rewriting logic representation maps between others logic and rewriting logic is proposed. This reification takes the form in Maude of an equationally defined function between the abstract data type representing the finitary theories of the logic being mapped and the abstract data type representing the finitary theories of rewriting logic (already defined in META-LEVEL). A complete specification of a reified map between full quantifier-free first-order linear logic and rewriting logic is given in [26] and has already been implemented in Maude.

Modularity Perhaps one of the most important new contributions of Maude is the *metaprogramming methodology* that it supports in a simple and powerful way. This methodology is well integrated with the language's semantic foundations, particularly with its logical foundations for reflection.

By "metaprogramming" we of course mean the capacity of defining programs that operate on other programs as their data; in our case, equational and rewrite theories that operate on other such theories as their data. By observing that we can not only reify theories, but also views among them, this includes the more traditional "parameterized programming" capabilities in the Clear-OBJ tradition [6, 12] as a particular instance. The difference is that in that tradition theories are metalevel entities not accessible at the object level of the logic, since this is only possible in an explicitly reflective logical context.

What reflection accomplishes is to *open up to the user* the metalelevel of the language, so that instead of having a fixed repertoire of parameterized programming operations we can now define a much wider range of theory-transforming and theory-combining operations that could not be defined using more traditional means. In summary, what reflection makes possible in Maude is the definition of an open, extensible, and user-definable *module algebra* supporting a new style of metaprogramming with very promising advantages for software methodology.

Reflection and internal strategies The most general Maude modules are system modules. They specify the initial model of a rewrite theory $\mathcal{R}$ [31]. This initial model is a transition system whose states are equivalence classes $[t]$ of ground terms modulo the equations E in $\mathcal{R}$, and whose transitions are proofs $\alpha : [t] \longrightarrow [t']$ in rewriting logic—that is, concurrent computations in the system so described. Such proofs are equated modulo a natural notion of proof equivalence that computationally corresponds to the "true concurrency" of the computations.

Consider for example a system module `NIM` specifying a version of the game of Nim. There are two players and two bags of pebbles: a "draw" bag to remove pebbles from, and a "limit" bag to limit the number of pebbles that can be removed. The two players take turns making moves in the game. At each move a player draws a nonempty set of pebbles not exceeding those in the limit bag. The limit bag is then readjusted to contain the least number of pebbles in either the double of what the player just drew, or what was left in the draw bag. The game then continues with the two bags in this new state. The player who empties the draw bag wins. An intermediate move is axiomatized by the rule `[mv]`; the last, winning move is axiomatized by the rule `win`.

```
mod NIM is
  protecting BOOL .
  sorts Pebble Bag State .
  subsorts Pebble < Bag .
  op o : -> Pebble .
  op nil : -> Bag .
  op __ : Bag Bag -> Bag [assoc comm id: nil] .
  op _=<_ : Bag Bag -> Bool .
  op least : Bag Bag -> Bag .
  op state : Bag Bag -> State .
  vars X Y Z : Bag .
  eq nil =< X = true .
  eq o X =< nil = false .
  eq o X =< o Y = X =< Y .
  eq least(X,Y) = if X =< Y then X else Y fi .
  crl [mv] : state(X Y,Z) => state(Y,least(X X,Y))
```

```
                                if X =< Z and X =/= nil .
  crl [win] : state(X,Y) => state(nil,nil)
                               if X =< Y and X =/= nil .
endm
```

The initial model described by this module is the transition system containing exactly all the possible game moves allowed by the game. But there are many bad moves that would allow the other player to win. A good player should avoid such bad moves by having a *winning strategy*. With such a strategy, each move made by the player inexorably leads to success, no matter what moves the other player attempts.

What we obviously want, in this and in many other examples, is to have good ways of controlling the rewriting inference process—which in principle could go in many undesired directions—by means of adequate *strategies*. Many systems, for example theorem provers and declarative languages implementations, support certain strategies of this nature. However, such strategies are often *external* to the languages they control: they may constitute a separate programming language external to the logic, or may be part of the language's "extralogical features." In Maude, thanks to the reflective capabilities of rewriting logic, strategies can be made *internal* to rewriting logic [8, 9]. That is, they can be defined by rewrite rules, and can be reasoned about as with rules in any other theory. For example, a winning strategy for the above game of Nim can be easily specified in this way [7]. The value of specifying strategies with rewrite rules is also emphasized in the most recent work on ELAN [4].

In fact, there is great freedom for defining many different strategy languages inside Maude. This can be done in a completely user-definable way, so that users are not limited by a fixed and closed strategy language. Also, even if some users decide to adopt a particular strategy language because of its good features, such a language remains fully *extensible*, so that new features and new strategy concepts can be defined on top of them. Of course, such languages should be defined in a disciplined way that guarantees that they are correct, that is, that they only produce valid rewrites. In [9] a method for defining and proving correct an internal strategy language for Maude based on the correctness of the universal theory is given. This method is very general and should apply to any reflective logical language.

In Maude, a strategy language is a *function on theories*, that assigns to a module M another module $strat(M)$, whose terms are called *strategy expressions* specifying desired, possibly quite complex, set of rewrite deductions in the original theory M. Executing such a strategy expression is simply rewriting it using the rules in $strat(M)$. In some cases, such executions may never terminate. However, as the expression is being rewritten, more and more of the desired rewrites in the theory M that the strategy expression in question was supposed to describe become directly "visible" in the partially rewritten strategy expression. In this way, we can tame the wildness of M by shifting our ground to a much more controllable theory $strat(M)$. For example, $strat(M)$ may be Church Rosser, and therefore essentially a functional module, so that computations of strategy expressions become essentially deterministic. This is of course not a necessary requirement, but it is nevertheless an attractive possibility in the context of a sequential implementation.

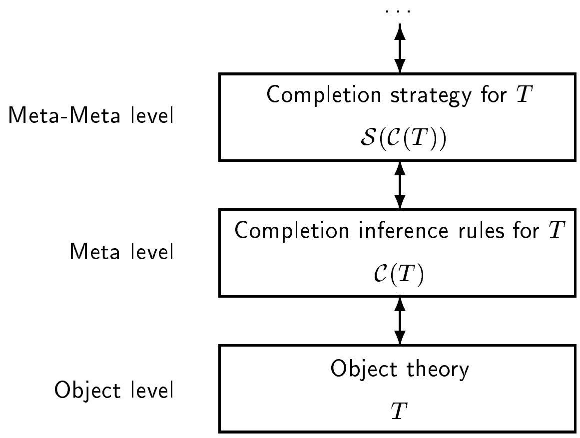
Formal environments Reflection can also be exploited to make the development and interoperation of new formal tools for rewriting logic, and for other logics, much easier. The idea is that, with the help of reflection, such tools can themselves be specified and prototyped in rewriting logic.

In [10] we explain in detail 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 whatsoever to the inference rules themselves. As it is well known, the Knuth-Bendix completion algorithm [21] attempts to construct a convergent rewrite system from a finite axiomatization of an equational theory. A more abstract view of completion as an inference system $\mathcal{C}$ was proposed by Bachmair, Dershowitz and Hsiang [2]. In this view, a (*standard*) *completion procedure* is any program that takes a set of equations and rules, $(E_0 \cup R_0)$, and a reduction ordering $\succ$ containing R_0, and uses some strategy to apply the inference rules of $\mathcal{C}$ to generate a derivation from $(E_0 \cup R_0)$. A completion procedure is called correct if all non-failing derivations result in a convergent system.

By proving the correctness of this more abstract notion of completion, Bachmair, Dershowitz and Hsiang were able to guarantee the correctness of a wide class of different completion procedures. A key element of their methodology is to cleanly separate the logical, or proof-theoretical, component of the completion method from issues pertaining to strategy and control. As we shall see, this is precisely what reflection and a convenient internal strategy language allow us to accomplish.

There have been several attempts to use this abstract view of completion in actual rewrite-rule implementations of completion procedures [22, 20]. The challenge has been to control the rewriting inference process generated by the rules in $\mathcal{C}$—which in principle could go in many undesired directions—by means of adequate *strategies*. Unfortunately, these implementations have not been able to preserve the separation between the logical and the control components of the completion method. For example, control of the rewriting process is deeply embedded in the data structure on which the inference rules operate. As a consequence, improvements in the control of the rewriting process require changes in the original data structure and consequently in the original inference system $\mathcal{C}$ as shown in [22]. In particular, proofs of correctness based on the original inference system need to be redone for the new inference rules combining inference and control.

We believe that the difficulties encountered in previous approaches are quite intrinsic and probably cannot be overcome without the use of new techniques such as reflection. Essentially, the possibility of allowing several levels of reflection is the key to a fully modular approach in which no change whatsoever is required in the original inference rules. This clean separation by levels is illustrated in the figure below. The theory T that we want to complete appears at the object level. The inference rules $\mathcal{C}$ are clearly metalevel rules defining a theory one level up. The key point is that the strategies controlling the application of such inference rules and therefore yielding different completion procedures appear as terms in the theory $\mathcal{S}(\mathcal{C}(T))$, corresponding to instantiating an internal strategy language $\mathcal{S}$ with the inference rules $\mathcal{C}$ as its object theory. Since $\mathcal{S}$ transforms any theory into a metalevel theory, and $\mathcal{C}$ is already a metalevel theory, $\mathcal{S}(\mathcal{C}(T))$ is a theory in the meta-metalevel. Keeping these three levels of reflection separate is the key to maintaining modularity. Of course, in other applications, involving for example meta-strategies that control the application of lower-level strategies, more than three levels of reflection may in fact be required.



Besides completion, we have experimented with several other examples, such as strategies for different rewriting logic interpreters (these are strategies for the universal theory $\mathcal{U}$), and several games and AI problem-solving examples. In all cases our solutions cleanly separate the declarative part, given by the rules of the object theory, from the control part, given by internal strategies. Pita and Martí-Oliet [34] report an interesting application of Maude reflection to specify the management of broadband telecommunication networks in which an internal strategy language again permits a clean and succint separation of the object specification from the control aspects.

The completion example briefly presented here is a first experiment within a more ambitious collaboration with the Cafe Project in Japan in which we plan to use reflection and internal strategies in Maude to build several theorem-proving tools for Cafe and Maude, including a Knuth-Bendix checker and an inductive theorem prover. The building of these tools will give us very useful additional experience on the use of reflection and internal strategies in Maude, and will go hand in hand with further extensions of the current Maude implementation to further enrich its reflective capabilities.

Executable specification of reflective languages Rewriting logic is a very general *semantic framework* in which many different languages and models of computation can be specified [31, 24, 32]. In particular, thanks to its reflective capabilities, it is also an attractive semantic framework to formally specify novel *reflective* languages. Since such specifications are *executable*, one obtains in this way not only a precise formal semantics for the language in question, but actually an interpreter that can be very useful for experimentation and exploratory design to add new features and capabilities to a language. In particular, as part of the IMSA project and in collaboration with researchers at JAIST and ETL, we have used Maude to give an executable formal specification of the GAEA reflective language [16]. In this way, the semantic framework of rewriting logic has proved very useful to develop new *formal models* of software architectures.

4 Concluding Remarks

A number of very interesting problems await further research, including the following:

- *Verification of internal strategies.* As already mentioned, one of the important benefits of having internal strategies is that they can be verified within the same logic of their object theories. This type of metalevel formal reasoning looks quite promising, but more experience and case studies are needed to demonstrate its usefulness.

- *User interaction and I/O.* We have already experimented with interactive strategies in some examples such as games. Such interactive strategies are entirely natural. What remains to be done is to provide a general methodology for defining and interacting easily with such strategies. We expect that the object-oriented capabilities of Maude will be very helpful in this regard.

- *Strategies on proofs.* As pointed out in [8, 9], the entire theory of reflective logics can similarly be developed in the richer case in which we are not only interested in entailments, but also in *proofs*. For strategy languages this opens up the possibility of defining internal strategy languages in which, instead of guiding the deduction process using only the properties of the intermediate or end results to which we arrive, we can in addition use properties of the proofs we develop to guide the deduction process. This will obviously lead to more expressive strategy languages whose usefulness should be studied and illustrated with examples.

- *Distributed strategies.* Since rewriting logic is naturaly concurrent, distributed deduction strategies are entirely natural objects of study. They are of interest for parallel implementations such as the rewrite rule machine [23] and will become even more important when a distributed Maude implementation is achieved. There are interesting connections with the work on distributed strategies by Bonacina and Hsiang [3] that should be further investigated.

- *Reflection and higher-order programming.* Note that, although rewriting logic is first-order, reflection provides what amount to powerful higher-order capabilities. The completion example already illustrates this. We plan to study other examples, such as applications in which several strategy levels can be used to obtain more sophisticated behavior, to further illustrate the "higher-order" power of using several levels of reflection.

Aknowledgements

We cordially thank Carolyn Talcott and Narciso Martí-Oliet for their very helpful suggestions and their encouragement of this work along many discussions. Our warm thanks also to Francisco Durán, Steven Eker, and Patrick Lincoln with whom we have exchanged many ideas about these topics, and with whom we have worked to realize them in the Maude language. We also thanks our colleagues Kokichi Futatsugi, Hiroshi Ishikawa, Hideyuki Nakshima, and Takuo Watanabe at JAIST and ETL for the very fruitful collaboration on the rewriting logic executable specification of the GAEA language.

References

[1] William E. Aitken, Robert L. Constable, and Judith L. Underwood. Metalogical frameworks II: Using reflected decision procedures. Technical Report, Computer Sci. Dept., Cornell University, 1993; also, lecture at the Max Planck Institut für Informatik, Saarbrücken, Germany, July 1993.

[2] L. Bachmair, N. Dershowitz, and J. Hsiang. Ordering for equational proofs. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, pages 346–357, 1986.

[3] Maria Paola Bonacina and Jieh Hsiang. Parallelization of deduction strategies: An analytical study. *Journal of Automated Reasoning*, 13:1–33, 1994.

[4] P. Borovanský, C. Kirchner, and H. Kirchner. Controlling rewriting by rewriting. This volume.

[5] R. S. Boyer and J Strother Moore. Metafunctions: proving them correct and using them efficiently as new proof procedures. In Robert Boyer and J Moore, editors, *The Correctness Problem in Computer Science*, pages 103–185. Academic Press, 1981.

[6] Rod Burstall and Joseph Goguen. The semantics of Clear, a specification language. In Dines Bjorner, editor, *Proceedings of the 1979 Copenhagen Winter School on Abstract Software Specification*, pages 292–332. Springer LNCS 86, 1980.

[7] Manuel Clavel, Steven Eker, Patrick Lincoln, and José Meseguer. Principles of Maude. In José Meseguer, editor, *Proceedings of the First International Workshop on Rewriting Logic*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 65–89. Elsevier, September 1996.

[8] Manuel Clavel and José Meseguer. Axiomatizing reflective logics and languages. In Gregor Kiczales, editor, *Proceedings of Reflection'96*, pages 263–288. Xerox PARC, 1996.

[9] Manuel Clavel and José Meseguer. Reflection and strategies in rewriting logic. In José Meseguer, editor, *Proceedings of the First International Workshop on Rewriting Logic*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 125–147. Elsevier, September 1996.

[10] Manuel Clavel and José Meseguer. Internal strategies in a reflective logic. In Bernhard Gramlich and Hélène Kirchner, editors, *Proceedings of the CADE-14 Workshop on Strategies in Automated Deduction*, pages 1–12, Townsville, Australia, July 1997.

[11] Fausto Giunchiglia, Paolo Traverso, Alessandro Cimatti, and Paolo Pecchiari. A system for multi-level reasoning. In *IMSA '92*, pages 190–195. Information-Technology Promotion Agency, Japan, 1992.

[12] Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. Technical Report SRI-CSL-92-03, SRI International, Computer Science Laboratory, 1996. To appear in J.A. Goguen, editor, *Applications of Algebraic Specification Using OBJ*, Cambridge University Press.

[13] John Harrison. Metatheory and reflection in theorem proving: a survey and critique. University of Cambridge Computer Laboratory, 1995.

[14] Patricia Hill and John Lloyd. *The Gödel Programming Language*. MIT Press, 1994.

[15] Douglas J. Howe. Reflecting the semantics of reflected proof. In Peter Aczel, Harold Simmons, and Stanley S. Wainer, editors, *Proof Theory*, pages 229–250. Cambridge University Press, 1990.

[16] Hiroshi Ishikawa, José Meseguer, Takuo Watanabe, Kokichi Futatsugi, and Hideyuki Nakashima. On the semantics of GAEA—an object-oriented specification of a concurrent reflective language in rewriting logic—. This volume.

[17] Gregor Kiczales, editor. *Proceedings of Reflection'96, San Francisco, California, April 1996*. Xerox PARC, 1996.

[18] Gregor Kiczales, Jim des Rivieres, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.

[19] C. Kirchner, H. Kirchner, and M. Vittek. Designing constraint logic programming languages using computational systems. In V. Saraswat and P. van Hentryck, editors, *Principles and Practice of Constraint Programming: The Newport Papers*, pages 133–160. MIT Press, 1995.

[20] H. Kirchner and P. E. Moreau. Prototyping completion with constraints using computational systems. In J. Hsiang, editor, *Proceedings 6th Conference on Rewriting Techniques and Applications*, volume 914 of *Lecture Notes in Computer Science*, pages 438–443. Springer-Verlag, 1995.

[21] Donald Knuth and P. Bendix. Simple word problems in universal algebra. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[22] P. Lescanne. Completion procedures as transition rules + control. In M. Diaz and F. Orejas, editors, *TAPSOFT'89*, volume 351 of *Lecture Notes in Computer Science*, pages 28–41. Springer-Verlag, 1989.

[23] Patrick Lincoln, José Meseguer, and Livio Ricciulli. The Rewrite Rule Machine Node Architecture and its Performance. In B. Buchberger and J. Volkert, editors, *Proceedings of CONPAR'94*, pages 509–520, 1994.

[24] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. Technical Report SRI-CSL-93-05, SRI International, Computer Science Laboratory, August 1993. To appear in D. Gabbay, ed., *Handbook of Philosophical Logic*, Kluwer Academic Publishers.

[25] Narciso Martí-Oliet and José Meseguer. General logics and logical frameworks. In D. Gabbay, editor, *What is a Logical System?*, pages 355–392. Oxford University Press, 1994.

[26] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. In José Meseguer, editor, *Proceedings of the First International Workshop on Rewriting Logic*, volume 4, pages 189–224, Asilomar (California), September 1996. Electronic Notes in Theoretical Computer Science.

[27] Satoshi Matsuoka, Takuo Watanabe, Yuuji Ichisugi, and Akinori Yonezawa. Object-oriented concurrent reflective architectures. In M. Tokoro, O. Nierstrasz, and P. Wegner, editors, *Object-Based Concurrent Computing*, pages 211–226. Springer LNCS 612, 1992.

[28] Seán Matthews. Reflection in logical systems. In *IMSA'92*, pages 178–183. Information-Technology Promotion Agency, Japan, 1992.

[29] François-Nicola Demers and Jacques Malenfant. Reflection in logic, functional and object-oriented programming: a short comparative study. In *IJCAI '95 Workshop on Reflection and Metalevel Architectures and their Applications in AI*, pages 29–38, August 1995.

[30] José Meseguer. General logics. In H.-D. Ebbinghaus et al., editor, *Logic Colloquium'87*, pages 275–329. North-Holland, 1989.

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

[32] José Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In *Proceedings of the CONCUR'96 Conference, Pisa, August 1996*. Springer LNCS, 1996.

[33] Hideaki Okamura, Yutaka Ishikawa, and Mario Tokoro. AL-1/D: A distributed programming system with multi-model reflection framework. In *IMSA'92*, pages 36–47. Information-Technology Promotion Agency, Japan, 1992.

[34] Isabel Pita and Narciso Martí-Oliet. Using reflection to specify transaction sequences in rewriting logic. In *Proceedings of APPIA-GULP-PRODE*, 1997.

[35] Luis H. Rodriguez, Jr. A study on the viability of a production-quality metaobject protocol-based statically parallelizing compiler. In *IMSA'92*, pages 107–112. Information-Technology Promotion Agency, Japan, 1992.

[36] N. Shankar. *Metamathematics, Machines, and Gödel's Proof*. Cambridge University Press, 1994.

[37] Brian Smith and Akinori Yonezawa, editors. *Proc. of the IMSA'92 International Workshop on Reflection and Meta-Level Architecture, Tokyo, November 1992*. Research Institute of Software Engineering, 1992.

[38] Brian C. Smith. Reflection and Semantics in Lisp. In *Proc. POPL'84*, pages 23–35. ACM, 1984.

[39] G. L. Steele, Jr. and G. J. Sussman. The art of the interpreter or, the modularity complex. Technical Report AIM-453, MIT AI-Lab, May 1978.

[40] Valentin F. Turchin. The concept of a supercompiler. *ACM Transactions on Programming Languages and Systems*, 8(3):292–325, 1986.

[41] M. Wand and D.P. Friedman. The mystery of the tower revealed. *Lisp and Symbolic Computation*, 1(1):11–38, 1988.

[42] Richard W. Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, 13:133–170, 1980.