A Rewriting Logic Sampler

José Meseguer

University of Illinois at Urbana-Champaign, USA

Abstract. Rewriting logic is a simple computational logic very well suited as a *semantic framework* within which many different models of computation, systems and languages can be naturally modeled. It is also a flexible *logical framework* in which many different logical formalisms can be both represented and executed. As the title suggests, this paper does not try to give a comprehensive overview of rewriting logic. Instead, after introducing the basic concepts, it focuses on some recent research directions emphasizing: (i) extensions of the logic to model real-time systems and probabilistic systems; and (ii) some exciting application areas such as: semantics of programming languages, security, and bioinformatics.

1 Introduction

Rewriting logic is now a teenager; a *quinceañera*, as they call adolescent women reaching 15 in Spain and Latin America. There are hundreds of papers; five rewriting logic workshops have already taken place and a sixth will meet in Vienna next March; and a host of tools and applications have been developed. Taking pictures of this "young person" as it grows up is a quite interesting intellectual exercise, one that can help other people become familiar with this field and its possibilities. I, with the help of others, have done my share of picture taking in earlier stages [69,70,72,67]. In particular, the "roadmap" [67] that Narciso Matí-Oliet and I wrote, gives a brief but comprehensive overview and cites 328 papers in the area as of 2002. This paper takes a different tack. I will *not* try to give you an overview. I will give you a *sampler*, some rewriting logic *tapas* if you will, to tease your curiosity so that hopefully you may find some things that you like and excite your interest.

I should of course say something about my choice of topics for the sampler; and about some important developments that I do not cover. At the theoretical level, one of the interesting questions to ask about a formalism is: how *general, flexible and extensible* is it? For example, how does it compare in generality to other formalisms? how can it deal with new application areas? how well can it be extended in new directions? can it represent its own metalevel? I address some of these questions by my choice of topics, but I consciously omit others. The most glaring omission is the theoretical extension from ordinary rewrite theories to *generalized rewrite theories* [10], that substantially extend the logic's expressive power. For the sake of a simpler exposition, this whole development

D.V. Hung and M. Wirsing (Eds.): ICTAC 2005, LNCS 3722, pp. 1–28, 2005.

is relegated here to Footnote 1. I do however discuss two other important theoretical extensions, namely, *real-time rewrite theories* [87] (Section 3.2), which extend rewriting logic to deal with real-time and hybrid systems; and *probabilistic rewrite theories* [63,64,2] (Section 3.3), that bring probabilistic systems, as well as systems exhibiting both probabilistic and nodeterministic behavior, within the rewriting logic fold. In both cases, the generality aspect is quite encouraging, in the sense that many models of real time and of probabilistic systems appear as special cases. However, to keep the exposition short, I do not discuss all those models except in passing, and refer to [87] and [63] for detailed comparisons. For the generality of rewriting logic itself see [67]. Reflection, that allows rewriting logic to represent its own metalevel, is of such great theoretical and practical importance that I also discuss it in Section 2.3.

At the practical level, one can ask questions such as: how well is this formalism supported by *tools*? (this I briefly answer in Section 2.4); and what are some exciting application areas? I have chosen three such areas for the sampler: (1) semantics of programming languages and formal analysis of programs (Section 3.1); (2) security (Section 3.4); and (3) bioinformatics (Section 3.5). Enjoy!

2 What Is Rewriting Logic?

A *rewrite theory*[1] is a tuple $\mathcal{R} = (\Sigma, E, R)$, with:

- (Σ, E) an equational theory with function symbols Σ and equations E; and
- R a set of *labeled rewrite rules* of the general form

$$r : t \longrightarrow t'$$

with t, t' Σ-terms which may contain variables in a countable set X of variables which we assume fixed in what follows; that is, t and t' are elements of the term algebra $T_\Sigma(X)$. In particular, their corresponding sets of variables, $vars(t), vars(t')$ are both contained in X.

[1] To simplify the exposition I present here the simplest version of rewrite theories, namely, *unconditional* rewrite theories over an *unsorted* equational theory (Σ, E). In general, however, the equational theory (Σ, E) can be many-sorted, order-sorted, or even a membership equational theory [71]. And the rules can be *conditional*, having a conjunction of rewrites, equalities, and even memberships in their condition, that is, they could have the general form

$$r : t \longrightarrow t' \text{ if } (\bigwedge_i u_i = u_i') \wedge (\bigwedge_j v_j : s_j) \wedge (\bigwedge_l w_l \longrightarrow w_l')$$

Furthermore, the theory may also specify an additional mapping $\phi : \Sigma \longrightarrow \mathcal{P}(\mathbb{N})$, assigning to each function symbol $f \in \Sigma$ (with, say, n arguments) a set $\phi(f) = \{i_1, \ldots, i_k\}$, $1 \leq i_1 < \ldots < i_k \leq n$ of *frozen argument positions* under which it is forbidden to perform any rewrites. Rewrite theories in this more general sense are studied in detail in [10]; they are clearly more expressive than the simpler unconditional and unsorted version presented here. This more general notion is the one supported by the Maude language [17,18].

Intuitively, $\mathcal{R}$ specifies a *concurrent system*, whose states are elements of the initial algebra $T_{\Sigma/E}$ specified by (Σ, E), and whose *concurrent transitions* are specified by the rules R. The equations E may decompose as a union $E = E_0 \cup A$, where A is a (possibly empty) set of structural axioms (such as associativity, commutativity, and identity axioms). To give a flavor for how concurrent systems are axiomatized in rewriting logic, I discuss below a fault-tolerant communication protocol example specified as a Maude [17,18] module[2]

```
mod FT-CHANNEL is
protecting NAT .
sorts NatList Msg MsgSet Channel .
subsorts Nat < NatList .
subsorts Msg < MsgSet .
op nil : -> NatList .
op _;_ : NatList NatList -> NatList [assoc id: nil] .
op null : -> MsgSet .
op __ : MsgSet MsgSet -> MsgSet [assoc comm id: null] .
op [_,_]_[_,_] : NatList Nat MsgSet NatList Nat -> Channel .
op {_,_} : Nat Nat -> Msg .
op ack : Nat -> Msg .

vars N M I J K : Nat .
vars L P Q R : NatList .
var MSG : Msg .
var S : MsgSet .

rl [send] : [J ; L,N] S [P,M] => [J ; L,N] {J,N} S [P,M] .
rl [recv] : [J ; L,N] {J,K} S [P,M] =>
              if K == M then [J ; L,N] S ack(M) [P ; J,s(M)]
                        else [J ; L,N] S ack(K) [P,M] fi .
rl [ack-recv] : [J ; L,N] ack(K) S [P,M] =>
                  if K == N then [L,s(N)] S [P,M]
                            else [J ; L,N] S [P,M] fi .
rl [loss] : [L,N] MSG S [P,M] => [L,N] S [P,M] .
endm
```

This rewrite theory imports the natural numbers module `NAT` and has an order-sorted signature Σ specified by its sorts, subsorts, and operations. All its equations are *structural axioms* A, which in Maude are not specified explicitly as equations, but are instead declared as attributes of their corresponding operator: here the list concatenation operator `_;_` has been declared associative and

[2] The Maude syntax is so close to the corresponding mathematical notation for defining rewrite theories as to be almost self-explanatory. The general point to keep in mind is that each item: a sort, a subsort, an operation, an equation, a rule, etc., is declared with an obvious keyword: `sort`, `subsort`, `op`, `eq` (or `ceq` for conditional equations), `rl` (or `crl` for conditional rules), etc., with each declaration ended by a space and a period. Another important point is the use of "mix-fix" user-definable syntax, with the argument positions specified by underbars; for example: `if_then_else_fi`.

having `nil` as its identity element with the `assoc` and `id:` keywords. Similarly, the multiset union operator has been declared with empty syntax (juxtaposition) `__` and with associativity, commutativity (`comm`), and identity axioms, making `null` its identity element. The rules R are `send`, `recv`, `ack-recv`, and `loss`; they are applied *modulo* the structural axioms A, that is, we get the effect of rewriting in A-equivalence classes. This theory specifies a fault-tolerant communication protocol in a bidirectional faulty channel, where messages can be received out of order and can be lost. The sender is placed at the left of the channel and has a list of numbers to send and a counter. The receiver is placed at the right, with also a list of numbers to receive and another counter. The contents of the channel in the middle is a multiset of messages (since there can be several repeated copies of the same message). The protocol is *fault-tolerant*, in that it will work even when some messages are permuted or lost, provided the `recv` and `ack-recv` rules are applied in a *fair* way (for fairness in rewriting logic see [74]).

2.1 Rewriting Logic Deduction

Given $\mathcal{R} = (\Sigma, E, R)$, the sentences that $\mathcal{R}$ proves are rewrites of the form, $t \longrightarrow t'$, with $t, t' \in T_\Sigma(X)$, which are obtained by finite application of the following *rules of deduction*:

- **Reflexivity.** For each $t \in T_\Sigma(X)$, $\dfrac{}{t \longrightarrow t}$
- **Equality.** $\dfrac{u \longrightarrow v \quad E \vdash u = u' \quad E \vdash v = v'}{u' \longrightarrow v'}$
- **Congruence.** For each $f : k_1 \ldots k_n \longrightarrow k$ in Σ, and $t_i, t_i' \in T_\Sigma(X)$, $1 \leq i \leq n$,

$$\frac{t_1 \longrightarrow t_1' \quad \ldots \quad t_n \longrightarrow t_n'}{f(t_1, \ldots, t_n) \longrightarrow f(t_1', \ldots, t_n')}$$

- **Replacement.** For each substitution $\theta : X \longrightarrow T_\Sigma(X)$, and for each rule $r : t \longrightarrow t'$ in R, with, say, $vars(t) \cup vars(t') = \{x_1, \ldots, x_n\}$, and $\theta(x_l) = p_l$, $1 \leq l \leq n$, then

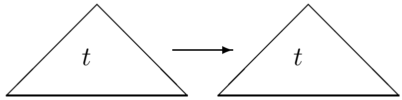
$$\frac{p_1 \longrightarrow p_1' \quad \ldots \quad p_n \longrightarrow p_n'}{\theta(t) \longrightarrow \theta'(t')}$$

  where for $1 \leq i \leq n$, $\theta'(x_i) = p_i'$, and for each $x \in X - \{x_1, \ldots, x_n\}$, $\theta'(x) = \theta(x)$.
- **Transitivity.**

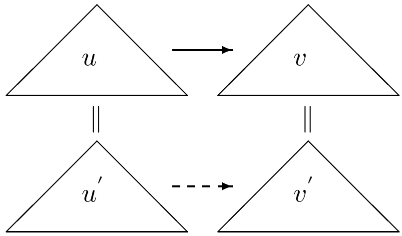
$$\frac{t_1 \longrightarrow t_2 \qquad t_2 \longrightarrow t_3}{t_1 \longrightarrow t_3}$$

We can visualize the above inference rules as follows:

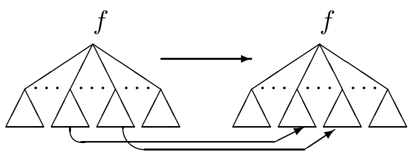
Reflexivity



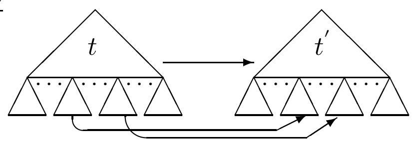
Equality



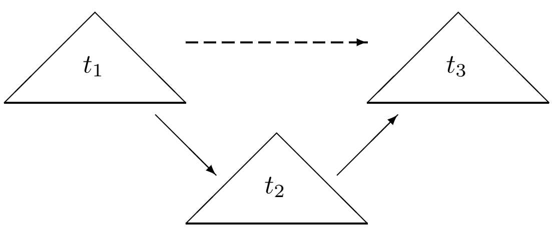
Congruence



Replacement



Transitivity



The notation $\mathcal{R} \vdash t \longrightarrow t'$ states that the sequent $t \longrightarrow t'$ is *provable* in the theory $\mathcal{R}$ using the above inference rules. Intuitively, we should think of the inference rules as different ways of *constructing* all the (finitary) *concurrent computations* of the concurrent system specified by $\mathcal{R}$. The **Reflexivity** rule says that for any state t there is an *idle transition* in which nothing changes. The **Equality** rule specifies that the states are in fact equivalence classes modulo the equations E. The **Congruence** rule is a very general form of "sideways parallelism," so that each operator f can be seen as a *parallel state constructor*, allowing its arguments to evolve in parallel. The **Replacement** rule supports a different form of parallelism, which could be called "parallelism under one's feet," since besides rewriting an instance of a rule's lefthand side to the corresponding righthand side instance, the state fragments in the substitution of the rule's variables can also be rewritten. Finally, the **Transitivity** rule allows us to build longer concurrent computations by composing them sequentially.

For execution purposes, a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ should satisfy some additional requirements. As already mentioned, the equations E may decompose as a union $E = E_0 \cup A$, where A is a (possibly empty) set of structural axioms. We should require that matching modulo A is decidable, and that the equations E_0 are ground Church-Rosser and terminating modulo A; furthermore, the rules $r : t \longrightarrow t'$ in R should satisfy $vars(t') \subseteq vars(t)$, and should be coherent with respect to E modulo A [109]. In the Maude language [17,18], modules are rewrite theories that are assumed to satisfy the above executability requirements (in an extended form that covers conditional rules [17]).

2.2 Operational and Denotational Semantics of Rewrite Theories

A rewrite theory $\mathcal{R} = (\Sigma, E, R)$ has both a *deduction-based operational semantics*, and an *initial model denotational semantics*. Both semantics are defined naturally out of the proof theory described in Section 2.1. The deduction-based operational semantics of $\mathcal{R}$ is defined as the collection of *proof terms* [69,10] of the form $\alpha : t \longrightarrow t'$. A proof term α is an algebraic description of a proof tree proving $\mathcal{R} \vdash t \longrightarrow t'$ by means of the inference rules of Section 2.1. As already mentioned, all such proof trees describe all the possible *finitary concurrent computations* of the concurrent system axiomatized by $\mathcal{R}$. When we specify $\mathcal{R}$ as a Maude module and rewrite a term t with the `rewrite` or `frewrite` commands, obtaining a term t' as a result, we can use Maude's `trace` mode to obtain what amounts to a proof term $\alpha : t \longrightarrow t'$ of the particular rewrite proof built by the Maude interpreter.

A rewrite theory $\mathcal{R} = (\Sigma, E, R)$ has also a model theory, so that the inference rules of rewriting logic are sound and complete with respect to satisfaction in the class of models of $\mathcal{R}$ [69,10]. Such models are *categories* with a (Σ, E)-algebra structure [69,10]. These are "true concurrency" denotational models of the concurrent system axiomatized by $\mathcal{R}$. That is, this model theory gives a precise mathematical answer to the question: when do two descriptions of two concurrent computations denote *the same* concurrent computation? The class of models of a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ has an *initial model* $\mathcal{T}_{\mathcal{R}}$ [69,10]. The initial model semantics is obtained as a *quotient* of the just-mentioned deduction-based operational semantics, precisely by axiomatizing algebraically when two proof terms $\alpha : t \longrightarrow t'$ and $\beta : u \longrightarrow u'$ denote the same concurrent computation. Of course, α and β should have identical beginning states and identical ending states. By the **Equality** rule this forces $E \vdash t = u$, and $E \vdash t' = u'$. That, is, the objects of the category $\mathcal{T}_{\mathcal{R}}$ are E-equivalence classes $[t]$ of ground Σ-terms, which denote the states of our system. The arrows or morphisms in $\mathcal{T}_{\mathcal{R}}$ are *equivalence classes of proof terms*, so that $[\alpha] = [\beta]$ iff both proof terms denote the same concurrent computation according to the "true concurrency" axioms. Such axioms are very natural. They for example express that the **Transitivity** rule behaves as an arrow composition and is therefore associative. Similarly, the **Reflexivity** rules provides an identity arrow for each object, satisfying the usual identity laws.

As discussed in Section 4.1 of [67], rewriting logic is a very general *semantic framework* in which a wide range of concurrency models such as process calculi, Petri nets, distributed object systems, Actors, and so on, can be naturally axiomatized as specific rewrite theories. Furthermore, as also explained in Section 4.1 of [67], the algebraically-defined true concurrency models of rewriting logic include as special cases many other true concurrency models such as residual models of term rewriting, parallel λ-calculus models, process models for Petri nets, proved transition models for CCS, and partial order of events models for object systems and for Actors. Note, however, that a rewrite rule

$$r : t \longrightarrow t'$$

has *two complementary readings, one computational, and another logical.* Computationally, as already explained, it axiomatizes a parametric family of *concurrent transitions* in a system. Logically, however, it represents and *inference rule*[3] in a *logic*, whose inference system is axiomatized by $\mathcal{R}$. It turns out that, with this second reading, rewriting logic has very good properties as a *logical framework*, in which many other logics can be naturally represented, so that we can simulate deduction in a logic as rewriting deduction in its representation [66].

2.3 Reflection

Reflection is a very important property of rewriting logic [22,15,23,24]. Intuitively, a logic is reflective if it can represent its metalevel at the object level in a sound and coherent way. Specifically, rewriting logic can represent its own theories and their deductions by having a finitely presented rewrite theory $\mathcal{U}$ that is *universal*, in the sense that for any finitely presented rewrite theory $\mathcal{R}$ (including $\mathcal{U}$ itself) we have the following equivalence

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

---

[3] The use of conditional rewrite rules is of course very important in this logical reading. Logically, we would denote a conditional rewrite rule

$$r : t \longrightarrow t' \;\text{ if }\; (\bigwedge_i u_i = u_i') \wedge (\bigwedge_j v_j : s_j) \wedge (\bigwedge_l w_l \longrightarrow w_l')$$

as an inference rule

$$\frac{(\bigwedge_i u_i = u_i') \wedge (\bigwedge_j v_j : s_j) \wedge (\bigwedge_l w_l \longrightarrow w_l')}{t \longrightarrow t'}$$

where $\overline{\mathcal{R}}$ and $\overline{t}$ are terms representing $\mathcal{R}$ and t as data elements of $\mathcal{U}$. Since $\mathcal{U}$ is representable in itself, we can achieve a "reflective tower" with an arbitrary number of levels of reflection [15,16].

Reflection is a very powerful property: it allows defining rewriting strategies by means of metalevel theories that extend $\mathcal{U}$ and guide the application of the rules in a given object-level theory $\mathcal{R}$ [15]; it is efficiently supported in the Maude implementation by means of *descent functions* [16]; it can be used to build a variety of theorem proving and theory transformation tools [15,19,20,25]; it can endow a rewriting logic language like Maude with powerful theory composition operations [40,35,37,42]; and it can be used to prove metalogical properties about families of theories in rewriting logic, and about other logics represented in the rewriting logic (meta-)logical framework [5,21,4].

2.4 Maude and Its Formal Tools

Rewrite theories can be executed in different languages such as CafeOBJ [53], and ELAN [7]. The most general support for the execution of rewrite theories is currently provided by the Maude language [17,18], in which rewrite theories with very general conditional rules, and whose underlying equational theories can be membership equational theories [71], can be specified and can be executed, provided they satisfy the already-mentioned requirements. Furthermore, Maude provides very efficient support for rewriting *modulo* any combination of associativity, commutativity, and identity axioms. Since an equational theory (Σ, E) can be regarded as a degenerate rewrite theory of the form $(\Sigma, E, \varnothing)$, equational logic is naturally a sublogic of rewriting logic. In Maude this sublogic is supported by *functional modules* [17], which are theories in membership equational logic.

Besides supporting efficient execution, typically in the order of several million rewrites per second, Maude also provides a range of formal tools and algorithms to analyze rewrite theories and verify their properties. A first very useful formal analysis feature is its *breadth-first search* command. Given an initial state of a system (a term), we can search for all reachable states matching a certain pattern and satisfying an equationally-defined semantic condition P. By making $P = \neg Q$, where Q is an invariant, we get in this way a *semi-decision procedure* for finding failures of invariant safety properties. Note that there is no finite-state assumption involved here: any executable rewrite theory can thus be analyzed. For systems where the set of states reachable from an initial state are finite, Maude also provides a linear time temporal logic (LTL) model checker. Maude's is an explicit-state LTL model checker, with performance comparable to that of the SPIN model checker [58] for the benchmarks that we have analyzed [45,46].

As already pointed out, *reflection* is a key feature of rewriting logic, and is efficiently supported in the Maude implementation through its `META-LEVEL` module. One important fruit of this is that it becomes quite easy to build new formal tools and to add them to the Maude environment. Indeed, such tools by their very nature manipulate and analyze rewrite theories. By reflection, a rewrite theory $\mathcal{R}$ becomes a *term* $\overline{\mathcal{R}}$ in the universal theory, which can be efficiently manipulated by the descent functions in the `META-LEVEL` module. As a consequence, Maude formal tools have a reflective design and are built in Maude as suitable extensions of the `META-LEVEL` module. They include the following:

– the Maude Church-Rosser Checker, and Knuth-Bendix and Coherence Completion tools [19,41,38,36]
– the Full Maude module composition tool [35,42]
– the Maude Predicate Abstraction tool [88]
– the Maude Inductive Theorem Prover (ITP) [15,19,25]
– the Real-Time Maude tool [82] (more on this in Section 3.2)
– the Maude Sufficient Completeness Checker (SCC) [57]
– the Maude Termination Tool (MTT) [39].

3 Some Research Directions

3.1 The Rewriting Logic Semantics Project

The fact that rewriting logic specifications provide an easy and expressive way to develop executable formal definitions of languages, which can then be subjected to different tool-supported formal analyses, is by now well established [107,8,108,103,98,73,105,14,91,106,51,49,59,9,75,76,13,12,50,26,93,3,99,27,77]. In fact, the just-mentioned papers by different authors are contributions to a collective ongoing research project which we call the *rewriting logic semantics project.* What makes this project promising is the combination of three interlocking facts:

1. that rewriting logic is a flexible and expressive *logical framework* that unifies denotational semantics[4] and SOS in a novel way, avoiding their respective limitations and allowing very succinct semantic definitions (see [77]);
2. that rewriting logic semantic definitions are *directly executable* in a rewriting logic language such as Maude [17], and can thus become quite efficient interpreters (see [76,77]) ; and
3. that *generic formal tools* such as the Maude LTL model checker [45], the Maude inductive theorem prover [19,25], and new tools under development such as a language-generic partial order reduction tool [50], allow us to amortize tool development cost across many programming languages, that can thus be endowed with powerful program analysis capabilities; furthermore, *genericity does not necessarily imply inefficiency*: in some cases the analyses so obtained outperform those of well-known language-specific tools [51,49].

For the most part, equational semantics and SOS have lived separate lives. Although each is very valuable in its own way, they are "single hammer" approaches and have some limitations [77]. Would it be possible to seamlessly

[4] I use in what follows the broader term *equational semantics* —that is, semantics based on *semantic equations*— to emphasize the fact that higher-order denotational and first-order algebraic semantics have many common features and can both be viewed as instances of a common equational semantics framework.

unify them within a more flexible and general framework? Could their respective limitations be overcome when they are thus unified? Rewriting logic does indeed provide one such unifying framework. The key to this, indeed very simple, unification is what Grigore Roşu and I call rewriting logic's *abstraction knob*. The point is that in equational semantics' model-theoretic approach entities are *identified by the semantic equations*, and have unique *abstract denotations* in the corresponding models. In our knob metaphor this means that in equational semantics the abstraction knob is *always turned all the way up to its maximum position*. By contrast, one of the key features of SOS is providing a very detailed, step-by-step formal description of a language's evaluation mechanisms. As a consequence, most entities —except perhaps for built-in data, stores, and environments, which are typically treated on the side— are *primarily syntactic*, and computations are described in full detail. In our metaphor this means that in SOS the abstraction knob is *always turned down to its minimum position*.

How is the unification and corresponding availability of an abstraction knob achieved? Since a rewrite theory (Σ, E, R) has an underlying equational theory (Σ, E) with Σ a signature of operations and sorts, and E a set of (possibly conditional) equations, and with R a set of (possibly conditional) rewrite rules, equational semantics is then obtained as the special case in which $R = \varnothing$, so we only have the semantic equations E and the abstraction knob is turned up to its maximum position. Roughly speaking,[5] SOS is then obtained as the special case in which $E = \varnothing$, and we only have (possibly conditional) rules R rewriting purely syntactic entities (terms), so that the abstraction knob is turned down to the minimum position.

Rewriting logic's "abstraction knob" is precisely its crucial distinction between equations E and rules R in a rewrite theory (Σ, E, R). *States of the computation* are then E-equivalence classes, that is, *abstract elements* in the initial algebra $T_{\Sigma/E}$. Because of rewriting logic's **Equality** inference rule (see Section 2.1) a rewrite with a rule in R is understood as a transition $[t] \longrightarrow [t']$ between such abstract states. The knob, however, can be turned up or down. We can turn it *all the way down to its minimum* by converting all equations into rules, transforming (Σ, E, R) into $(\Sigma, \varnothing, R \cup E)$. This gives us the most concrete, SOS-like semantic description possible. Instead, to make a specification *as abstract as possible* we can identify a subset $R_0 \subseteq R$ such that: (1) $R_0 \cup E$ is Church-Rosser; and (2) R_0 is biggest possible with this property. In actual language specification practice this is not hard to do. Essentially, we can use semantic equations for most of the sequential features of a programming language: only when interactions with memory could lead to nondeterminism (particularly if the language has threads, or they could later be added to the language in an extension) or for intrinsically concurrent features are rules (as opposed to

[5] I gloss over the technical difference that in SOS all computations are "one-step" computations, even if the step is a big one, whereas in rewriting logic, because of its built-in **Transitivity** inference rule (see Section 2.1) the rewriting relation is always transitive. For a more detailed comparison see [76].

equations) really needed. In this way, we can obtain drastic search space reductions, making formal analyses much more scalable than if we used only rules.

Many languages have already been given semantics in this way using Maude. The language definitions can then be used as interpreters, and —in conjunction with Maude's search command and its LTL model checker— to formally analyze programs in those languages. For example, large fragments of Java and the JVM have been specified in Maude this way, with the Maude rewriting logic semantics being used as the basis of Java and JVM program analysis tools that for some examples outperform well-known Java analysis tools [51,49]. A similar Maude specification of the semantics of Scheme at UIUC yields an interpreter with .75 the speed of the standard Scheme interpreter on average for the benchmarks tested. The specification of a C-like language and the corresponding formal analyses are discussed in detail in [77]. A semantics of a Caml-like language with threads was discussed in detail in [76], and a modular rewriting logic semantics of CML has been given by Chalub and Braga in [13]. d'Amorim and Roşu have given a definition of the Scheme language in [27]. Other language case studies, all specified in Maude, include: bc [9], CCS [107,108,9], CIAO [99], Creol [59], ELOTOS [105], MSR [11,97], PLAN [98,99], and the pi-calculus [103]. In fact, the semantics of large fragments of conventional languages are by now routinely developed by UIUC graduate students as course projects in a few weeks, including, besides the languages already mentioned: Beta, Haskell, Lisp, LLVM, Pict, Python, Ruby, and Smalltalk.

Besides search and model checking analyses, it is also possible to use a language's semantic definition to perform semantics-based deduction analyses either on programs in that language, or even about the correctness of a given logic of programs with respect to the language's rewriting semantics. Work in this direction includes [93,3,26,108,105].

Modularity of semantic definitions, that is, the property that a feature's semantics does not have to be redefined when a language is extended, is notoriously hard to achieve. To solve this problem for SOS, Peter Mosses has proposed the *modular structural operational semantics* (MSOS) methodology [80]. This inspired C. Braga and me to develop a similar modular methodology for rewriting logic semantics [75,9]. This has had the pleasant side-effect of providing a Maude-based execution environment for MSOS specifications, namely the Maude MSOS Tool developed at the Universidade Federal Fluminense in Brazil by F. Chalub and C. Braga [12], which is available on the web at `http://mmt.ic.uff.br/`.

3.2 Real-Time Rewrite Theories and Real-Time Maude

In many reactive and distributed systems, real-time properties are essential to their design and correctness. Therefore, the question of how systems with real-time features can be best specified, analyzed, and proved correct in the semantic framework of rewriting logic is an important one. This question has been investigated by several authors from two perspectives. On the one hand, an extension of rewriting logic called *timed rewriting logic* has been investigated, and has been applied to some examples and specification languages [62,84,96]. On the other hand, Peter Ölvecky and I have found a simple way to express real-time and hybrid system specifications *directly* in rewriting logic [85,87]. Such specifications are called *real-time rewrite theories* and have rules of the form

$$\{t\} \xrightarrow{r} \{t'\} \;\; \mathit{if} \;\; C$$

with r a term denoting the *duration* of the transition (where the time can be chosen to be either discrete or continuous), $\{t\}$ representing the *whole* state of a system, and C an equational condition. Peter Ölvecky and I have shown that, by making the clock an explicit part of the state, these theories can be *desugared* into semantically equivalent ordinary rewrite theories [85,87,82]. That is, in the desugared version we can model the state of a real-time or hybrid system as a pair (t, r), with t the current state, and with r the current global clock time. Rewrite rules can then be either *instantaneous rules*, that take no time and only change some part of the state t, or *tick rules*, that advance the global time of the system according to some time expression r and may also change the state t. By characterizing equationally the enabledness of each rule and using conditional rules and *frozen* operators [10], it is always possible to define tick rules so that instantaneous rules are always given higher priority; that is, so that a tick rule can never fire when an instantaneous rule is enabled [82]. When time is continuous, tick rules may be *nondeterministic*, in the sense that the time r advanced by the rule is not uniquely determined, but is instead a parametric expression (however, this time parameter is typically subjected to some equational condition C). In such cases, tick rules need a *time sampling strategy* to choose suitable values for time advance. Besides being able to show that a wide range of known real-time models, (including, for example, timed automata, hybrid automata, timed Petri nets, and timed object-oriented systems) and of discrete or dense time values, can be naturally expressed in a direct way in rewriting logic (see [87]), an important advantage of our approach is that one can use an existing implementation of rewriting logic to execute and analyze real-time specifications. Because of some technical subtleties, this seems difficult for the alternative of timed rewriting logic, although a mapping into our framework does exist [87].

Real-Time Maude [83,86,82], is a specification language and a formal tool built in Maude by reflection. It provides special syntax to specify real-time systems, and offers a range of formal analysis capabilities. The Real-Time Maude 2.0 tool [82] systematically exploits the underlying Maude efficient rewriting, search, and LTL model checking capabilities to both execute and formally analyze real-time specifications. Reflection is crucially exploited in the Real-Time Maude 2.0 implementation. On the one hand Real-Time Maude specifications are internally desugared into ordinary Maude specifications by transforming their meta-representations. On the other, reflection is also used for execution and analysis purposes. The point is that the desired modes of execution and formal properties to be analyzed have real-time aspects with no clear counterpart at the Maude level. To faithfully support these real-time aspects a *reflective transformational approach* is adopted: the original real-time theory and query (for either execution or analysis) are simultaneously transformed into a semantically equivalent pair of a Maude rewrite theory and a Maude query [82]. In practice, this makes those executions and analyses quite efficient and allows scaling up to highly nontrivial specifications and case studies.

In fact, both the naturalness of Real-Time Maude to specify large nontrivial real-time applications (particularly for distributed object-oriented real-time systems) and its effectiveness in simulating and analyzing the formal properties of such systems have been demonstrated in a number of substantial case studies, including the specification and analysis of advanced scheduling algorithms and of: (1) the AER/NCA suite of active network protocols [83,81]; (2) the NORM multicast protocol [65]; and (3) the OGDC wireless sensor network algorithm [104]. The Real-Time Maude tool is a mature and quite efficient tool freely available (with source code, a tool manual, examples, case studies, and papers) from `http://www.ifi.uio.no/RealTimeMaude`.

3.3 Probabilistic Rewrite Theories and PMaude

Many systems are probabilistic in nature. This can be due either to the uncertainty of the environment in which they must operate, such as message losses and other failures in an unreliable environment, or to the probabilistic nature of some of their algorithms, or to both. In general, particularly for distributed systems, both probabilistic and nondeterministic aspects may coexist, in the sense that different transitions may take place nondeterministically, but the outcomes of some of those transitions may be probabilistic in nature. To specify systems of this kind, rewrite theories have been generalized to *probabilistic rewrite theories* in [63,64,2]. Rules in such theories are *probabilistic rewrite rules* of the form

$$l : t(\boldsymbol{x}) \to t'(\boldsymbol{x}, \boldsymbol{y}) \ \ \textit{if} \ \ \textit{cond}(\boldsymbol{x}) \ \ \textit{with} \ \ \textit{probability} \ \ \boldsymbol{y} := \pi_r(\boldsymbol{x})$$

where the first thing to observe is that the term t' has new variables $\boldsymbol{y}$ disjoint from the variables $\boldsymbol{x}$ appearing in t. Therefore, such a rule is *nondeterministic*; that is, the fact that we have a matching substitution θ such that $\theta(\textit{cond})$ holds does not uniquely determine the next state fragment: there can be many different choices for the next state, depending on how we instantiate the extra variables $\boldsymbol{y}$ in t'. In fact, we can denote the different such next states by expressions of the form $t'(\theta(\boldsymbol{x}), \rho(\boldsymbol{y}))$, where θ is fixed as the given matching substitution, but ρ ranges along all the possible substitutions for the new variables $\boldsymbol{y}$. The probabilistic nature of the rule is expressed by the notation: *with probability* $\boldsymbol{y} := \pi_r(\boldsymbol{x})$, where $\pi_r(\boldsymbol{x})$ is a probability distribution *which may depend on the matching substitution* θ. We then choose the values for $\boldsymbol{y}$, that is, the substitution ρ, probabilistically according to the distribution $\pi_r(\theta(\boldsymbol{x}))$.

The fact that the probability distribution may depend on the substitution θ can be illustrated by means of a simple example. Consider a battery-operated clock. We may represent the state of the clock as a term `clock(T,C)`, with `T` a natural number denoting the time, and `C` a positive real denoting the amount of battery charge. Each time the clock ticks, the time is increased by one unit, and the battery charge slightly decreases; however, the lower the battery charge, the greater the chance that the clock will stop, going into a state of the form broken(T,C’). We can model this system by means of the probabilistic rewrite rule

```
rl [tick]: clock(T,C) => if B then clock(s(T),C - (C / 1000))
                              else broken(T,C (C / 1000))
                         fi
                 with probability B := BERNOULLI(C / 1000) .
```

that is, the probability of the clock breaking down instead of ticking normally *depends on the battery charge*, which is here represented by the battery-dependent bias of the coin in a Bernoulli trial. Note that here the new variable on the rule’s righthand side is the Boolean variable B, corresponding to the result of tossing the biased coin. As shown in [63], probabilistic rewrite theories can express a wide range of models of probabilistic systems, including continuous-time Markov chains [100], probabilistic non-deterministic systems [90,94], and generalized semi-Markov processes [54]; they can also naturally express probabilistic object-based distributed systems [64,2], including real-time ones.

The PMaude language [64,2] is an experimental specification language whose modules are probabilistic rewrite theories. Note that, due to their nondeterminism, probabilistic rewrite rules *are not directly executable*. However, probabilistic systems specified in PMaude *can be simulated in Maude*. This is accomplished by transforming a PMaude specification into a corresponding Maude specification in which actual values for the new variables appearing in the righthand side of a probabilistic rewrite rule are obtained by *sampling* the corresponding probability distribution functions. This theory transformation uses three key Maude modules as basic infrastructure, namely, COUNTER, RANDOM, and SAMPLER. The built-in module COUNTER provides a built-in strategy for the application of the nondeterministic rewrite rule

```
rl counter => N:Nat .
```

that rewrites the constant counter to a natural number. The built-in strategy applies this rule so that the natural number obtained after applying the rule is exactly the successor of the value obtained in the preceding rule application. The RANDOM module is a built-in Maude module providing a (pseudo-)random number generator function called random. The SAMPLER module supports sampling for different probability distributions. It has a rule

```
rl [rnd] : rand => float(random(counter + 1) / 4294967296) .
```

which rewrites the constant rand to a floating point number between 0 and 1 pseudo-randomly chosen according to the uniform distribution. This floating point number is obtained by converting the rational number random(counter + 1) / 4294967296 into a floating point number, where 4294967296 is the maximum value that the random function can attain. SAMPLER has rewrite rules supporting sampling according to different probability distributions; this is based on first sampling a floating point number between 0 and 1 pseudo-randomly chosen according to the uniform distribution by means of the above rnd rule.

For example, to sample the Bernoulli distribution we use the following operator and rewrite rule in `SAMPLER`:

```
op BERNOULLI : Float -> Bool .
rl BERNOULLI(R) => if rand < R then true else false fi .
```

that is, to sample a result of tossing a coin with bias `R`, we first sample the uniform distribution. If the sampled value is strictly smaller than `R`, then the answer is `true`; otherwise the answer is `false`. Any discrete probability distribution on a finite set can be sampled in a similar way. The ordinary Maude specification that *simulates* the PMaude specification for a clock with the above `tick` probabilistic rewrite rule imports `COUNTER`, `RANDOM`, and `SAMPLER`, and has then a corresponding Maude rewrite rule

```
rl [tick] : clock(T,C) => if BERNOULLI(C / 1000.0)
                             then clock(s(T),C - (C / 1000.0))
                             else broken(T,C - (C / 1000.0))
                          fi .
```

For a continuous probability distribution π with differentiable density function d_π, and with cumulative distribution function $F_\pi(x) = \int_{-\infty}^{x} d_\pi(y)dy$, we can use the well-known fact (see for example [89], Thm 8A, pg. 314) that if U is a random variable uniformly distributed on $[0,1]$, then $F_\pi^{-1}(U)$ is a random variable with probability distribution π, to sample elements according to the distribution π by means of a rewrite rule

$$sample_\pi \longrightarrow F_\pi^{-1}(\texttt{random})$$

Of course, π may not be a fixed probability distribution, but a *parametric family* $\pi(\boldsymbol{p})$ of distributions depending on some parameters $\boldsymbol{p}$, so that the above rule will then have extra variables for those parameters.

In general, provided that sampling for the probability distributions used in a PMaude module are supported in the underlying `SAMPLER` module, we can associate to it a corresponding Maude module. We can then use this associated Maude module to perform Monte Carlo simulations of the probabilistic systems thus specified. As explained in [2], provided all nondeterminism has been eliminated from the original PMaude module[6], we can then use the results of such Monte Carlo simulations to perform a *statistical model checking analysis* of the

[6] The point is that, as explained above, in general, given a probabilistic rewrite theory and a term t describing a given state, there can be several different rewrites, perhaps with different rules, at different positions, and with different matching substitutions, that can be applied to t. Therefore, the choice of rule, position, and substitution is *nondeterministic*. To eliminate all nondeterminism, at most one rule at exactly one position and with a unique substitution should be applicable to any term t. As explained in [2], for many systems, including probabilistic real-time object-oriented systems, this can be naturally achieved, essentially by scheduling events at real-valued times that are all different, because we sample a continuous probability distribution on the real numbers.

given system to verify certain properties. For example, for a PMaude specification of a TCP/IP protocol variant that is resistant to Denial of Service (DoS) attacks, we may wish to establish that, even if an attacker controls 90% of the network bandwith, it is still possible for the protocol to establish a connection in less than 30 seconds with 99% probability. Properties of this kind, including properties that measure quantitative aspects of a system, can be expressed in the QATEX probabilistic temporal logic, [2], and can be model checked using the VeStA tool [95]. See [1] for a substantial case study specifying a DoS-resistant TCP/IP protocol as a PMaude module, performing Monte Carlo simulations by means of its associated Maude module, and formally analyzing in VeStA its properties, expressed as QATEX specifications, according to the methodology just described.

3.4 Security Applications and Narrowing

Security is a concern of great practical importance for many systems, making it worthwhile to subject system designs and implementations to rigorous formal analysis. Security, however, is *many-faceted*: on the one hand, we are concerned with properties such as *secrecy*: malicious attackers should not be able to get secret information; on the other, we are also concerned with properties such as *availability*, which may be destroyed by a (DoS) attack: a highly reliable communication protocol ensuring secrecy may be rendered useless because it spends all its time checking spurious signatures generated by a DoS attacker. Rewriting logic has been successfully applied to analyze security properties, including both secrecy and availability, for a wide range of systems. More generally, using distributed object-oriented reflection techniques [28,78], it is possible to analyze *tradeoffs* between different security properties, and between them and other system properties; and it is possible to develop system composition and adaptation techniques allowing systems to behave adequately in changing environments.

Work in this general area includes: (1) work of Denker, Meseguer, and Talcott on the specification and analysis of cryptographic protocols using Maude [29,30] (see also [92]); (2) work of Basin and Denker on an experimental comparison of the advantages and disadvantages of using Maude versus using Haskell to analyze security protocols [6]; (3) work of Millen and Denker at SRI using Maude to give a formal semantics to their new cryptographic protocol specification language CAPSL, and to endow CAPSL with an execution and formal analysis environment [31,32,33,34]; (4) work of Gutierrez-Nolasco, Venkatasubramanian, Stehr, and Talcott on the Secure Spread protocol [56]; (5) work of Gunter, Goodloe, and Stehr on the formal specification and analysis of the L3A security protocol [55]; (6) work of Cervesato, Stehr, and Reich on the rewriting logic semantics of the MSR security specification formalism, leading to the first executable environment for MSR [11,97]; and (7) the already-mentioned work by Agha, Gunter, Greenwald, Khanna, Meseguer, Sen, and Thati on the specification and analysis of a DoS-resistant TCP/IP protocol using probabilistic rewrite theories [1].

A related technique with important security applications is *narrowing*, a symbolic procedure like rewriting, except that rules, instead of being applied by matching a subterm, are applied by unifying the lefthand side with a nonvariable subterm. Traditionally, narrowing has been used as a method to solve equations in a confluent and terminating equational theory. In rewriting logic, narrowing has been generalized by Meseguer and Thati to a semi-decision procedure for *symbolic reachability analysis* [79]. That is, instead of solving equational goals $\exists \boldsymbol{x}.\ t = t'$, we solve reachability goals $\exists \boldsymbol{x}.\ t \longrightarrow t'$. The relevant point for security applications is that, since narrowing with a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ is performed *modulo* the equations E, this allows more sophisticated analyses than those performed under the usual Dolev-Yao "perfect cryptography assumption". It is well-known that protocols that had been proved secure under this assumption can be broken if an attacker uses knowledge of the algebraic properties satisfied by the underlying cryptographic functions. In rewriting logic we can specify a cryptographic protocol as a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, and can model those algebraic properties as equations in E. Under suitable assumptions that are typically satisfied by cryptographic protocols, narrowing then gives us a complete semidecision procedure to find attacks *modulo* the equations E; therefore, any attack making use algebraic properties can be found this way [79]. Very recent work in this direction by Escobar, Meadows and Meseguer [47] is using rewriting logic and narrowing to give a precise rewriting semantics to the inference system of one of the most effective analysis tools for cryptographic protocols, namely the NRL Analyzer [68]. Further recent work on narrowing with rewrite theories focuses on: (1) generalizing the procedure to so-called "back-and-forth narrowing," so as to ensure completeness under very general assumptions about the rewrite theory $\mathcal{R}$ [102]; and (2) efficient lazy strategies to restrict as much as possible the narrowing search space [48].

3.5 Bioinformatics Modeling and Analysis

Biology lacks at present adequate mathematical models that can provide something analogous to the analytic and predictive power that mathematical models provide for, say, Physics. Of course, the mathematical models of Chemistry describing, say, molecular structures are still applicable to biochemistry. The problem is that they *do not scale up* to something like a cell, because they are too low-level. One can of course model biological phenomena at different *levels of abstraction*. Higher, more abstract levels seem both the most crucial and the least supported. The most abstract the level, the better the chances to scale up.

All this is analogous to the use of different levels of abstraction to model digital systems. There are great scaling up advantages in treating digital systems and computer designs at a *discrete* level of abstraction, above the continuous level provided by differential equations, or, even lower, the quantum electrodynamics (QED) level. The discrete models, when they can be had, can also be more *robust and predictable*: there is greater difficulty in predicting the behavior of a system that can only be modeled at lower levels. Indeed, the level at which biologists like to reason about cell behavior is typically the discrete level; however, at present descriptions at this level consist of semi-formal notations for the elementary reactions, together with informal and potentially ambiguous notations for things like pathways, cycles, feedback, etc. Furthermore, such notations are static and therefore offer little predictive power. What are needed are *new computable mathematical models of cell biology* that are at a high enough level of abstraction so that they fit biologist's intuitions, make those intuitions mathematically precise, and provide biologists with the *predictive power* of mathematical models, so that the consequences of their hypotheses and theories can be analyzed, and can then suggest laboratory experiments to prove them or disprove them.

Rewriting logic seems ideally suited for this task. The basic idea is that we can *model a cell as a concurrent system* whose concurrent transitions are precisely its biochemical reactions. In fact, the chemical notation for a reaction like $A\,B \longrightarrow C\,D$ is *exactly* a rewriting notation. In this way we can develop *symbolic bioinformatic models* which we can then analyze in their dynamic behavior just as we would analyze any other rewrite theory.

Implicit in the view of modeling a cell as a rewrite theory (Σ, E, R) is the idea of modeling the cell states as elements of an algebraic data type specified by (Σ, E). This can of course be done at different levels of abstraction. We can for example introduce basic sorts such as `AminoAcid`, `Protein`, and `DNA` and declare the most basic building blocks as constants of the appropriate sort. For example,

```
ops T U Y S K P : -> AminoAcid .
ops 14-3-3 cdc37 GTP Hsp90 Raf1 Ras : -> Protein .
```

But sometimes a protein is *modified*, for example by one of its component amino acids being phosphorylated at a particular *site* in its structure. Consider for example the c-Raf protein, denoted above by `Raf1`. Two of its `S` amino acid components can be phosphorilated at sites, say, 259 and 261. We then obtain a modified protein that we denote by the symbolic expression,

```
[Raf1 \ phos(S 259) phos(S 621)]
```

A fragment, relevant for this example, of the signature Σ needed to symbolically express and analyze such modified proteins is given by the following sorts, subsorts, and operators:

```
sorts Site Modification ModSet .
subsort Modification < ModSet .

op phos : Site -> Modification .
op none : -> ModSet .
op __ : ModSet ModSet -> ModSet [assoc comm id: none] .
op __ : AminoAcid MachineInt -> Site .
op [_\_] : Protein ModSet -> Protein [right id: none] .
```

Proteins can stick together to form *complexes*. This can be modeled by the following subsort and operator declarations

```
sort Complex .
subsort Protein < Complex .
op _:_ : Complex Complex -> Complex [comm] .
```

In the cell, proteins and other molecules exist in "soups," such as the cytosol, or the soups of proteins inside the cell and nucleus membranes, or the soup inside the nucleus. All these soups, as well as the "structured soups" making up the different structures of the cell, can be modeled by the following fragment of sort, subsort, and operator declarations,

```
sort Soup .
subsort Complex < Soup .
op __ : Soup Soup -> Soup [assoc comm] .
op cell{_{_}} : Soup Soup -> Soup .
op nucl{_{_}} : Soup Soup -> Soup .
```

that is, soups are made up out of complexes, including individual proteins, by means of the above binary "soup union" operator (with juxtaposition syntax) that combines two soups into a bigger soup. This union operator models the fluid nature of soups by obeying *associative and commutative* laws. A *cell* is then a *structured soup*, composed by the above `cell` operator out of two subsoups, namely the soup in the membrane, and that inside the membrane; but this second soup is itself also structured by the cytoplasm and the nucleus. Finally, the nucleus itself is made up of two soups, namely that in the nucleus membrane, and that inside the nucleus, which are composed using the above `nucl` operator. Then, the following expression gives a partial description of a cell:

```
cell{cm  (Ras : GTP) {cyto
      (([Raf1 \ phos(S 259)phos(S 621)] : (cdc37 : Hsp90)) : 14-3-3)
                                                              nucl{nm{n}}}}
```

where `cm` denotes the rest of the soup in the cell membrane, `cyto` denotes the rest of the soup in the cytoplasm, and `nm` and `n` likewise denote the remaining soups in the nucleus membrane and inside the nucleus.

Once we have cell states defined as elements of an algebraic data type specified by (Σ, E), the only missing information has to do with cell *dynamics*, that is, with its biochemical reactions. They can be modeled by suitable rewrite rules R, giving us a full model (Σ, E, R). Consider, for example, the following reaction described in a survey by Kolch [61]:

> "Raf-1 resides in the cytosol, tied into an inactive state by the binding of a 14-3-3 dimer to phosphosterines-259 and -621. When activation ensues, Ras-GTP binding . . . brings Raf-1 to the membrane."

We can model this reaction by the following rewrite rule:

```
 rl[10]: {CM  (Ras : GTP) {CY
 (([Raf1 \ phos(S 259)phos(S 621)] : (cdc37 : Hsp90)) : 14-3-3) }}
=>
 {CM ((Ras : GTP) :
 (([Raf1 \ phos(S 259)phos(S 621)] : (cdc37 : Hsp90)) : 14-3-3))
                                                             {CY}} .
```

where CM and CY are variables of sort Soup, representing, respectively, the rest of the soup in the cell membrane, and the rest of the soup inside the cell (including the nucleus). Note that in the new state of the cell represented by the righthand side of the rule, the complex has indeed migrated to the membrane.

Given a type of cell specified as a rewrite theory (Σ, E, R), rewriting logic then allows us to reason about the *complex changes* that are possible in the system, given the basic changes specified by R. That is, we can then use (Σ, E, R) together with Maude and its supporting formal tools to simulate, study, and analyze *cell dynamics*. In particular, we can study in this way *biological pathways*, that is, complex processes involving chains of biological reactions and leading to important cell changes. In particular we can:

- observe progress in time of the cell state by *symbolic simulation*, obtaining a corresponding trace;
- answer questions of *reachability* from a given cell state to another state satisfying some property; this can be done both *forwards* and *backwards*;
- answer more complex questions by *model checking* LTL properties; and
- do *meta-analysis* of proposed models of the cell to weed out spurious conjectures and to identify *consequences* of a given model that could be settled by *experimentation*.

Since the first research in this direction [43], on which the above summary is based, this line of research has been vigorously advanced, both in developing more sophisticated analyses of cell behavior in biological pathways, and in developing useful notations and visualization tools that can represent the Maude-based analyses in forms more familiar to biologists [44,101]. In particular, [101] contains a good discussion of related work in this area, using other formalisms, such as Petri nets or process calculi, that can also be understood as particular rewrite theories; and shows how cell behavior can be modeled with rewrite rules and can be analyzed *at different levels of abstraction*, and even across such levels. In fact, I view this research area as ripe for bringing in more advanced specification and analysis techniques —for example, techniques based on real-time and probabilistic rewrite theories as introduced in this paper— so as to develop a *range of complementary models* for cell biology. In this way, aspects such as the probabilistic nature of cell reactions, their dependence on the concentration of certain substances, and their real-time behavior could also be modeled, and even more sophisticated analyses could be developed.

4 Where to Go from Here?

This finishes the sampler. I have tried to give you a feeling for some of the main ideas of rewriting logic, some of its theoretical extensions to cover entire new areas, and some of its exciting application areas. I did not promise an overview: only an appetizer. If you would like to know more, I would recommend the roadmap in [67] for a good overview: it is a little dated by now, and there are many new references that nobody has yet managed to gather together, but this sampler puts the roadmap up to date in some areas; and reading both papers together is the best suggestion I can currently give for an introduction.

Acknowledgments. This research has been supported by ONR Grant N00014-02-1-0715 and NSF Grant CCR-0234524. I thank the ICTAC05 organizers for kindly giving me the opportunity of presenting these ideas. Many of them have been developed in joint work with students and colleagues; and many other ideas are not even my own work. Besides the credit given in each case through the references, I would like to point out that: (1) the recent work on foundations of rewriting logic is joint work with Roberto Bruni; (2) the work on Maude is joint work with all the members of the Maude team at SRI, UIUC, and the Universities of Madrid and Málaga; (3) the work on Maude tools is joint work with Manuel Clavel, Francisco Durán, Joseph Hendrix, Salvador Lucas, Claude Marché, Hitoshi Ohsaki, Peter Ölveczky, Miguel Palomino, and Xavier Urbain; (4) the work on the rewriting logic semantics project is a fairly wide collective effort, in which I have collaborated most closely with Feng Chen, Azadeh Farzan, and Grigore Roşu at UIUC, and with Christiano Braga at the Universidade Federal Fluminense in Brazil; (5) my part of the work on real-time rewrite theories is joint work with Peter Ölveczky at the University of Oslo; (6) the work on probabilistic rewrite theories is joint work with Gul Agha, Nirman Kumar, and Koushik Sen at UIUC; (7) security applications is again a wide effort in which I have collaborated most closely with Santiago Escobar, Grit Denker, Michael Greenwald, Carl Gunter, Sanjiv Khanna, Cathy Meadows, Koushik Sen, Carolyn Talcott, and Prasanna Thati; and (8) I was only involved in the early stages of the bioinformatics work, which has been continued by the Pathway Logic Team at SRI International.

References

1. G. Agha, C. Gunter, M. Greenwald, S. Khanna, J. Meseguer, K. Sen, and P. Thati. Formal modeling and analysis of DoS using probabilistic rewrite theories. In *Workshop on Foundations of Computer Security (FCS'05) (Affiliated with LICS'05)*, 2005.
2. G. Agha, J. Meseguer, and K. Sen. PMaude: Rewrite-based specification language for probabilistic object systems. In *3rd Workshop on Quantitative Aspects of Programming Languages (QAPL'05)*, 2005.
3. W. Ahrendt, A. Roth, and R. Sasse. Automatic validation of transformation rules for Java verification against a rewriting semantics. Manuscript, June 2005.
4. D. Basin, M. Clavel, and J. Meseguer. Reflective metalogical frameworks. *ACM Transactions on Computational Logic*, 2004.
5. D. Basin, M. Clavel, and J. Meseguer. Rewriting logic as a metalogical framework. In S. Kapoor and S. Prasad, editors, *Twentieth Conference on the Foundations of Software Technology and Theoretical Computer Science, New Delhi, India, December 13–15, 2000, Proceedings*, volume 1974 of *Lecture Notes in Computer Science*, pages 55–80. Springer-Verlag, 2000.

6. D. Basin and G. Denker. Maude versus Haskell: An experimental comparison in security protocol analysis. In Futatsugi [52], pages 235–256. `http://www.elsevier.nl/locate/entcs/volume36.html`.
7. P. Borovanský, C. Kirchner, H. Kirchner, and P.-E. Moreau. ELAN from a rewriting logic point of view. *Theoretical Computer Science*, 285:155–185, 2002.
8. C. Braga. *Rewriting Logic as a Semantic Framework for Modular Structural Operational Semantics*. PhD thesis, Departamento de Informática, Pontificia Universidade Católica de Rio de Janeiro, Brasil, 2001.
9. C. Braga and J. Meseguer. Modular rewriting semantics in practice. in Proc. *WRLA'04*, ENTCS.
10. R. Bruni and J. Meseguer. Generalized rewrite theories. In J. Baeten, J. Lenstra, J. Parrow, and G. Woeginger, editors, *Proceedings of ICALP 2003, 30th International Colloquium on Automata, Languages and Programming*, volume 2719 of *Springer LNCS*, pages 252–266, 2003.
11. I. Cervesato and M.-O. Stehr. Representing the msr cryptoprotocol specification language in an extension of rewriting logic with dependent types. In P. Degano, editor, *Proc. Fifth International Workshop on Rewriting Logic and its Applications (WRLA'2004)*. Elsevier ENTCS, 2004. Barcelona, Spain, March 27 - 28, 2004.
12. F. Chalub. An implementation of modular SOS in maude. Master's thesis, Universidade Federal Fluminense, May 2005. `http://www.ic.uff.br/~frosario/dissertation.pdf`.
13. F. Chalub and C. Braga. A Modular Rewriting Semantics for CML. *Journal of Universal Computer Science*, 10(7):789–807, July 2004. `http://www.jucs.org/jucs_10_7/a_modular_rewriting_semantics`.
14. F. Chen, G. Roşu, and R. P. Venkatesan. Rule-based analysis of dimensional safety. In *Rewriting Techniques and Applications (RTA'03)*, volume 2706 of *Springer LNCS*, pages 197–207, 2003.
15. M. Clavel. *Reflection in Rewriting Logic: Metalogical Foundations and Metaprogramming Applications*. CSLI Publications, 2000.
16. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, and J. Meseguer. Metalevel computation in Maude. In Kirchner and Kirchner [60], pages 3–24. `http://www.elsevier.nl/locate/entcs/volume15.html`.
17. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.
18. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. Maude 2.0 Manual. June 2003, `http://maude.cs.uiuc.edu`.
19. M. Clavel, F. Durán, S. Eker, and J. Meseguer. Building equational proving tools by reflection in rewriting logic. In *CAFE: An Industrial-Strength Algebraic Formal Method*, pages 1–31. Elsevier, 2000. `http://maude.cs.uiuc.edu`.
20. M. Clavel, F. Durán, S. Eker, J. Meseguer, and M.-O. Stehr. Maude as a formal meta-tool. In J. M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 — Formal Methods, World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 20–24, 1999 Proceedings, Volume II*, volume 1709 of *Lecture Notes in Computer Science*, pages 1684–1703. Springer-Verlag, 1999.
21. M. Clavel, F. Durán, and N. Martí-Oliet. Polytypic programming in Maude. In Futatsugi [52], pages 339–360. `http://www.elsevier.nl/locate/entcs/volume36.html`.

22. M. Clavel and J. Meseguer. Reflection and strategies in rewriting logic. In J. Meseguer, editor, *Proceedings First International Workshop on Rewriting Logic and its Applications, WRLA'96, Asilomar, California, September 3–6, 1996*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 125–147. Elsevier, Sept. 1996. `http://www.elsevier.nl/locate/entcs/volume4.html`.
23. M. Clavel and J. Meseguer. Reflection in conditional rewriting logic. *Theoretical Computer Science*, 285:245–288, 2002.
24. M. Clavel, J. Meseguer, and M. Palomino. Reflection in membership equational logic, many-sorted equational logic, horn logic with equality, and rewriting logic. In F. Gadducci and U. Montanari, editors, *Proc. 4th. Intl. Workshop on Rewriting Logic and its Applications*. ENTCS, Elsevier, 2002.
25. M. Clavel and M. Palomino. The ITP tool's manual. Universidad Complutense, Madrid, April 2005, `http://maude.sip.ucm.es/itp/`.
26. M. Clavel and J. Santa-Cruz. ASIP+ITP: A verification tool based on algebraic semantics. To appear in *Proc. PROLE'05*, `http://maude.sip.ucm.es/~clavel/pubs/`.
27. M. d'Amorim and G. Roşu. An Equational Specification for the Scheme Language. In *Proceedings of the 9th Brazilian Symposium on Programming Languages (SBLP'05)*, to appear 2005. Also Technical Report No. UIUCDCS-R-2005-2567, April 2005.
28. G. Denker, J. Meseguer, and C. Talcott. Rewriting semantics of meta-objects and composable distributed services. ENTCS, Elsevier, 2000. Proc. 3rd. Intl. Workshop on Rewriting Logic and its Applications.
29. G. Denker, J. Meseguer, and C. L. Talcott. Protocol specification and analysis in Maude. In N. Heintze and J. Wing, editors, *Proceedings of Workshop on Formal Methods and Security Protocols, June 25, 1998, Indianapolis, Indiana*, 1998. `http://www.cs.bell-labs.com/who/nch/fmsp/index.html`.
30. G. Denker, J. Meseguer, and C. L. Talcott. Formal specification and analysis of active networks and communication protocols: The Maude experience. In D. Maughan, G. Koob, and S. Saydjari, editors, *Proceedings DARPA Information Survivability Conference and Exposition, DISCEX 2000, Hilton Head Island, South Carolina, January 25–27, 2000*, pages 251–265. IEEE Computer Society Press, 2000. `http://schafercorp-ballston.com/discex/`.
31. G. Denker and J. Millen. CAPSL and CIL language design: A common authentication protocol specification language and its intermediate language. Technical Report SRI-CSL-99-02, Computer Science Laboratory, SRI International, 1999. `http://www.csl.sri.com/~denker/pub_99.html`.
32. G. Denker and J. Millen. CAPSL intermediate language. In N. Heintze and E. Clarke, editors, *Proceedings of Workshop on Formal Methods and Security Protocols, FMSP'99, July 1999, Trento, Italy*, 1999. `http://www.cs.bell-labs.com/who/nch/fmsp99/program.html`.
33. G. Denker and J. Millen. CAPSL integrated protocol environment. In D. Maughan, G. Koob, and S. Saydjari, editors, *Proceedings DARPA Information Survivability Conference and Exposition, DISCEX 2000, Hilton Head Island, South Carolina, January 25-27, 2000*, pages 207–222. IEEE Computer Society Press, 2000. `http://schafercorp-ballston.com/discex/`.
34. G. Denker and J. Millen. The CAPSL integrated protocol environment. Technical Report SRI-CSL-2000-02, Computer Science Laboratory, SRI International, 2000. `http://www.csl.sri.com/~denker/pub_99.html`.
35. F. Durán. A reflective module algebra with applications to the Maude language. Ph.D. Thesis, University of Málaga, 1999.

36. F. Durán. Coherence checker and completion tools for Maude specifications. Manuscript, Computer Science Laboratory, SRI International, `http://maude.cs.uiuc.edu/papers`, 2000.
37. F. Durán. The extensibility of Maude's module algebra. In T. Rus, editor, *Algebraic Methodology and Software Technology, 8th International Conference, AMAST 2000, Iowa City, Iowa, USA, May 20–27, 2000, Proceedings*, volume 1816 of *Lecture Notes in Computer Science*, pages 422–437. Springer-Verlag, 2000.
38. F. Durán. Termination checker and Knuth-Bendix completion tools for Maude equational specifications. Manuscript, Computer Science Laboratory, SRI International, `http://maude.cs.uiuc.edu/papers`, 2000.
39. F. Durán, S. Lucas, J. Meseguer, C. Marché, and X. Urbain. Proving termination of membership equational programs. In P. Sestoft and N. Heintze, editors, *Proc. of ACM SIGPLAN 2004 Symposium on Partial Evaluation and Program Manipulation, PEPM'04*, pages 147–158. ACM Press, 2004.
40. F. Durán and J. Meseguer. An extensible module algebra for Maude. In Kirchner and Kirchner [60], pages 185–206. `http://www.elsevier.nl/locate/entcs/volume15.html`.
41. F. Durán and J. Meseguer. A Church-Rosser checker tool for Maude equational specifications. Manuscript, Computer Science Laboratory, SRI International, `http://maude.cs.uiuc.edu/papers`, 2000.
42. F. Durán and J. Meseguer. On parameterized theories and views in Full Maude 2.0. In K. Futatsugi, editor, *Proc. 3rd. Intl. Workshop on Rewriting Logic and its Applications*. ENTCS, Elsevier, 2000.
43. S. Eker, M. Knapp, K. Laderoute, P. Lincoln, J. Meseguer, and K. Sonmez. Pathway logic: Symbolic analysis of biological signaling. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 400–412, January 2002.
44. S. Eker, M. Knapp, K. Laderoute, P. Lincoln, and C. Talcott. Pathway Logic: executable models of biological networks. In F. Gadducci and U. Montanari, editors, *Proc. 4th. Intl. Workshop on Rewriting Logic and its Applications*. ENTCS, Elsevier, 2002.
45. S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker. In F. Gadducci and U. Montanari, editors, *Proc. 4th. Intl. Workshop on Rewriting Logic and its Applications*. ENTCS, Elsevier, 2002.
46. S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker and its implementation. In *Model Checking Software: Proc.* 10^{th} *Intl. SPIN Workshop*, volume 2648, pages 230–234. Springer LNCS, 2003.
47. S. Escobar, C. Meadows, and J. Meseguer. A rewriting-based inference system for the NRL Protocol Analyzer. Submitted for publication, 2005.
48. S. Escobar, J. Meseguer, and P. Thati. Natural narrowing for general term rewriting systems. In J. Giesl, editor, *Proceedings of the 16th Intl. Conference on Term Rewriting and Applications, RTA 2005*, pages 279–293. Springer LNCS Vol. 3467, 2005.
49. A. Farzan, F. Cheng, J. Meseguer, and G. Roşu. Formal analysis of Java programs in JavaFAN. in Proc. CAV'04, Springer LNCS, 2004.
50. A. Farzan and J. Meseguer. Partial order reduction for rewriting semantics of programming languages. Technical Report UIUCDCS-R-2005-2598, CS Dept., University of Illinois at Urbana-Champaign, June 2005.
51. A. Farzan, J. Meseguer, and G. Roşu. Formal JVM code analysis in JavaFAN. in Proc. *AMAST'04*, Springer LNCS 3116, 132–147, 2004.

52. K. Futatsugi, editor. *Proceedings Third International Workshop on Rewriting Logic and its Applications, WRLA 2000, Kanazawa, Japan, September 18–20, 2000*, volume 36 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2000. `http://www.elsevier.nl/locate/entcs/volume36.html`.
53. K. Futatsugi and R. Diaconescu. *CafeOBJ Report*. World Scientific, AMAST Series, 1998.
54. P. Glynn. The role of generalized semi-Markov processes in simulation output analysis, 1983.
55. C. Gunter, A. Goodloe, and M.-O. Stehr. Formal prototyping in early stages of protocol design. In *In* Proceedings of the Workshop on Issues in the Theory of Security (WITS'05). January 10-11, 2005, Long Beach, California. To appear in the ACM Digital Library. Paper available at `http://formal.cs.uiuc.edu/stehr/l3a-wits.pdf`.
56. S. Gutierrez-Nolasco, N. Venkatasubramanian, M.-O. Stehr, and C. L. Talcott. Exploring adaptability of secure group communication using formal prototyping techniques. In *Proceedings of the 3rd Workshop on Reflective and Adaptive Middleware (RM2004). October 19, 2004, Toronto, Ontario, Canada*. To appear in ACM Digital Library. Extended version available at `http://formal.cs.uiuc.edu/stehr/spread_eng.html`.
57. J. Hendrix, J. Meseguer, and M. Clavel. A sufficient completeness reasoning tool for partial specifications. In J. Giesl, editor, *Proceedings of the 16th Intl. Conference on Term Rewriting and Applications, RTA 2005*, pages 165–174. Springer LNCS Vol. 3467, 2005.
58. G. Holzmann. *The Spin Model Checker - Primer and Reference Manual*. Addison-Wesley, 2003.
59. E. B. Johnsen, O. Owe, and E. W. Axelsen. A runtime environment for concurrent objects with asynchronous method calls. In N. Martí-Oliet, editor, *Proc. 5th. Intl. Workshop on Rewriting Logic and its Applications*. ENTCS, Elsevier, 2004.
60. C. Kirchner and H. Kirchner, editors. *Proceedings Second International Workshop on Rewriting Logic and its Applications, WRLA'98, Pont-à-Mousson, France, September 1–4, 1998*, volume 15 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998. `http://www.elsevier.nl/locate/entcs/volume15.html`.
61. W. Kolch. Meaningful relationships: the regulation of the Ras/Raf/MEK/ERK pathway by protein interactions. *Biochem. J.*, 351:289–305, 2000.
62. P. Kosiuczenko and M. Wirsing. Timed rewriting logic with application to object-oriented specification. Technical report, Institut für Informatik, Universität München, 1995.
63. N. Kumar, K. Sen, J. Meseguer, and G. Agha. Probabilistic rewrite theories: Unifying models, logics and tools. Technical Report UIUCDCS-R-2003-2347, University of Illinois at Urbana-Champaign, May 2003.
64. N. Kumar, K. Sen, J. Meseguer, and G. Agha. A rewriting based model of probabilistic distributed object systems. Proc. of Formal Methods for Open Object-Based Distributed Systems, FMOODS 2003, Springer LNCS Vol. 2884, 2003.
65. E. Lien. Formal modeling and analysis of the NORM multicast protocol in Real-Time Maude. Master's thesis, Dept. of Linguistics, University of Oslo, 2004. `http://wo.uio.no/as/WebObjects/theses.woa/wo/0.3.9`.
66. N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd. Edition*, pages 1–87. Kluwer Academic Publishers, 2002. First published as SRI Tech. Report SRI-CSL-93-05, August 1993.

67. N. Martí-Oliet and J. Meseguer. Rewriting logic: roadmap and bibliography. *Theoretical Computer Science*, 285:121–154, 2002.
68. C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.
69. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
70. J. Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In *Proc. CONCUR'96, Pisa, August 1996*, pages 331–372. Springer LNCS 1119, 1996.
71. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proc. WADT'97*, pages 18–61. Springer LNCS 1376, 1998.
72. J. Meseguer. Research directions in rewriting logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, NATO Advanced Study Institute, Marktoberdorf, Germany, July 29 – August 6, 1997*. Springer-Verlag, 1999.
73. J. Meseguer. Software specification and verification in rewriting logic. In M. Broy and M. Pizka, editors, *Models, Algebras, and Logic of Engineering Software, NATO Advanced Study Institute, Marktoberdorf, Germany, July 30 – August 11, 2002*, pages 133–193. IOS Press, 2003.
74. J. Meseguer. Localized fairness: A rewriting semantics. In J. Giesl, editor, *Proceedings of the 16th Intl. Conference on Term Rewriting and Applications, RTA 2005*, pages 250–263. Springer LNCS Vol. 3467, 2005.
75. J. Meseguer and C. Braga. Modular rewriting semantics of programming languages. in Proc. AMAST'04, Springer LNCS 3116, 364–378, 2004.
76. J. Meseguer and G. Roşu. Rewriting logic semantics: From language specifications to formal analysis tools. In *Proc. Intl. Joint Conf. on Automated Reasoning IJCAR'04, Cork, Ireland, July 2004*, pages 1–44. Springer LNAI 3097, 2004.
77. J. Meseguer and G. Roşu. The rewriting logic semantics project. In *Proc. of SOS 2005*. Elsevier, ENTCS, 2005. To appear.
78. J. Meseguer and C. Talcott. Semantic models for distributed object reflection. In *Proceedings of ECOOP'02, Málaga, Spain, June 2002*, pages 1–36. Springer LNCS 2374, 2002.
79. J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to the verification of cryptographic protocols. In N. Martí-Oliet, editor, *Proc. 5th. Intl. Workshop on Rewriting Logic and its Applications*. ENTCS, Elsevier, 2004.
80. P. D. Mosses. Modular structural operational semantics. *J. Log. Algebr. Program.*, 60–61:195–228, 2004.
81. P. Ölveczky, M. Keaton, J. Meseguer, C. Talcott, and S. Zabele. Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. In *Proc. of FASE'01, 4th Intl. Conf. on Fundamental Approaches to Software Engineering*, Springer LNCS. Springer-Verlag, 2001.
82. P. Ölveczky and J. Meseguer. Real-Time Maude 2.0. in *Proc. WRLA'04*, ENTCS.
83. P. C. Ölveczky. *Specification and Analysis of Real-Time and Hybrid Systems in Rewriting Logic*. PhD thesis, University of Bergen, Norway, 2000. `http://maude.csl.sri.com/papers`.
84. P. C. Ölveczky, P. Kosiuczenko, and M. Wirsing. An object-oriented algebraic steam-boiler control specification. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *The Steam-Boiler Case Study Book*, pages 379–402. Springer-Verlag, 1996. Vol. 1165.

85. P. C. Ölveczky and J. Meseguer. Specifying real-time systems 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://www.elsevier.nl/cas/tree/store/tcs/free/noncas/pc/volume4.htm`.
86. P. C. Ölveczky and J. Meseguer. Real-Time Maude: a tool for simulating and analyzing real-time and hybrid systems. ENTCS, Elsevier, 2000. Proc. 3rd. Intl. Workshop on Rewriting Logic and its Applications.
87. P. C. Ölveczky and J. Meseguer. Specification of real-time and hybrid systems in rewriting logic. *Theoretical Computer Science*, 285:359–405, 2002.
88. M. Palomino. A predicate abstraction tool for maude. Documentation and tool available at `http://maude.sip.ucm.es/~miguelpt/bibliography.html`.
89. E. Parzen. *Modern Probability Theory and its Applications*. Wiley, 1960.
90. M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.
91. G. Roşu, R. P. Venkatesan, J. Whittle, and L. Leustean. Certifying optimality of state estimation programs. In *Computer Aided Verification (CAV'03)*, pages 301–314. Springer, 2003. LNCS 2725.
92. D. E. Rodríguez. Case studies in the specification and analysis of protocols in Maude. In Futatsugi [52], pages 257–275.
`http://www.elsevier.nl/locate/entcs/volume36.html`.
93. R. Sasse. Taclets vs. rewriting logic – relating semantics of Java. Master's thesis, Fakultät für Informatik, Universität Karlsruhe, Germany, May 2005. Technical Report in Computing Science No. 2005-16,
`http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=ira/2005/16`.
94. R. Segala. *Modelling and Verification of Randomized Distributed Real Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.
95. K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In *17th conference on Computer Aided Verification (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science (To Appear)*, Edinburgh, Scotland, July 2005. Springer.
96. L. Steggles and P. Kosiuczenko. A timed rewriting logic semantics for SDL: a case study of the alternating bit protocol. *Proc. 2nd Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, Vol. 15, North Holland, 1998.
97. M.-O. Stehr, I. Cervesato, and S. Reich. An execution environment for the MSR cryptoprotocol specification language.
`http://formal.cs.uiuc.edu/stehr/msr.html`.
98. M.-O. Stehr and C. Talcott. PLAN in Maude: Specifying an active network programming language. In F. Gadducci and U. Montanari, editors, *Proc. 4th. Intl. Workshop on Rewriting Logic and its Applications*. ENTCS, Elsevier, 2002.
99. M.-O. Stehr and C. L. Talcott. Practical techniques for language design and prototyping. In J. L. Fiadeiro, U. Montanari, and M. Wirsing, editors, *Abstracts Collection of the Dagstuhl Seminar 05081 on Foundations of Global Computing. February 20 – 25, 2005. Schloss Dagstuhl, Wadern, Germany.*, 2005.
100. W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton, 1994.
101. C. Talcott, S. Eker, M. Knapp, P. Lincoln, and K. Laderoute. Pathway logic modeling of protein functional domains in signal transduction. In *Proceedings of the Pacific Symposium on Biocomputing*, January 2004.
102. P. Thati and J. Meseguer. Complete symbolic reachability analysis using back-and-forth narrowing. To appear in *Proc. CALCO 2005*, Springer LNCS, 2005.

103. P. Thati, K. Sen, and N. Martí-Oliet. An executable specification of asynchronous Pi-Calculus semantics and may testing in Maude 2.0. In F. Gadducci and U. Montanari, editors, *Proc. 4th. Intl. Workshop on Rewriting Logic and its Applications*. ENTCS, Elsevier, 2002.
104. S. Thordvalsen. Modeling and analysis of the OGDC wireless sensor network algorithm in Real-Time Maude. Master's thesis, Dept. of Informatics, University of Oslo, 2005. `http://heim.ifi.uio.no/~peterol/RealTimeMaude/OGDC/`.
105. A. Verdejo. *Maude como marco semántico ejecutable*. PhD thesis, Facultad de Informática, Universidad Complutense, Madrid, Spain, 2003.
106. A. Verdejo and N. Martí-Oliet. Executable structural operational semantics in Maude. Manuscript, Dto. Sistemas Informáticos y Programación, Universidad Complutense, Madrid, August 2003.
107. A. Verdejo and N. Martí-Oliet. Implementing CCS in Maude. In *Proc. FORTE/PSTV 2000*, pages 351–366. IFIP, vol. 183, 2000.
108. A. Verdejo and N. Martí-Oliet. Implementing CCS in Maude 2. In F. Gadducci and U. Montanari, editors, *Proc. 4th. Intl. Workshop on Rewriting Logic and its Applications*. ENTCS, Elsevier, 2002.
109. P. Viry. Equational rules for rewriting logic. *Theoretical Computer Science*, 285:487–517, 2002.