Equational Abstractions

José Meseguer[1], Miguel Palomino[1,2], and Narciso Martí-Oliet[2]

[1] Computer Science Department, University of Illinois at Urbana-Champaign
[2] Departamento de Sistemas Informáticos, Universidad Complutense de Madrid
meseguer@cs.uiuc.edu {miguelpt,narciso}@sip.ucm.es

Abstract. Abstraction reduces the problem of whether an infinite state system satisfies a temporal logic property to model checking that property on a finite state abstract version. The most common abstractions are quotients of the original system. We present a simple method of defining quotient abstractions by means of equations collapsing the set of states. Our method yields the minimal quotient system together with a set of proof obligations that guarantee its executability and can be discharged with tools such as those in the Maude formal environment.

1 Introduction

Abstraction techniques (see for example [1–15]) allow reducing the problem of whether an infinite state system, or a finite but too large one, satisfies a temporal logic property to model checking that property on a finite state abstract version. The most common way of defining such abstractions is by a *quotient* of the original system's set of states, together with abstract versions of the transitions and the predicates. Many methods differ in their details but agree on their general use of a quotient map. There is always a minimal system (Kripke structure) making this quotient map a simulation. We present a simple method to build minimal quotient abstractions in an equational way. The method assumes that the concurrent system has been specified by means of a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, with (Σ, E) an equational theory specifying the set of states as an algebraic data type, and R specifying the system's transitions as a set of rewrite rules. The method consists on adding more equations, say E', to get a quotient system specified by the rewrite theory $\mathcal{R}/E' = (\Sigma, E \cup E', R)$. We call such a system an *equational abstraction* of $\mathcal{R}$. This equational abstraction is useful for model checking purposes if: (1) $\mathcal{R}/E'$ is an *executable* rewrite theory in a appropriate sense; and (2) the state predicates are *preserved* by the quotient simulation. Requirements (1) and (2) are *proof obligations* that can be discharged by theorem proving methods. Our approach can be mechanized using the rewriting logic language Maude [16] and its associated LTL model checker [17], inductive theorem prover [18], Church-Rosser checker [19], and coherence checker [20]. Our present experience with case studies, involving different abstractions discussed in the literature, suggests a fairly wide applicability for this method.

After summarizing LTL prerequisites (Sect. 2) and discussing simulations (Sect. 3), we explain in Sect. 4 how a concurrent system specified by a rewrite theory $\mathcal{R}$ has an associated Kripke structure giving semantics to its LTL properties; we also explain how Maude can model check such LTL properties for initial states having finitely many reachable states. Equational abstractions, their associated proof methods, and case studies are discussed in Sect. 5. Sect. 6 discusses related work and future research. Proofs of all the results in this paper and all case studies can be found in [21].

2 Prerequisites on Kripke Structures and LTL

To specify the properties of interest about our systems we will use *linear temporal logic* (LTL), which is interpreted in a standard way in Kripke structures. In what follows, we assume a fixed non-empty set of atomic propositions AP.

Definition 1. *A* Kripke structure *is a triple* $\mathcal{A} = (A, \rightarrow_{\mathcal{A}}, L_{\mathcal{A}})$*, where* A *is a set of states,* $\rightarrow_{\mathcal{A}} \subseteq A \times A$ *is a* total *transition relation, and* $L_{\mathcal{A}} : A \rightarrow \mathcal{P}(AP)$ *is a labeling function associating to each state the set of atomic propositions that hold in it.*

We will usually employ the notation $a \rightarrow_{\mathcal{A}} b$ to say that $(a, b) \in \rightarrow_{\mathcal{A}}$. Note that the transition relation must be *total*, that is, for each $a \in A$ there is a $b \in A$ such that $a \rightarrow_{\mathcal{A}} b$. Given an arbitrary relation $\rightarrow$, we write $\rightarrow^{\bullet}$ for the total relation that extends $\rightarrow$ by adding a pair $a \rightarrow^{\bullet} a$ for each a such that there is no b with $a \rightarrow b$. A *path* in a Kripke structure $\mathcal{A}$ is a function $\pi : \mathbb{N} \longrightarrow A$ such that, for each $i \in \mathbb{N}$, $\pi(i) \rightarrow_{\mathcal{A}} \pi(i+1)$.

The syntax of LTL(AP) is given by the following grammar:

$$\varphi = p \in AP \mid \varphi \vee \varphi \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi\,.$$

The semantics of the logic, specifying the satisfaction relation $\mathcal{A}, a \models \varphi$ between a Kripke structure $\mathcal{A}$, an initial state $a \in A$, and $\varphi \in$ LTL(AP), is defined as usual (see for example [4, Sect. 3.1], where $\varphi \,\mathcal{U}\, \psi$ and $\bigcirc \varphi$ are expressed in CTL* notation as $\mathbf{A}(\varphi \mathbf{U} \psi)$ and $\mathbf{AX}\varphi)$. Other Boolean and temporal operators (e.g., $\top$, $\bot$, $\wedge$, $\rightarrow$, $\Box$, $\Diamond$, $\mathcal{R}$, and $\leadsto$) can be defined as syntactic sugar.

It is sometimes useful to restrict ourselves to the *negation-free fragment* LTL$^-(AP)$ of LTL(AP), defined as follows:

$$\varphi = p \in AP \mid \top \mid \bot \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi \mid \varphi \,\mathcal{R}\, \varphi\,.$$

Negation is no longer available in LTL$^-$, and therefore the duals of the basic operators must be considered as basic ones, too. Since LTL$^-$ is a sublogic of LTL, its semantics is the same. Furthermore, in a very practical sense there is no real loss of generality by restricting ourselves to formulas in LTL$^-$, because we can always transform any LTL formula φ into a semantically equivalent LTL$^-$ formula $\hat{\varphi}$. For that, we consider the extended set of atomic propositions $\widehat{AP} = AP \cup \overline{AP}$, where $\overline{AP} = \{\bar{p} \mid p \in AP\}$, and construct $\hat{\varphi}$ by first forming the negation normal form of φ (i.e., all negations are pushed to the atoms), and then replacing each negated atom $\neg p$ by $\bar{p}$. Given $\mathcal{A} = (A, \rightarrow_{\mathcal{A}}, L_{\mathcal{A}})$, we define $\widehat{\mathcal{A}} = (A, \rightarrow_{\mathcal{A}}, L_{\widehat{\mathcal{A}}})$ where $L_{\widehat{\mathcal{A}}}(a) = L_{\mathcal{A}}(a) \cup \{\bar{p} \in \overline{AP} \mid p \notin L_{\mathcal{A}}(p)\}$. Then we have, $\mathcal{A}, a \models \varphi \iff \widehat{\mathcal{A}}, a \models \hat{\varphi}$.

3 Simulations

We present a notion of simulation similar to that in [4], but somewhat more general (simulations in [4] essentially correspond to our *strict* simulations).

Definition 2. *Given Kripke structures $\mathcal{A} = (A, \rightarrow_{\mathcal{A}}, L_{\mathcal{A}})$ and $\mathcal{B} = (B, \rightarrow_{\mathcal{B}}, L_{\mathcal{B}})$, both having the same set AP of atomic propositions, an AP-*simulation *$H : \mathcal{A} \longrightarrow \mathcal{B}$ of $\mathcal{A}$ by $\mathcal{B}$ is given by a* total *binary relation $H \subseteq A \times B$ such that:*

- *if $a \rightarrow_{\mathcal{A}} a'$ and aHb, then there is $b' \in B$ such that $b \rightarrow_{\mathcal{B}} b'$ and $a'Hb'$, and*
- *if $a \in A$, $b \in B$, and aHb, then $L_{\mathcal{B}}(b) \subseteq L_{\mathcal{A}}(a)$.*

If the relation H is a function, *then we call H an AP-*simulation map. *If both H and H^{-1} are AP-simulations, then we call H an AP-*bisimulation. *Also we call H* strict *if aHb implies $L_{\mathcal{B}}(b) = L_{\mathcal{A}}(a)$.*

The first condition guarantees that there is an abstract path in $\mathcal{B}$ corresponding to each concrete path in $\mathcal{A}$; the second condition guarantees that an abstract state in $\mathcal{B}$ can satisfy only those atomic propositions that hold in all the concrete states in $\mathcal{A}$ that it simulates.

We say that an AP-simulation $H : \mathcal{A} \longrightarrow \mathcal{B}$ *reflects* the satisfaction of an LTL formula $\varphi \in \mathrm{LTL}$ iff $\mathcal{B}, b \models \varphi$ and aHb imply $\mathcal{A}, a \models \varphi$. The following theorem slightly generalizes Thm. 16 in [4]:

Theorem 1. *AP-simulations always reflect satisfaction of $\mathrm{LTL}^{-}(AP)$ formulas. In addition, strict simulations also reflect satisfaction of $\mathrm{LTL}(AP)$ formulas.*

This theorem is the key basis for the method of model checking by abstraction: given an infinite (or too large) system $\mathcal{M}$, find a finitely reachable system $\mathcal{A}$ that simulates it and use model checking to try to prove that φ holds in $\mathcal{A}$; then, by Thm. 1, φ also holds in $\mathcal{M}$. In general, however, we typically only have our concrete system $\mathcal{M}$ and a *surjective* function $h : M \longrightarrow A$ mapping concrete states to a simplified (usually finitely reachable) abstract domain A. In these cases there is a canonical way of constructing a Kripke structure out of h in such a way that h becomes a simulation.

Definition 3. *The* minimal system *$\mathcal{M}^h_{\mathrm{min}}$ corresponding to $\mathcal{M}$ and the surjective function $h : M \longrightarrow A$ is given by the triple $(A, h(\rightarrow_{\mathcal{M}}), L_{\mathcal{M}^h_{\mathrm{min}}})$, where $L_{\mathcal{M}^h_{\mathrm{min}}}(a) = \bigcap_{x \in h^{-1}(a)} L_{\mathcal{M}}(x)$.*

The following proposition is an immediate consequence of the definitions.

Proposition 1. *For all such $\mathcal{M}$ and h, $h : \mathcal{M} \longrightarrow \mathcal{M}^h_{\mathrm{min}}$ is a simulation map.*

Minimal systems can also be seen as quotients. Let $\mathcal{A} = (A, \rightarrow_{\mathcal{A}}, L_{\mathcal{A}})$ be a Kripke structure on AP, and let $\equiv$ be an arbitrary equivalence relation on A. We can use $\equiv$ to define a new Kripke structure, $\mathcal{A}/{\equiv} = (A/{\equiv}, \rightarrow_{\mathcal{A}/\equiv}, L_{\mathcal{A}/\equiv})$, where:

– $[a_1] \rightarrow_{\mathcal{A}/\equiv} [a_2]$ iff there exists $a'_1 \in [a_1]$ and $a'_2 \in [a_2]$ such that $a'_1 \rightarrow_{\mathcal{A}} a'_2$;
– $L_{\mathcal{A}/\equiv}([a]) = \bigcap_{x \in [a]} L_{\mathcal{A}}(x)$.

It is then trivial to check that the projection map to equivalence classes $q_{\equiv} : a \mapsto [a]$ is an AP-simulation map $q_{\equiv} : \mathcal{A} \longrightarrow \mathcal{A}/\equiv$, which we call the *quotient abstraction* defined by $\equiv$. Hence, an equivalent presentation of the minimal system is expressed by the following.

Proposition 2. *Let $\mathcal{M} = (M, \rightarrow_{\mathcal{M}}, L_{\mathcal{M}})$ be a Kripke structure and $h : M \longrightarrow A$ a surjective function. Then, there exists a strict bijective bisimulation map between the Kripke structures $\mathcal{M}^h_{\min}$ and $\mathcal{M}/{\equiv_h}$, where by definition $x \equiv_h y$ iff $h(x) = h(y)$.*

That is, we can perform the abstraction either by mapping the concrete states to an abstract domain or, as we will do in Sect. 5, by identifying some states and thereafter working with the corresponding equivalence classes.

The use of the adjective "minimal" is appropriate since, as pointed out in [3], $\mathcal{M}^h_{\min}$ is the most accurate approximation to $\mathcal{M}$ that is consistent with h. However, it is not always possible to have a computable description of $\mathcal{M}^h_{\min}$. The definition of $\rightarrow_{\mathcal{M}^h_{\min}}$ can be rephrased as $x \rightarrow_{\mathcal{M}^h_{\min}} y$ iff there exist a and b such that $h(a) = x$, $h(b) = y$, and $a \rightarrow_{\mathcal{M}} b$. This relation, even if $\rightarrow_{\mathcal{M}}$ is recursive, is in general only recursively enumerable. However, Sect. 5 develops equational methods that, when successful, yield a computable description of $\mathcal{M}^h_{\min}$.

4 Rewriting Logic Specifications and Model Checking

One can distinguish two specification levels: a *system specification* level, in which the computational system of interest is specified; and a *property specification* level, in which the relevant properties are specified. The main interest of rewriting logic [22] is that it provides a very flexible framework for the system-level specification of concurrent systems. A concurrent system is axiomatized by a *rewrite theory* $\mathcal{R} = (\Sigma, E, R)$, where (Σ, E) is an equational theory describing its set of *states* as the algebraic data type $T_{\Sigma/E,k}$ associated to the initial algebra $T_{\Sigma/E}$ of (Σ, E) by the choice of a type k of states in Σ[3]. The system's *transitions* are axiomatized by the *conditional rewrite rules* R which are of the form $l : t \longrightarrow t' \Leftarrow cond$, with l a label, t and t' Σ-terms, possibly with variables, and *cond* a condition[4]. Under reasonable assumptions about E and R, rewrite theories are *executable* (more on this below). Indeed, there are several rewriting logic

[3] We allow very general equational theories in *membership equational logic* [23], that can have types, subtypes defined by semantic conditions, and operator overloading. The desired set of states is then described by the carrier $T_{\Sigma/E,k}$ of the initial algebra $T_{\Sigma/E}$ for one of those types k, technically called either *sorts* or *kinds* in [23]. The elements of $T_{\Sigma/E}$ are E-equivalence classes of terms $[t]_E$; that is, two terms are equal iff they can be proved so by E.

[4] In this paper we assume that the condition *cond* can involve a conjunction of equations $u = v$ and *memberships* of the form $w : s$ stating that the term w has sort s. The conjunction must hold for a substitution instance θ before we are allowed to rewrite $\theta(t)$ to $\theta(t')$. We also assume that $vars(t') \cup vars(cond) \subseteq vars(t)$.

language implementations, including ELAN [24], CafeOBJ [25], and Maude [16]. We can illustrate rewriting logic specifications by means of a simple example, namely Lamport's bakery protocol [26]. This is an infinite state protocol that achieves mutual exclusion between processes by dispensing a number to each process and serving them in sequential order according to the number they hold. A simple Maude specification for the case of two processes is as follows:

```
mod BAKERY is protecting NAT .
  sorts Mode State .
  ops sleep wait crit : -> Mode .
  op <_,_,_,_> : Mode Nat Mode Nat -> State .
  op initial : -> State .
  vars P Q : Mode .    vars X Y : Nat .
  eq initial = < sleep, 0, sleep, 0 > .
  rl [p1_sleep] : < sleep, X, Q, Y > => < wait, s Y, Q, Y > .
  rl [p1_wait] : < wait, X, Q, 0 > => < crit, X, Q, 0 > .
  crl [p1_wait] : < wait, X, Q, Y > => < crit, X, Q, Y > if not (Y < X) .
  rl [p1_crit] : < crit, X, Q, Y > => < sleep, 0, Q, Y > .
  rl [p2_sleep] : < P, X, sleep, Y > => < P, X, wait, s X > .
  rl [p2_wait] : < P, 0, wait, Y > => < P, 0, crit, Y > .
  crl [p2_wait] : < P, X, wait, Y > => < P, X, crit, Y > if Y < X .
  rl [p2_crit] : < P, X, crit, Y > => < P, X, sleep, 0 > .
endm
```

This specification corresponds to a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, where (Σ, E) imports the equational theory `NAT` of the natural numbers, and where Σ has additional types (called here *sorts*) `Mode` and `State`, with `Mode` consisting of just the constants `sleep`, `wait`, and `crit`. States are represented by terms of sort `State`, which are constructed by a 4-tuple operator `<_,_,_,_>`; the first two components describe the status of the first process (the mode it is currently in, and its priority as given by the number according to which it will be served), and the last two components the status of the second process. E consists of just the equations imported from `NAT`, plus the above equation defining the `initial` state. R consists of eight rewrite rules, four for each process. These rules describe how each process passes from being sleeping to waiting, from waiting to its critical section, and then back to sleeping. In this case, the chosen type k for states is of course `State`.

Rewriting logic then has inference rules to infer all the possible concurrent computations in a system [22], in the sense that, given two states $[u], [v] \in T_{\Sigma/E,k}$, we can reach $[v]$ from $[u]$ by some possibly complex concurrent computation iff we can prove $\mathcal{R} \vdash u \longrightarrow v$ in the logic. In particular we can easily define the *one-step $\mathcal{R}$-rewriting relation*, which is a binary relation $\rightarrow^1_{\mathcal{R},k}$ on $T_{\Sigma,k}$ that holds between terms $u, v \in T_{\Sigma,k}$ iff there is a one-step proof of $\mathcal{R} \vdash u \longrightarrow v$, that is, a proof in which only one rewrite rule in R is applied to a single subterm. We can get a binary relation (with the same name) $\rightarrow^1_{\mathcal{R},k}$ on $T_{\Sigma/E,k}$ by defining $[u] \rightarrow^1_{\mathcal{R},k} [v]$ iff $u' \rightarrow^1_{\mathcal{R},k} v'$ for some $u' \in [u]$, $v' \in [v]$.

The relationship with Kripke structures is now almost obvious, since we can associate to a concurrent system axiomatized by a rewrite theory $\mathcal{R} = (\Sigma, E, R)$

with a chosen type k of states a Kripke structure, $\mathcal{K}(\mathcal{R}, k)_\Pi = (T_{\Sigma/E,k}, (\rightarrow^1_{\mathcal{R},k})^\bullet, L_\Pi)$. We say "almost obvious," because nothing has yet been said about the choice of state predicates Π and the associated labeling function L_Π. The reason for this is methodological: Π, L_Π, and the LTL formulas φ describing properties of the system specified by $\mathcal{R}$ belong to the property specification level. Indeed, for the same system specification $\mathcal{R}$ we may come up with different predicates Π, labeling functions L_Π, and properties φ at the property specification level.

The question of when a rewrite theory $\mathcal{R}$ is executable is closely related with wanting $T_{\Sigma/E,k}$ to be a *computable* set, and $(\rightarrow^1_{\mathcal{R},k})^\bullet$ to be a *computable* relation in the above Kripke structure $\mathcal{K}(\mathcal{R}, k)_\Pi$, an obvious precondition for any model checking. We say that $\mathcal{R} = (\Sigma, E \cup A, R)$ is *executable* if: (1) there exists a *matching algorithm modulo* the equational axioms A[5]; (2) the equational theory $(\Sigma, E \cup A)$ is (ground) *Church-Rosser and terminating modulo* A [27]; and (3) the rules R are (ground) *coherent* [28] relative to the equations E modulo A. Conditions (1–2) ensure that $T_{\Sigma/E\cup A,k}$ is a computable set, since each ground term t can be simplified by applying the equations E from left to right modulo A to reach a *canonical form* $can_{E/A}(t)$ which is unique modulo the axioms A. We can then reduce the equality problem $[u]_{E\cup A} = [v]_{E\cup A}$ to the decidable equality problem $[can_{E/A}(u)]_A = [can_{E/A}(v)]_A$. Condition (3) means that for each ground term t, whenever we have $t \rightarrow^1_{\mathcal{R}} u$ we can always find $can_{E/A}(t) \rightarrow^1_{\mathcal{R}} v$ such that $[can_{E/A}(u)]_A = [can_{E/A}(v)]_A$. This implies that $(\rightarrow^1_{\mathcal{R},k})^\bullet$ is a computable binary relation on $T_{\Sigma/E\cup A,k}$, since we can decide $[t]_{E\cup A} \rightarrow^1_{\mathcal{R}} [u]_{E\cup A}$ by enumerating the finite set of all one-step $\mathcal{R}$-rewrites modulo A of $can_{E/A}(t)$, and for any such rewrite, say v, we can decide $[can_{E/A}(u)]_A = [can_{E/A}(v)]_A$.

4.1 LTL Properties of Rewrite Theories and Model Checking

One appealing feature of rewriting logic is that it provides a seamless integration of the system specification level and the property specification level, because we can specify the relevant state predicates Π *equationally*, and this then determines the labeling function L_Π and the semantics of the LTL formulas φ in a unique way. Indeed, to associate LTL properties to a rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R)$ with a chosen type k of states we only need to make explicit the relevant state predicates Π, which need not be part of the system specification $\mathcal{R}$. The state predicates Π can be defined by means of equations D in an equational theory $(\Sigma', E \cup A \cup D)$ extending $(\Sigma, E \cup A)$ in a conservative way; specifically, the unique Σ-homomorphism $T_{\Sigma/E\cup A} \rightarrow T_{\Sigma'/E\cup A\cup D}$ should be bijective at each sort s in Σ. The syntax defining the state predicates consists of a subsignature $\Pi \subseteq \Sigma'$ of function symbols p of the general form $p : s_1 \ldots s_n \longrightarrow \mathit{Prop}$ (with *Prop* a shorthand for *Proposition*), reflecting the fact that state predicates can be *parametric*. The semantics of the state predicates Π is defined by D with the help of an operator $\_ \models \_ : k\ \ \mathit{Prop} \longrightarrow \mathit{Bool}$ in Σ'. By definition, given ground

[5] In Maude, the axioms A for which the rewrite engine supports matching modulo are any combination of *associativity*, *commutativity*, and *identity* axioms for different binary operators.

terms $u_1, \ldots, u_n$, we say that the state predicate $p(u_1, \ldots, u_n)$ *holds* in the state $[t]$ iff

$$E \cup A \cup D \vdash t \models p(u_1, \ldots, u_n) = true\,.$$

We can now associate to $\mathcal{R}$ a Kripke structure $\mathcal{K}(\mathcal{R}, k)_\Pi$, whose atomic predicates are specified by the set $AP_\Pi = \{\theta(p) \mid p \in \Pi, \theta \text{ ground substitution}\}$[6]. We define $\mathcal{K}(\mathcal{R}, k)_\Pi = (T_{\Sigma/E,k}, (\rightarrow^1_{\mathcal{R},k})^\bullet, L_\Pi)$, where $L_\Pi([t]) = \{\theta(p) \in AP_\Pi \mid \theta(p) \text{ holds in } [t]\}$. In practice we want the equality $t \models p(u_1, \ldots, u_n) = true$ to be *decidable*. This can be achieved by giving equations in $D \cup E$ that are Church-Rosser and terminating modulo A. Then, if we begin with an *executable* rewrite theory $\mathcal{R}$ and define decidable state predicates Π by the method just described, we obtain a *computable* Kripke structure $\mathcal{K}(\mathcal{R}, k)_\Pi$ which, if it has finite reachability sets, can be used for model checking.

The Maude 2.0 system has an on-the-fly, explicit-state LTL model checker [17] which supports the methodology just mentioned. Given an executable rewrite theory specified in Maude by a module `M`, and an initial state `init` of sort `State`$_{\texttt{M}}$, we can model check different LTL properties beginning at this state. For that, a new module `CHECK-M` must be defined importing `M` and the predefined module `MODEL-CHECKER`, and a subsort declaration `State`$_{\texttt{M}}$ `< State` must be added. Then the syntax of the state predicates must be declared by means of operations of sort `Prop`, and their semantics must be given by equations involving the satisfaction operator `op _|=_ : State Prop -> Bool`. Once the semantics of the state predicates has been defined, and assuming that the set of states reachable from `init` is finite, we can model check any LTL `formula` in LTL(AP_Π) by giving to Maude the command: `reduce modelCheck(init, formula)`.

Continuing with our bakery protocol example, two basic properties that we may wish to verify are: (1) *mutual exclusion*: the two processes are never simultaneously in their critical section; and (2) *liveness*: any process in waiting mode will eventually enter its critical section. In order to specify these properties it is enough to specify in Maude the following set Π of state predicates:

```
mod BAKERY-CHECK is inc MODEL-CHECKER . inc BAKERY .
  ops 1wait 2wait 1crit 2crit : -> Prop .
  vars P Q : Mode .    vars X Y : Nat .
  eq (< P, X, Q, Y > |= 1wait) = (P == wait) .
  eq (< P, X, Q, Y > |= 2wait) = (Q == wait) .
  eq (< P, X, Q, Y > |= 1crit) = (P == crit) .
  eq (< P, X, Q, Y > |= 2crit) = (Q == crit) .
endm
```

Since the set of states reachable from `initial` (defined in the `BAKERY` module) is *infinite*, we should not model check the above specification as given. Instead, we should first define an *abstraction* of it where `initial` has only finitely many reachable states and then model check the abstraction.

[6] By convention, if p has n parameters, $\theta(p)$ denotes the term $\theta(p(x_1, \ldots, x_n))$.

5 Equational Abstractions

Let $\mathcal{R} = (\Sigma, E \cup A, R)$ be a rewrite theory. A quite general method for defining abstractions of the Kripke structure $\mathcal{K}(\mathcal{R}, k)_\Pi = (T_{\Sigma/E\cup A,k}, (\rightarrow^1_{\mathcal{R},k})^\bullet, L_\Pi)$ is by specifying an equational theory extension of the form

$$(\Sigma, E \cup A) \subseteq (\Sigma, E \cup A \cup E').$$

Since this defines an equivalence relation $\equiv_{E'}$ on $T_{\Sigma/E\cup A,k}$, namely,

$$[t]_{E\cup A} \equiv_{E'} [t']_{E\cup A} \iff E \cup A \cup E' \vdash t = t' \iff [t]_{E\cup A\cup E'} = [t']_{E\cup A\cup E'},$$

we can obviously define our quotient abstraction as $\mathcal{K}(\mathcal{R}, k)_\Pi/\equiv_{E'}$. We call this the *equational quotient abstraction* of $\mathcal{K}(\mathcal{R}, k)_\Pi$ defined by E'.

But can $\mathcal{K}(\mathcal{R}, k)_\Pi/\equiv_{E'}$, which we have just defined in terms of the underlying Kripke structure $\mathcal{K}(\mathcal{R}, k)_\Pi$, be understood as the Kripke structure associated to *another rewrite theory*? Let us take a closer look at

$$\mathcal{K}(\mathcal{R}, k)_\Pi/\equiv_{E'} = (T_{\Sigma/E\cup A,k}/\equiv_{E'}, (\rightarrow^1_{\mathcal{R},k})^\bullet/\equiv_{E'}, L_{\Pi/\equiv_{E'}}).$$

The first observation is that, by definition, we have $T_{\Sigma/E\cup A,k}/\equiv_{E'} \cong T_{\Sigma/E\cup A\cup E',k}$. A second observation is that if $\mathcal{R}$ is *k-deadlock free*, that is, if we have $(\rightarrow^1_{\mathcal{R},k})^\bullet = \rightarrow^1_{\mathcal{R},k}$, then the rewrite theory $\mathcal{R}/E' = (\Sigma, E \cup A \cup E', R)$ is also k-deadlock free, and we have

$$(\rightarrow^1_{\mathcal{R}/E',k})^\bullet = \rightarrow^1_{\mathcal{R}/E',k} = (\rightarrow^1_{\mathcal{R},k})^\bullet/\equiv_{E'}.$$

Therefore, for $\mathcal{R}$ k-deadlock free, our obvious candidate for a rewrite theory having $\mathcal{K}(\mathcal{R}, k)_\Pi/\equiv_{E'}$ as its underlying Kripke structure is the rewrite theory $\mathcal{R}/E' = (\Sigma, E \cup A \cup E', R)$. That is, we just add to $\mathcal{R}$ the equations E' and do not change at all the rules R. How restrictive is the requirement that $\mathcal{R}$ is k-deadlock free? There is no essential loss of generality: in Sect. 5.2 we show how we can always associate to an executable rewrite theory $\mathcal{R}$ a semantically equivalent (from the LTL point of view) theory $\mathcal{R}_{d.f.}$ which is both deadlock free and executable.

Therefore, at a purely mathematical level, $\mathcal{R}/E'$ seems to be what we want. Assuming that we have an A-matching algorithm, the problem comes with the following two *executability questions* about $\mathcal{R}/E'$, which are essential for $\mathcal{K}(\mathcal{R}, k)_\Pi/\equiv_{E'}$ to be computable, and therefore for model checking:

- Are the equations $E \cup E'$ ground Church-Rosser and terminating modulo A?
- Are the rules R ground coherent relative to $E \cup E'$ modulo A?

The answer to each of these questions may be positive or negative. In practice, sufficient care on the part of the user when specifying E' should result in an affirmative answer to the first question. In any case, we can always try to check such a property with a tool such as Maude's Church-Rosser checker [19]; if the check fails, we can try to complete the equations with a Knuth-Bendix completion tool, for example [29, 30], to get a theory $(\Sigma, E'' \cup A)$ equivalent to

$(\Sigma, E \cup A \cup E')$ for which the first question has an affirmative answer. Likewise, we can try to check whether the rules R are ground coherent relative to $E \cup E'$ (or to E'') modulo A using the tools described in [20]. If the check fails we can again try to complete the rules R to a semantically equivalent set of rules R', using also those tools [20]. By this process we can hopefully arrive at an *executable* rewrite theory $\mathcal{R}' = (\Sigma, E'' \cup A, R')$ which is semantically equivalent to $\mathcal{R}/E'$. We can then use $\mathcal{R}'$ to try to model check properties about $\mathcal{R}$.

But we are not finished yet. What about the state predicates Π? Recall (see the end of Sect. 4) that these (possibly parameterized) state predicates will have been defined by means of equations D in a Maude module importing the specification of $\mathcal{R}$ and also the `MODEL-CHECKER` module. The question is whether the state predicates Π are *preserved* under the equations E'. This indeed may be a problem. We need to unpack a little the definition of the innocent-looking labeling function $L_{\Pi/\equiv_{E'}}$, which is defined by the intersection formula

$$L_{\Pi/\equiv_{E'}}([t]_{E\cup A\cup E'}) = \bigcap_{[x]_{E\cup A}\subseteq[t]_{E\cup A\cup E'}} L_{\Pi}([x]_{E\cup A}).$$

In general, computing such an intersection and coming up with new equational definitions D' capturing the new labeling function $L_{\Pi/\equiv_{E'}}$ may not be easy. It becomes much easier if the state predicates Π are *preserved* under the equations E'. By definition, we say that the state predicates Π are preserved under the equations E' if for any $[t]_{E\cup A}, [t']_{E\cup A} \in T_{\Sigma/E\cup A,k}$ we have the implication

$$[t]_{E\cup A\cup E'} = [t']_{E\cup A\cup E'} \quad \Longrightarrow \quad L_{\Pi}([t]_{E\cup A}) = L_{\Pi}([t']_{E\cup A}).$$

Note that in this case, assuming that the equations $E \cup E' \cup D$ (or $E'' \cup D$) are ground Church-Rosser and terminating modulo A, we *do not need to change the equations* D to define the state predicates Π on $\mathcal{R}/E'$ (or its semantically equivalent $\mathcal{R}'$). Therefore, we have an isomorphism (given by a pair of invertible bisimulation maps)

$$\mathcal{K}(\mathcal{R}, k)_{\Pi}/{\equiv_{E'}} \cong \mathcal{K}(\mathcal{R}/E', k)_{\Pi},$$

or, in case we need the semantically equivalent $\mathcal{R}'$, an isomorphism

$$\mathcal{K}(\mathcal{R}, k)_{\Pi}/{\equiv_{E'}} \cong \mathcal{K}(\mathcal{R}', k)_{\Pi}.$$

The crucial point in both isomorphisms is that the labeling function of the righthand side Kripke structure is now equationally defined by the same equations D as before. Since by construction either $\mathcal{R}/E'$ or $\mathcal{R}'$ are executable theories, for an initial state $[t]_{E\cup A\cup E'}$ having a *finite* set of reachable states we can use the Maude model checker to model check any LTL formula in this equational quotient abstraction. Furthermore, since the quotient AP_{Π}-simulation map

$$\mathcal{K}(\mathcal{R}, k)_{\Pi} \longrightarrow \mathcal{K}(\mathcal{R}/E', k)_{\Pi}$$

is then by construction *strict*, it reflects satisfaction of *arbitrary* LTL formulas by Thm. 1.

A practical problem remains: how can we actually try to *prove* the implication

$$[t]_{E\cup A\cup E'} = [t']_{E\cup A\cup E'} \quad\Longrightarrow\quad L_\Pi([t]_{E\cup A}) = L_\Pi([t']_{E\cup A})$$

to show the desired preservation of state predicates? A particularly easy case is that of *k-topmost* rewrite theories, that is, theories in which the type k of states only appears as the codomain of an operation $f : k_1 \dots k_n \longrightarrow k$. This is not a very restrictive condition, since any rewrite theory $\mathcal{R}$ can be transformed into a semantically equivalent k'-topmost one just by encapsulating the original type of states k in a new type k' through an operation $\{\_\} : k \longrightarrow k'$ [21].

Proposition 3. *Suppose a k-topmost rewrite theory in which all (possibly conditional) equations in E' are of the form $t = t'$ if C with $t, t' \in T_{\Sigma,k}$, and that the equations $E\cup E'\cup D$ are Church-Rosser and terminating modulo A. Furthermore, suppose that no equations between terms in $T_{\Sigma,k}$ appear in the conditions of any equation in E'. If for each equation $t = t'$ if C in E' and each state predicate p we can prove the inductive property*

$$E \cup A \cup D \vdash_{ind} (\forall \vec{x}\, \forall \vec{y})\; C \rightarrow (t(\vec{x}) \models p(\vec{y}) = true \;\leftrightarrow\; t'(\vec{x}) \models p(\vec{y}) = true)$$

then we have established the preservation of the state predicates Π by the equations E'.

We can use a tool like Maude's ITP [18] to mechanically discharge proof obligations of this kind.

5.1 Case Studies

The Bakery Protocol Example Revisited. We can use the bakery protocol example to illustrate how equational quotient abstractions can be used to verify infinite-state systems. We can define such an abstraction by adding to the equations of BAKERY-CHECK a set E' of additional equations defining a quotient of the set of states. We can do so in the following module extending BAKERY-CHECK by equations and leaving the transition rewrite rules unchanged:

```
mod ABSTRACT-BAKERY-CHECK is inc BAKERY-CHECK .
  vars P Q : Mode .   vars X Y : Nat .
  eq < P, 0, Q, s s Y > = < P, 0, Q, s 0 > .
  eq < P, s s X, Q, 0 > = < P, s 0, Q, 0 > .
  ceq < P, s X, Q, s Y > = < P, s s 0, Q, s 0 >
      if (Y < X) /\ not(Y == 0 and X == s 0) .
  ceq < P, s X, Q, s Y > = < P, s 0, Q, s 0 >
      if not (Y < X) /\ not (Y == 0 and X == 0) .
endm
```

Note that $\langle P, N, Q, M\rangle \equiv \langle P', N', Q', M'\rangle$ according to the above equations iff (1) $P = P'$ and $Q = Q'$, (2) $N = 0$ iff $N' = 0$, (3) $M = 0$ iff $M' = 0$, (4) $M < N$ iff $M' < N'$. Three key questions are: (1) Is the set of states now finite?; (2)

Does this abstraction correspond to a rewrite theory whose equations are ground Church-Rosser and terminating?; (3) Are the rules still ground coherent?

The equations are indeed ground Church-Rosser and terminating. It is also clear that the set of states is now finite, since in the canonical forms obtained with these equations the natural numbers possible in the state can never be greater than $s(s(0))$. This leaves us with the ground coherence question. We have to analyze possible "relative critical pairs" between rules and equations. For example, consider the following pair of a rule and an equation:

```
rl [p1_sleep] : < sleep, X, Q, Y > => < wait, s Y, Q, Y > .
eq < P, 0, Q, s s Y > = < P, 0, Q, s 0 > .
```

The only possible overlap corresponds to the unification (after making the variables disjoint) of the two lefthand sides yielding the term `< sleep, 0, Q, s s Y >`, which is rewritten by the rule to `< wait, s s s Y, Q, s s Y >` and by the equation to `< sleep, 0, Q, s 0 >`, with both terms being finally reduced to `< wait, s s 0, Q, s 0 >`, in the first case by means of the third equation, and in the second one by the rule `[p1_sleep]`. All the other rule-equation pairs can likewise be proved coherent. A fourth pending question is the deadlock freedom of `BAKERY-CHECK`. This property holds and can be checked with the ITP.

What about state predicates? Are they preserved by the abstraction? Note that, since the rewrite theory is `State`-topmost and the equations are all between terms of sort `State`, according to Prop. 3 we only need to check that each of the equations preserves the above state predicates. But this is trivial, since the predicates only depend on `Mode` components that are left unchanged by the equations. This can be mechanically checked using Maude's ITP [21].

In other words, we have just shown that, for Π the state predicates declared in `BAKERY-CHECK`, we have a strict quotient simulation map,

$$\mathcal{K}(\texttt{BAKERY-CHECK}, \texttt{State})_{\Pi} \longrightarrow \mathcal{K}(\texttt{ABSTRACT-BAKERY-CHECK}, \texttt{State})_{\Pi}.$$

Therefore, we can establish the mutual exclusion property of `BAKERY-CHECK` by model checking in `ABSTRACT-BAKERY-CHECK` the following:

```
reduce modelCheck(initial, []~ (1crit /\ 2crit)) .
result Bool: true
```

Likewise, we can establish the liveness property of `BAKERY-CHECK` by model checking in `ABSTRACT-BAKERY-CHECK`:

```
reduce modelCheck(initial, (1wait |-> 1crit) /\ (2wait |-> 2crit)) .
result Bool: true
```

Other Examples. In addition to the bakery protocol we have also dealt successfully with a number of examples that have been used in the literature to illustrate other abstraction methods, including a readers/writers system [11], the alternating bit protocol [13, 6, 12], a mutual exclusion protocol discussed in

[7], and the bounded retransmission protocol [1, 2, 6]. The abstractions were obtained simply by adding some equations to the specifications. Only in the last two cases was it necessary to add some extra (but semantically equivalent) rules to guarantee coherence; the details can be found in [21].

5.2 The Deadlock Difficulty

The reason why we have focused on deadlock-free rewrite theories is because deadlocks can pose a problem due to a subtle point in the semantics of LTL. As emphasized in its definition, the transition relation of a Kripke structure is *total*, and this requirement is also imposed on the Kripke structures arising from rewrite theories. Consider then the following specification of a rewrite theory, together with the declaration of two state predicates:

```
mod FOO is inc MODEL-CHECKER .
  ops a b c : -> State .  ops p1 p2 : -> Prop .
  eq (a |= p1) = true .   eq (b |= p2) = true .   eq (c |= p1) = true .
  rl a => b .             rl b => c .
endm
```

The transition relation of the Kripke structure corresponding to this specification has three elements: $\mathtt{a} \to \mathtt{b}$, $\mathtt{b} \to \mathtt{c}$, and $\mathtt{c} \to \mathtt{c}$, the last one consistently added by the model checker according to the semantics given to LTL. Suppose now that we wanted to abstract this system and that we decided to identify `a` and `c` by means of a simulation map h. For that, according to the previous sections, it would be enough to add the equation `eq c = a` to the above specification. The resulting system is coherent, and `a` and `c` satisfy the same state predicates. Note that the resulting Kripke structure has only two elements in its transition relation: one from the equivalence class of `a` to that of `b`, and another in the opposite direction. Now, since no deadlock can occur in any of the states, the model checker does not add any additional transition steps. In particular, there is no transition from the equivalence class of `a` to itself, but that means that the resulting specification does *not* correspond to the minimal system associated to h in which such a transition does exist. The lack of this idle transition is a serious problem, because now we can prove properties about the simulating system that are actually false in the original one, for example, $\Box \Diamond \mathtt{p2}$.

One simple way to deal with this difficulty is to just add idle transitions for each of the states in the resulting specification by means of a rule of the form `x => x`. The resulting system, in addition to all the rules that the minimal system should contain, may in fact have some extra "junk" rules that are not part of it. Therefore, we end up with a system that can be soundly used to infer properties of the original system (it is immediate to see that we have a simulation map) but that in general will be coarser than the minimal system.

A better way of addressing the problem is to characterize the set of deadlock states. For this, given a rewrite theory $\mathcal{R}$ we can define a new operation *enabled* : $k \longrightarrow$ *Bool?* for each type k in $\mathcal{R}$, where *Bool?* is a supersort of *Bool*. Then we add, for each rule $t \to t'$ *if* C, the equation *enabled*$(t) =$ *true if* C and, for each operation $f : k_1 \dots k_n \longrightarrow k$, n equations of the form *enabled*$(f(x_1, \dots, x_n)) =$ *true if enabled*$(x_i) =$ *true*, so that $(\exists t')\ t \to^1_{\mathcal{R},k} t'$ iff *enabled*$(t) =$ *true*. This *enabled* predicate is the key point in the proof of the following proposition, which allows us to transform an executable rewrite theory into a semantically equivalent one that is both deadlock-free and executable.

Proposition 4. *Let $\mathcal{R} = (\Sigma, E \cup A, R)$ be an executable rewrite theory. Given a chosen type of states k, we can construct an executable theory extension $\mathcal{R} \subseteq \mathcal{R}^k_{d.f.} = (\Sigma', E' \cup A, R')$ such that:*

- *$\mathcal{R}^k_{d.f.}$ is k'-deadlock free and k'-topmost for a certain type k';*
- *there is a function $h : T_{\Sigma',k'} \longrightarrow T_{\Sigma,k}$ inducing a bijection $h : T_{\Sigma'/E' \cup A,k'} \longrightarrow T_{\Sigma/E \cup A,k}$ such that for each $t, t' \in T_{\Sigma',k'}$ we have*

$$h(t)(\to^1_{\mathcal{R},k})^\bullet h(t') \iff t \to^1_{\mathcal{R}^k_{d.f.},k'} t'.$$

Furthermore, if Π are state predicates for $\mathcal{R}$ and k defined by equations D, then we can define state predicates Π for $\mathcal{R}^k_{d.f.}$ and k' by equations D' such that the above map h becomes a bijective AP_Π-bisimulation

$$h : \mathcal{K}(\mathcal{R}^k_{d.f.}, k')_\Pi \longrightarrow \mathcal{K}(\mathcal{R}, k)_\Pi\,.$$

6 Related Work and Conclusions

In [3] the simulation of a system $\mathcal{M}$ by another $\mathcal{M}'$ through a surjective function h was defined and the optimal simulation $\mathcal{M}^h_{\min}$ was identified. The idea of simulating by quotient has been further explored in [4, 5, 2, 10, 12, 7] among others, although the construction in [7] requires a Galois connection instead of just a function. Theorem proving is proposed in [2] to construct the transition relation of the abstract system, and in [12] to prove that a function is a representative function that can be used as input to an algorithm to extract $\mathcal{M}^h_{\min}$ out of $\mathcal{M}$. While those uses of theorem proving focus on the correctness of the abstract transition relation, our method focuses on making the minimal transition relation (which is correct by construction) *computable*, and on proving the preservation of the labeling function. In [3, 4], on the other hand, the minimal model $\mathcal{M}^h_{\min}$ is discarded in favor of less precise but easier to compute approximations; this would correspond, in our approach, to the addition of rewrite rules to the specification to simplify the proofs of the proof obligations. In all the papers mentioned two states can become identified only if they satisfy the same atomic propositions; our definition of simulation is more general, but we have not yet exploited this.

The equational abstraction method that we have presented seems to apply in practice to a good number of examples discussed in the literature. But we need to further test its applicability on a wider and more challenging range of examples. Also, the method itself should be generalized along several directions.

For example, we should generalize the equational theory extension $(\Sigma, E \cup A) \subseteq (\Sigma, E \cup A \cup E')$ to an arbitrary *theory interpretation* $H : (\Sigma, E \cup A) \longrightarrow (\Sigma', E'')$, allowing arbitrary transformations on the data representation of states. A particular instance of this seems to be *predicate abstraction* [14, 6]. Under this approach, the abstract domain is a Boolean algebra over a set of assertions and the abstraction function, typically as part of a Galois connection, is symbolically constructed as the conjunction of all expressions satisfying a certain condition, which is proved using theorem proving. This would correspond to a theory interpretation $H : (\Sigma, E) \longrightarrow (\Sigma \cup \Sigma', E \cup E')$, with Σ' introducing operations of the form $p : State \longrightarrow Bool$, and with H mapping states S to Boolean tuples $\langle p_1(S), \ldots, p_n(S)\rangle$. Similarly, we should consider simulation maps between *different* sets AP and AP' of state predicates, yielding another increase in generality when relating systems. Finally, equational abstractions that do not require strict preservation of state predicates should also be investigated.

Acknowledgments. Research supported by ONR Grant N00014-02-1-0715, NSF Grant CCR-0234524, and by DARPA through Air Force Research Laboratory Contract F30602-02-C-0130; and by the Spanish project AMEVA CICYT TIC 2000–0701–C02–01. We warmly thank Saddek Bensalem, Yassine Lakhnech, David Basin, Felix Klaedtke, Natarajan Shankar, Hassen Saidi, and Tomás Uribe for many useful discussions that have influenced the ideas presented here, Manuel Clavel and Francisco Durán for their help in the preparation of this paper, and Roberto Bruni and Joe Hendrix for many useful comments on previous drafts.

References

1. Abdulla, P., Annichini, A., Bouajjani, A.: Symbolic verification of lossy channel systems: Application to the bounded retransmission protocol. In Cleaveland, W.R., ed.: Tools and Algorithms for the Construction of Analysis of Systems, TACAS'99. LNCS 1579., Springer (1999)
2. Bensalem, S., Lakhnech, Y., Owre, S.: Computing abstractions of infinite state systems compositionally and automatically. In Hu, A.J., Vardi, M.Y., eds.: Computer Aided Verification, CAV'98. LNCS 1427., Springer (1998) 319–331
3. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Transactions on Programming Languages and Systems **16** (1994) 1512–1542
4. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
5. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In Emerson, E.A., Sistla, A.P., eds.: Computer Aided Verification, CAV'00. LNCS 1855., Springer (2000) 154–169
6. Colón, M.A., Uribe, T.E.: Generating finite-state abstractions of reactive systems using decision procedures. In Hu, A.J., Vardi, M.Y., eds.: Computer Aided Verification, CAV'98. LNCS 1427., Springer (1998) 293–304
7. Dams, D., Gerth, R., Grumberg, O.: Abstract interpretation of reactive systems. ACM Transactions on Programming Languages and Systems **19** (1997) 253–291
8. Havelund, K., Shankar, N.: Experiments in theorem proving and model checking for protocol verification. In Gaudel, M.C., Woodcock, J., eds.: FME '96: Industrial Benefit and Advances in Formal Methods. LNCS 1051., Springer (1996) 662–681

9. Kesten, Y., Pnueli, A.: Control and data abstraction: The cornerstones of practical formal verification. International Journal on Software Tools for Technology Transfer **4** (2000) 328–342
10. Kesten, Y., Pnueli, A.: Verification by augmentary finitary abstraction. Information and Computation **163** (2000) 203–243
11. Loiseaux, C., Graf, S., Sifakis, J., Bouajjani, A., Bensalem, S.: Property preserving abstractions for the verification of concurrent systems. Formal Methods in System Design **6** (1995) 1–36
12. Manolios, P.: Mechanical Verification of Reactive Systems. PhD thesis, Univ. of Texas at Austin (2001)
13. Müller, O., Nipkow, T.: Combining model checking and deduction for I/O-automata. In Brinksma, E., et al., eds.: Tools and Algorithms for the Construction and Analysis of Systems, TACAS '95. LNCS 1019., Springer (1995) 1–16
14. Saïdi, H., Shankar, N.: Abstract and model check while you prove. In Halbwachs, N., Peled, D., eds.: Computer Aided Verification, CAV'99. LNCS 1633., Springer (1999) 443–454
15. Uribe Restrepo, T.E.: Abstraction-Based Deductive-Algorithmic Verification of Reactive Systems. PhD thesis, Dept. of Computer Science, Stanford Univ. (1998)
16. Clavel, M., Durán, F., Ecker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: Specification and programming in rewriting logic. Theoretical Computer Science **285** (2002) 187–243
17. Eker, S., Meseguer, J., Sridharanarayanan, A.: The Maude LTL model checker. In Gadducci, F., Montanari, U., eds.: Rewriting Logic and its Applications, WRLA 2004. ENTCS 71., Elsevier (2002)
18. Clavel, M.: The ITP tool. In Nepomuceno, A., et al., eds.: Logic, Language, and Information, Kronos (2001) 55–62
19. Durán, F., Meseguer, J.: A Church-Rosser checker tool for Maude equational specifications. `http://maude.cs.uiuc.edu/tools` (2000)
20. Durán, F.: Coherence checker and completion tools for Maude specifications. `http://maude.cs.uiuc.edu/tools` (2000)
21. Meseguer, J., Palomino, M., Martí-Oliet, N.: Notes on model checking and abstraction in rewriting logic. `http://formal.cs.uiuc.edu/texts/nmcarl.ps` (2002)
22. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science **96** (1992) 73–155
23. Meseguer, J.: Membership algebra as a logical framework for equational specification. In Parisi-Presicce, F., ed.: Recent Trends in Algebraic Development Techniques WADT'97. LNCS 1376., Springer (1998) 18–61
24. Borovanský, P., Kirchner, C., Kirchner, H., Moreau, P.E.: ELAN from a rewriting logic point of view. Theoretical Computer Science **285** (2002) 155–185
25. Futatsugi, K., Diaconescu, R.: CafeOBJ Report. World Scientific (1998)
26. Lamport, L.: A new solution of Dijkstra's concurrent programming problem. Communications of the ACM **17** (1974) 453–455
27. Dershowitz, N., Jouannaud, J.P.: Rewrite systems. In van Leeuwen, J., ed.: Handbook of Theoretical Computer Science, Vol. B. North-Holland (1990) 243–320
28. Viry, P.: Equational rules for rewriting logic. Theoretical Computer Science **285** (2002)
29. Contejean, E., Marché, C.: The CiME system: tutorial and user's manual. Manuscript, Univ. Paris-Sud, Centre d'Orsay
30. Durán, F.: Termination checker and Knuth-Bendix completion tools for Maude equational specifications. Manuscript, Computer Science Laboratory, SRI International, `http://maude.cs.uiuc.edu/papers` (2000)