A Reflective Framework for Formal Interoperability

José Meseguer and Carolyn Talcott

SRI International and Stanford University

Introduction

Given the complexity of large software and hardware 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.

Besides, the different views of a system do not fit neatly within a single refinement hierarchy. Instead, they form a *web* of views that typically involve very different requirements expressed in *different* formal systems—for example:

- architectural structure
- functional correctness
- real time behavior
- performance
- concurrency and distribution
- security
- fault-tolerance

Introduction (II)

The problem is: how to put the elephant back together

Introduction (III)

A serious difficulty comes from the fact that often these different and heterogeneous requirements *leak* across formalization boundaries. That is, each view of the system imposes *semantic requirements* on other views.

Therefore, in practice we find ourselves in constant need of *moving back and forth* between different formalizations capturing different aspects of a system to reason about their mutual interaction.

Although in particular cases this sometimes can be done, in general this is at present extremely difficult, because we lack:

- general (meta-)formal methodologies for integrating the different formalizations of a system

- meta-tools supporting such an integration

Introduction (IV)

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?*

Introduction (V)

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.

Furthermore, in a *reflective* logical framework it is possible to define such logical translations, to build new theorem proving tools, and to interoperate them with existing tools, all within the framework logic

About this Talk

In this talk we will briefly sketch recent work on meta-logical foundations that seems promising as a conceptual basis on which to achieve the goal of formal interoperability.

We will briefly discuss:

- The theory of *general logics*, and the associated notion of *map* between logics.

- The notion of a *meta-logical framework*, and the use of *rewriting logic* for this purpose.

- A general theory of *reflection*, yielding general methods for making formal systems and their tools and enviroments much more flexible, reusable, and extensible.

- The idea of a *meta-logical module calculus* to combine specifications and software modules across different formalisms.

- Our research plans to build tools and metatools realizing these ideas.

General Logics

The theory of genereral logics (Meseguer [10]) provides a modular meta-logical axiomatization of the different components of a logic by means of the following basic concepts:

- *an entailment system* axiomatizes the consequence relation of a logic
- *an institution* (Goguen and Burstall [7]) 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

Maps of Logics

Translations from one logic into another by *maps* of logics

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

that preserve the relevant logical structure permit *relating and combining logical formalisms together*. These maps can be used to "borrow" missing logical structure and tools from one logic for use in another (Cerioli and Meseguer [1]).

In fact, the philosophy of the theory of general logics is 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 can be instantiated to implement any such map of logics.

Meta-Logical Frameworks

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.

A simpler alternative recently proposed is *rewriting logic* (Martí-Oliet and Meseguer [9, 11]) in which 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 and can be prototyped and executed.

Meta-Logical Frameworks (II)

Rewriting logic can also be used as a very flexible *semantic framework* in which many different languages and systems, such as for example:

- architectural description languages;
- domain-specific languages;
- prototyping languages;
- parallel languages;
- functional languages;
- logic programming languages;
- object-oriented languages; and
- conventional languages;

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

In addition, the *interoperation* between such languages can also be naturally specified and prototyped in the semantic framework.

The Maude system is a *meta-logical framework interpreter* based on rewriting logic to execute representations of logics and languages.

Reflection

Reflection is a system's capacity to represent itself and to control its own behavior. Many researchers have recognized the great importance and usefulness of reflection in programming languages, theorem-proving, concurrent and distributed computation, compilation, programming environments, operating systems, fault-tolerance, and databases. However, the metalogical foundations of reflection have been missing.

The papers [5, 6, 2] provide metalogical foundations for reflection based on the theory of general logics, and show that rewriting logic is a reflective metalogical framework. In particular, rewrite theories are reified in an abstract data type `Module` in a *universal theory* that can simulate all other theories (including itself).

This can be exploited to define logical translations, and to build and interoperate new and already existing theorem proving tools in rewriting logic.

Reflection (II)

Suppose that we can represent a logic $\mathcal{L}$ with a finitary syntax and inference system within rewriting logic by means of an effective and conservative representation map

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

Then, using reflection we can define and execute maps Φ of this kind inside rewriting logic itself getting an equationally defined map

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

More generally, an effective map

$$\Phi : \mathcal{L} \longrightarrow \mathcal{Q}$$

between two logics $\mathcal{L}$ and $\mathcal{Q}$, can be reified as an equationally-defined function

$$\overline{\Phi} : \mathtt{Module}_{\mathcal{L}} \longrightarrow \mathtt{Module}_{\mathcal{Q}}$$

within rewriting logic.

Building Tools by Reflection

Theorem provers and other formal methods tools have underlying *inference systems* that can be naturally specified and prototyped in rewriting logic. Furthermore, using reflection the strategy aspects of such tools and inference systems can then be specified by rewriting strategies.

Using a high-performance reflective rewriting logic language like Maude it is possible to turn a theorem prover prototype into an implementation with competitive performance. We have recently developed an inductive theorem prover for membership equational logic, and a Church-Rosser checker for order-sorted equational specifications this way [3].

Building Tools by Reflection (II)

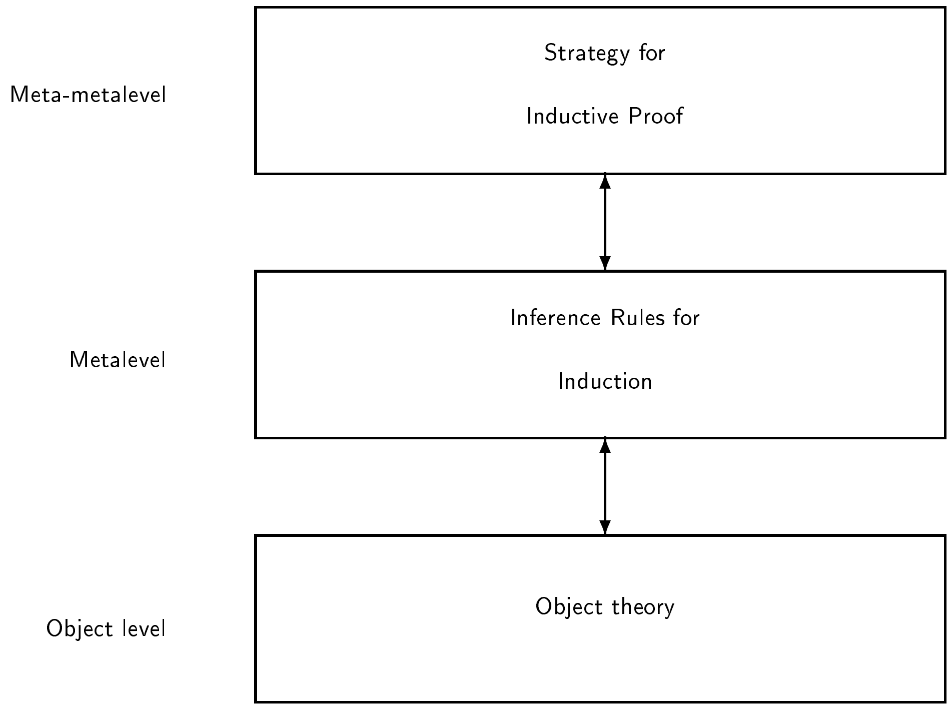
Reflection allows a very simple and modular design for such tools. Consider, for example, the design of the inductive theorem prover. Its purpose is to prove inductive properties of a Maude functional module T.

The rules of inference for induction can be naturally expressed as a rewrite theory $\mathcal{I}$. But since this rewrite theory uses T as a data structure—that is, it actually uses its representation $\overline{T}$—the theory $\mathcal{I}$ should be defined at the metalevel.

Proving an inductive theorem for T corresponds to applying the rules in $\mathcal{I}$ with some *strategy*. But since the strategies for any rewrite theory belong to the metalevel of such a theory, and $\mathcal{I}$ is already at the metalevel, we need *three levels* to make our design entirely *modular*.

Building Tools by Reflection (III)

This is illustrated by the following picture



The inductive Church-Rosser checker tool has a similar reflective design, but only needs two levels.

Meta-Logical Module Calculus

Present software technology provides quite limited support for reusing, adapting and modifying software systems and their components. New software foundations are needed to enable drastic improvements in software adaptability and reuse.

Such foundations must provide powerful and mathematically rigorous *composition mechanisms* to architecture, adapt, and reuse software systems. As it has been clearly pointed out by approaches such as "wrappings" [8], it is not only code, but also specifications and documentation that have to be composed together.

The theories of institutions [7] and of general logics [10] provide powerful formalism-independent notions of composition for which very encouraging practical experience has been obtained in specification languages such as Clear, OBJ, and Maude.

Meta-Logical Module Calculus (II)

Based on these ideas, but going considerably further, we envision a powerful *meta-logical module calculus* with the following characteristics:

- *Wide-Spectrum.* The module composition operations should simultaneously transform specifications, prototypes, documentation, and code.
- *Semantics-Based.* Formally stating the semantic requirements that must be met in order for the corresponding compositions to be correct.
- *Formalism-Independent.* So that it can be applied to different specification formalisms.
- *Language-Independent.* So that the calculus can be applied to many languages.
- *Extensible.* So that new module composition operations can easily be added to the calculus to support the widest possible variety of system adaptations and reuses.

In this way, specification and programming languages, could be systematically endowed with powerful module compositionality operations leading to a dramatic increase in reusability, adaptability, and reliability.

Research Plans

We plan to advance all these ideas and to build tools and meta-tools supporting them as follows:

- Building of a generic formal language translation meta-tool.

- Building of a more powerful inductive theorem prover for equational logic and for rewriting logic.

- Building of a Knuth-Bendix Completion tool, a coherence completion tool, and a temporal logic proof assistant.

- Interoperation of all these tools among themselves and with an existing temporal logic tool.

- Studies of general methods and implementation techniques for heterogeneous proof-checking and for a logic-independent module calculus.

Concluding Remarks

Integrating the different formalizations of a system in a correct and systematic way is a pressing and unavoidable problem for large systems.

This talk has presented some research directions leading towards the goal of having foundations, methods and tools supporting formal interoperability to achieve integration.

The Maude interpreter is an efficient implementation of rewriting logic that supports its reflective aspects. We plan to use Maude as a metatool to build other tools and metatools in support of formal interoperability.

References

[1] Maura Cerioli and José Meseguer. May I borrow your logic? (Transporting logical structure along maps). *Theoretical Computer Science*, 173:311–347, 1997.

[2] Manuel Clavel. Reflection in general logics, rewriting logic, and Maude. Ph.D. Thesis, University of Navarre, 1998.

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

[4] Manuel 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`.

[5] Manuel 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.

[6] Manuel 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`.

[7] Joseph Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[8] C. Landauer and K. Bellman. Integrated simulation environments. In *Proceedins of DARPA Variable-Resolution Modeling Conference, 5-6 May 1992, Herndon, Virginia*, 1993.

[9] 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.

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

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