

Formal Interoperability*

José Meseguer
Computer Science Laboratory
SRI International
Menlo Park, CA 94025, USA
`meseguer@csl.sri.com`

1 Introduction

At present, formal methods for software specification and verification are *monolithic*, in the sense that in each approach only one formal system or specification language is used to formalize the desired system properties. For this reason, although formal approaches are very useful at their appropriate level of abstraction, their formal systems, and the tools based on them, are as it were *autistic*, because they lack the theoretical basis for being related to other formalisms and to their supporting tools.

As a consequence, it is at present extremely difficult to integrate in a rigorous way different formal descriptions, and to reason across such descriptions. This situation is very unsatisfactory, and presents one of the biggest obstacles to the use of formal methods in software engineering because, given the complexity of large software systems, it is a fact of life that no single perspective, no single formalization or level of abstraction suffices to represent a system and reason about its behavior.

In practice we find ourselves in constant need of *moving back and forth* between different formalizations capturing different aspects of a system. For example, as pointed out by Walter and Bellman [55], in the simulation area this need is felt very strongly because of the different types of mathematical models and corresponding constraints that a simulation must satisfy. Similarly, in a large software system we typically have very different requirements, such as functional correctness, proper real time behavior, concurrency, security, and fault-tolerance, which correspond to different views of the system and that are typically expressed in different formal systems. Often these requirements affect each other, but it can be extremely difficult to reason about their mutual interaction, and no tools exist to support such reasoning.

We need mathematical foundations, (meta-)formal methods, and tools to achieve

Formal Interoperability

that is, the capacity to move in a mathematically rigorous way across the different formalizations of a system, and to use in a rigorously integrated way the different tools supporting such formalizations.

Since the most basic need appears at the level of relating and integrating different mathematical formalisms, the foundations required must be *metamathematical* in nature. That is, they must provide rigorous foundational answers to the questions:

- *What is a logical formalism?*
- *How can different formalisms be related?*

Answers to such questions can provide a rigorous basis on which to build very powerful and widely applicable *meta-tools* to support the formal interoperation of different formalisms and their associated tools. Meta-tools of this kind may include:

- **meta-logical frameworks**, in which many different logics and specification languages can be defined, executed, and interoperated.
- **meta-logical translators**, in which rigorous translations between any given pair of logical formalisms can be defined and automated. This can be extremely useful to vastly increase the applicability of tools, for example to use a theorem prover for one logic to prove theorems in a different logic.

*Supported by Office of Naval Research Contracts N00014-95-C-0225, N00014-96-C-0114, and NSF Grant CCR-9633363.

- **meta-logical module calculus tools**, which given specifications in an arbitrary logic, or in several logics, can support powerful parameterization and module composition operations to combine those specifications in a highly modular and reusable way.

In this paper I briefly sketch recent work on meta-logical foundations that seems promising as a conceptual basis on which to achieve the goal of formal interoperability. Specifically, I will briefly discuss:

- The theory of *general logics* [39], and the associated notion of *map* between logics.
- The notion of a *meta-logical framework*, and the use of *rewriting logic* [32, 40] for this purpose.
- The idea of a *meta-logical module calculus* to combine specifications across different formalisms.
- The concept of *reflection* in general logics and in rewriting logic [8, 7] as a very powerful technique to achieve formal interoperability.

At the mathematical level, category theory [31] provides very good methods to articulate all these concepts. In fact, this is entirely natural, since formal interoperability has essentially to do with mappings between different logical systems. Therefore, I also sketch some ways in which category theory is very useful as a foundation.

2 General Logics

The theory of *general logics* first proposed in [39] provides axiomatic notions formalizing the different aspects of a logic and of their combinations into fuller notions of logic. The theory is in a sense modular, in that it permits focusing on different aspects of a logic—such as the provability relation, the models, or the proofs themselves—as well as combining several aspects together, depending on the particular applications. These different aspects are axiomatized by means of the following basic concepts:

- *an entailment system* axiomatizes the consequence relation of a logic
- *an institution* (a notion due to Goguen and Burstall [18]) covers the model-theoretic aspects of a logic, focusing on the notion of validity
- *a logic* is obtained by combining an entailment system and an institution
- *a proof calculus* enriches an entailment system with an actual proof theory
- *a logical system* is a logic with a choice of a proof theory for it

A key notion is that of a *mapping* translating one logic into another

$$\mathcal{L} \longrightarrow \mathcal{L}'$$

that preserves whatever logical properties are relevant, such as provability of formulas, or satisfaction of a formula by a model. Such mappings allow us to relate in a rigorous way different logics, to combine different formalisms together, and to explore new logics for computational purposes.

We can summarize the basic conceptual standpoint of the theory of general logics by stating that:

No logical formalism (specification language, prototyping language, etc.) will be best for all purposes. What exists is a space of possibilities (the universe of logics) in which careful choice of the formalisms that best suit some given purposes can be exercised.

What is crucial in practice is to have correct and flexible methods for changing our point of view, so that we can move from one model or formalism to another. This is what maps of logics make possible.

The definition and automation of maps of this kind could be supported by tools such as a *meta-logical translator* that, when supplied with the formal definition of a given map of logics, would implement such a map. Such a meta-tool could be very useful to enable the formal interoperation of tools in different formal systems. For example, it could be used to automatically translate specifications in one logic into corresponding specifications in another for which a theorem prover exists. Then, if the map has the required properties—namely, if it is *conservative* [39]—such a theorem prover could be used to mechanically prove theorems about the source specification.

3 Meta-Logical Frameworks and Rewriting Logic

A *meta-logical framework* \mathcal{F} is a “universal logic” into which a wide variety of other logics can be faithfully represented by means of conservative maps of logics

$$\mathcal{L} \longrightarrow \mathcal{F}.$$

A number of logics have been proposed in the recent literature as (meta-)logical frameworks of this kind. Several of these proposals offer some variant of higher-order type theory as the framework logic (see [33] for references). A simpler alternative recently proposed by Martí-Oliet and Meseguer [32, 33] is to use *rewriting logic* [40] as a very flexible and natural meta-logical framework.

In rewriting logic axioms are rewrite rules of the form

$$t \rightarrow t'$$

that can be applied *modulo* given structural axioms specified with the syntax. Such rewrite rules have both a logical and a computational reading. Logically, each rewriting step is a logical *entailment* in a formal system. Computationally, each rewriting step is a *parallel local transition* in a concurrent system. Therefore, rewriting logic can be used both as a *meta-logical framework* in which to represent other logics, and as a *semantic framework* for defining and prototyping computational systems and languages.

Our experience so far indicates that many logics, including for example:

- first-order logic;
- intuitionistic logic;
- linear logic;
- equational logic;
- Horn logic with equality; and
- any logic having a sequent calculus presentation;

can be represented in a very simple and natural way in rewriting logic, and—using an implementation of rewriting logic—could be prototyped and executed using their corresponding representations. Indeed, we conjecture that any effectively presented logic—that is any logic of practical computational interest—could be represented and prototype in rewriting logic in this way.

Due to its great flexibility for representing computational systems, including concurrent ones, rewriting logic can also be used as a very flexible *semantic framework* in which many different languages and systems, such as for example:

- CCS, the π -calculus, and LOTOS;
- Dataflow, Graph Rewriting, and Neural Networks;
- Petri nets;
- Actors;
- the UNITY language;
- functional languages;
- object-oriented languages; and
- conventional languages;

can be given a precise semantics and can be prototyped and executed.

The use of rewriting logic as a meta-logical and semantic framework for applications such as those described above can be supported by tools such as a *rewriting logic interpreter*. In this way, it is possible to combine different logics by combining their rewriting logic representations, and to executably interoperate such logics in combination. Similarly, it is also possible to combine and to interoperate executable specifications of different languages and systems.

We are working on one such rewriting logic interpreter for the Maude language [41, 7], and two other rewriting logic-based language implementation efforts—the Cafe [15] and the ELAN [3] systems—are being actively pursued in Japan and France. Furthermore, by choosing an adequate subset of the logic, such as the Simple Maude sublanguage, and using adequate program transformations [29] to move specifications into this subset, it is possible to efficiently compile such a subset of rewriting logic on a wide variety of parallel architectures [30].

4 A Meta-Logical Module Calculus

Another very important fruit of the theories of general logics [39] and of institutions [18] is that, for each logic, they provide a general notion of *theory*—that can be used to formally specify some aspect or component of a software system in that logic—and of *interpretation* or *view* from one theory into another. Using maps of logics it is also possible to relate theories—and therefore specifications—in different logics.

The specification of large systems should be done in a highly modular and parameterized way. Using theory composition operations that can be defined in a logic-independent manner it is possible to develop a very powerful *module calculus* for combining specifications in this way.

Much work, beginning with the ideas of Goguen and Burstall on institutions [18], has already been done in this area, and extremely encouraging experience already exists in this regard in specification languages such as Clear [5], OBJ [20], ACT [12], Eqlog [19], and Maude [41]. However, much work remains to be done in order to:

- extend such a module calculus with powerful new operations;
- support theory composition operations across different logics;
- generalize such a module calculus from specifications to entire software modules¹ (in a style similar to “wrappings” [27]) to support the composition and semantic interoperation of software modules.

A very useful tool would be a *meta-logical module calculus system* that could be instantiated to compose specifications in any logical formalism, or across such formalisms, and the further extension of such a tool to one for entire software modules. In this way, specification languages, and also programming languages, could be systematically endowed with powerful module compositionality operations with a dramatic increase in the reusability of their modules and in the reliability of their semantic-based compositions.

Furthermore, such a module calculus does not have to be reduced to a fixed repertoire of module operations, such as for example colimits of theories. By using reflective techniques that bring a data type of modules within the object level of a logic, module operations become both extensible and user-definable, opening an unlimited range of possibilities for module composition and transformation.

5 Reflection

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 [52, 51, 56, 54, 26, 23, 38], in theorem-proving [57, 4, 49, 24, 1, 36, 16, 22], in concurrent and distributed computation [35, 44, 46], and in many other areas such as compilation, programming environments, operating systems, fault-tolerance, and databases (see [50, 25] for recent snapshots of research in reflection). In particular, reflective methods are enormously useful to achieve formal interoperability.

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 Manuel Clavel and I have recently proposed a general axiomatic notion of reflective logic [8]. Our axioms are based on the theory of general logics and 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.”

¹Very encouraging experience on extensions of this kind such as LIL [17] and LILEANNA [53] already exists.

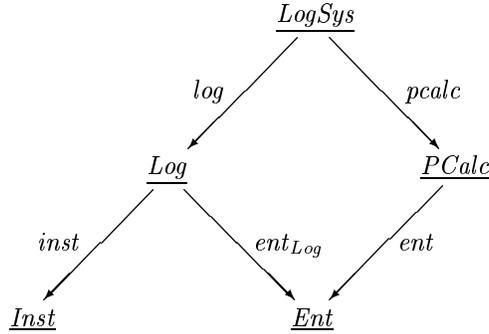


Figure 1: Categories of logical structures and their relationships.

In [8] several examples of reflective logics, including the lambda calculus and Turing machines, are discussed. And in [9] Manuel Clavel and I have shown that a specific universal theory for rewriting logic makes it reflective. This is exploited by the Maude language implementation, that efficiently supports reflective rewriting logic computation.

We are currently exploiting reflection in a wide range of applications using Maude. All these applications are directly relevant for formal interoperability. They include:

- Logical framework applications: in a reflective logical framework like rewriting logic, one can define and execute maps between logics withing the framework itself [34].
- Modularity applications: by reflection, theories—corresponding to modules—become a data type inside the logic on which a very open and extensible *module algebra* can be defined.
- 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 [9, 10].
- 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 [10].

6 Categories Everywhere

The theory of general logics [39] extends the theory of institutions [18] and is related to other notions proposed in the literature, including [2, 47, 11, 48, 37, 45, 14, 13, 21] (see [42] for a discussion of work in this area).

All these approaches, beginning with institutions, use category theory as a foundation in an essential way. First of all, specifying a logic, or some component of it like an institution or a proof calculus, amounts to specifying a number of categories, functors, and natural transformation. For example, syntax is specified by a category of signatures; sentences are given by a functor from signatures to sets; models for each signature also form a category; and the process of associating to each signature its category of models is a (contravariant) functor from signatures to the category of categories. The use of categories becomes even more intense when each model of a theory in the logic is itself a category with structure, in which the theory is interpreted as a structure-preserving functor from the free category generated by the theory. This is the case of *categorical logics* [39], of which Lawvere’s functorial semantics of algebraic theories is a paradigmatic example [28].

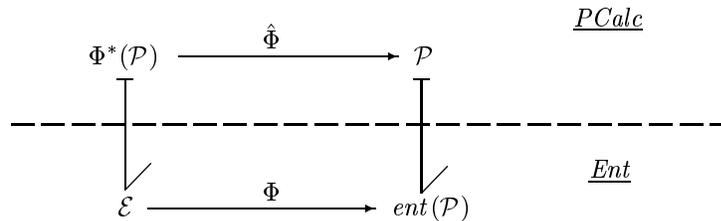
Given a logic, or an appropriate component of it, one can then define the category of *theories* in that logic—that can be used as specifications or even as declarative programs, and therefore provide the appropriate notion of *module*. Theories are typically described as pairs consisting of a signature and a set of sentences—the theory’s axioms, or its theorems, depending on the particular formulation. One of the fundamental contributions of the theory of institutions has been to show that important theory composition operations, including parameterized constructions, can be given a precise categorical semantics as colimit constructions in the category of theories. As already mentioned in Section 4, this semantics has then been exploited to provide module operations in a number of specification languages in the Clear tradition, including [5, 20, 12, 19, 41].

More important yet, for the purposes of formal interoperability, are the maps *between* different logic, which allow us to translate one logic into another. Such maps—preserving the appropriate logical structure in a logic or in a

logic’s component—also form categories in a natural way. For example, in the theory of general logics, entailment systems and entailment-preserving maps form a category \underline{Ent} , and we have similar categories \underline{Inst} , \underline{Log} , \underline{PCalc} , and \underline{LogSys} of institutions, logics, proof calculi, and logical systems, with maps preserving the corresponding logical structure. These categories are themselves related by functors that forget the extra logical structure as shown in Figure 1.

The functors $inst$, log , ent , and (therefore also) $log;inst$ have right adjoints, and $inst$ has also a left adjoint [39]. Furthermore, Maura Cerioli and I have shown that all the above functors are *fibrations* [6]. This automatically gives a method for “borrowing” missing logical structure from one logic along a map, to endow another logic with the missing structure.

For example, consider the case where our source formalism is an entailment system \mathcal{E} and we wish to borrow a proof calculus structure from \mathcal{P} along a map $\Phi : \mathcal{E} \rightarrow ent(\mathcal{P})$ to endow \mathcal{E} with a proof calculus structure $\hat{\Phi}^*(\mathcal{P})$. By the fact that ent is a fibration, this can be done by means of a map $\hat{\Phi}$ in \underline{PCalc} that “lifts” to the richer context of proof calculi our original map Φ in a universal way and such that $ent(\hat{\Phi}) = \Phi$.



This borrowing construction is very useful as a logical interoperability technique. In particular, if \mathcal{P} is a logical framework, the universal property of the lifted map permits a very economical way of defining representation maps into the framework, since only the map of the underlying entailment systems has to be specified.

7 Concluding Remarks

Foundations should *not* be an academic exercise in describing past practice, some *autopsy* of the past. Their great *practical* importance is in *imagining the future*, in helping us find new, more powerful and general techniques and tools to specify, reason about, develop, evolve, and reuse software systems. This paper has argued that the present lack of (meta-)formal methods and tools to support the interoperation of formal methods and formal systems is one of the greatest obstacles blocking the practical applicability of formal techniques in software engineering. I have presented some ideas in software foundations that seem promising to reach the goal of formal interoperability, but much work remains ahead.

Acknowledgements

Besides having been stimulated by the original work of Goguen and Burstall on institutions and by other related work in the literature, many of these ideas have been developed as joint work with Narciso Martí-Oliet, Maura Cerioli, Manuel Clavel, and Carolyn Talcott. To all of them my warmest thanks. Besides the references already given, [43] is a related paper having also a direct bearing on formal interoperability.

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] E. Astesiano and M. Cerioli. Relationships between logical frameworks. In M. Bidoit and C. Choppy, editors, *Recent Trends in Data Type Specification*, volume 655 of *LNCS*, pages 126–143. Springer-Verlag, 1993.
- [3] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. 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>.

- [4] 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.
- [5] 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.
- [6] Maura Cerioli and José Meseguer. May I borrow your logic? (Transporting logical structure along maps). *Theoretical Computer Science*, 173:311–347, 1997.
- [7] Manuel G. Clavel, Steven Eker, Patrick Lincoln, and José 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>.
- [8] Manuel G. Clavel and José Meseguer. Axiomatizing reflective logics and languages. In Gregor Kiczales, editor, *Proceedings of Reflection'96, San Francisco, California, April 1996*, pages 263–288. Xerox PARC, 1996.
- [9] Manuel G. Clavel and José 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>.
- [10] Manuel G. Clavel and José Meseguer. Internal strategies in a reflective logic. In B. Gramlich and H. Kirchner, editors, *Proceedings of the CADE-14 Workshop on Strategies in Automated Deduction (Townsville, Australia, July 1997)*, pages 1–12, 1997.
- [11] H. Ehrig, M. Baldamus, and F. Orejas. New concepts of amalgamation and extension of a general theory of specifications. In M. Bidoit and C. Choppy, editors, *Recent Trends in Data Type Specification*, volume 655 of *LNCS*, pages 199–221. Springer-Verlag, 1993.
- [12] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1*. Springer-Verlag, 1985.
- [13] J. Fiadeiro and T. Maibaum. Generalising interpretations between theories in the context of (π -)institutions. In G. Burn, S. Gay, and M. Ryan, editors, *Theory and Formal Methods 93*, pages 126–147. Springer-Verlag, 1993.
- [14] J. Fiadeiro and A. Sernadas. Structuring theories on consequence. In D. Sannella and A. Tarlecki, editors, *Recent Trends in Data Type Specification*, pages 44–72. Springer LNCS 332, 1988.
- [15] K. Futatsugi and T. Sawada. Cafe as an extensible specification environment. In *Proc. of the Kunming International CASE Symposium, Kunming, China, November, 1994*.
- [16] 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.
- [17] J.A. Goguen. Reusing and interconnecting software components. *Computer*, 19(2):16–28, February 1986.
- [18] Joseph Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
- [19] Joseph Goguen and José Meseguer. Eqlog: Equality, types, and generic modules for logic programming. In Douglas DeGroot and Gary Lindstrom, editors, *Logic Programming: Functions, Relations and Equations*, pages 295–363. Prentice-Hall, 1986.
- [20] 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, 1997. To appear in J.A. Goguen, editor, *Applications of Algebraic Specification Using OBJ*, Cambridge University Press.
- [21] R. Harper, D. Sannella, and A. Tarlecki. Structure theory presentations and logic representations. *Annals of Pure and Applied Logic*, 67:113–160, 1994.
- [22] John Harrison. Metatheory and reflection in theorem proving: a survey and critique. University of Cambridge Computer Laboratory, 1995.

- [23] Patricia Hill and John Lloyd. *The Gödel Programming Language*. MIT Press, 1994.
- [24] 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.
- [25] Gregor Kiczales, editor. *Proceedings of Reflection'96, San Francisco, California, April 1996*. Xerox PARC, 1996.
- [26] Gregor Kiczales, Jim des Rivieres, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.
- [27] C. Landauer and K. Bellman. Integrated simulation environments. In *Proceedings of DARPA Variable-Resolution Modeling Conference, 5-6 May 1992, Herndon, Virginia*, 1993.
- [28] F. William Lawvere. Functorial semantics of algebraic theories. *Proceedings, National Academy of Sciences*, 50:869–873, 1963. Summary of Ph.D. Thesis, Columbia University.
- [29] Patrick Lincoln, Narciso Martí-Oliet, and José Meseguer. Specification, transformation, and programming of concurrent systems in rewriting logic. In G.E. Blelloch, K.M. Chandy, and S. Jagannathan, editors, *Specification of Parallel Algorithms*, pages 309–339. DIMACS Series, Vol. 18, American Mathematical Society, 1994.
- [30] Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Livio Ricciulli. Compiling rewriting onto SIMD and MIMD/SIMD machines. In *Proceedings of PARLE'94, 6th International Conference on Parallel Architectures and Languages Europe*, pages 37–48. Springer LNCS 817, 1994.
- [31] Saunders MacLane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.
- [32] 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.
- [33] 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.
- [34] Narciso Martí-Oliet and José 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>.
- [35] 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.
- [36] Seán Matthews. Reflection in logical systems. In *IMSA '92*, pages 178–183. Information-Technology Promotion Agency, Japan, 1992.
- [37] Brian H. Mayoh. Galleries and institutions. Technical Report DAIMI PB-191, Computer Science Dept., Aarhus University, 1985.
- [38] 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.
- [39] José Meseguer. General logics. In H.-D. Ebbinghaus et al., editor, *Logic Colloquium '87*, pages 275–329. North-Holland, 1989.
- [40] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
- [41] José Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In Gul Agha, Peter Wegner, and Akinori Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 314–390. MIT Press, 1993.
- [42] José Meseguer and Narciso Martí-Oliet. From abstract data types to logical frameworks. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Type Specification, Santa Margherita, Italy, May/June 1994*, pages 48–80. Springer LNCS 906, 1995.

- [43] José Meseguer and Carolyn Talcott. Reasoning theories and rewriting logic. Manuscript, Stanford University, June 1996.
- [44] 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.
- [45] A. Poigné. Foundations are rich institutions, but institutions are poor foundations. In H. Ehrig et al., editors, *Categorical Methods in Computer Science with Aspects from Topology*, volume 393 of *LNCS*, pages 82–101. Springer-Verlag, 1989.
- [46] 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.
- [47] A. Salibra and G. Scollo. Compactness and Löwenheim-Skolem properties in pre-institution categories. Technical Report LIENS-92-10, Laboratoire d'Informatique de l'Ecole Normale Supérieure, Paris, March 1992.
- [48] D. Sannella and A. Tarlecki. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Informatica*, 25:233–281, 1988.
- [49] N. Shankar. *Metamathematics, Machines, and Gödel's Proof*. Cambridge University Press, 1994.
- [50] 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.
- [51] Brian C. Smith. Reflection and Semantics in Lisp. In *Proc. POPL'84*, pages 23–35. ACM, 1984.
- [52] 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.
- [53] William Tracz. Formal specification of parameterized programs in LILEANNA. Manuscript, Version 7.0, 1993.
- [54] Valentin F. Turchin. The concept of a supercompiler. *ACM Transactions on Programming Languages and Systems*, 8(3):292–325, 1986.
- [55] D. Walter and K. Bellman. Some issues in model integration. In *1990 Eastern Simulation Conference*, 1990.
- [56] M. Wand and D.P. Friedman. The mystery of the tower revealed. *Lisp and Symbolic Computation*, 1(1):11–38, 1988.
- [57] Richard W. Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, 13:133–170, 1980.