Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 290 (2012) 19–36

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Rewriting-Based Model Checker for the Linear Temporal Logic of Rewriting

Kyungmin Bae[1] José Meseguer[2]

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, USA

Abstract

This paper presents a model checker for $LTLR$, a subset of the temporal logic of rewriting TLR^* extending linear temporal logic with *spatial action patterns*. Both $LTLR$ and TLR^* are very expressive logics generalizing well-known state-based and action-based logics. Furthermore, the semantics of TLR^* is given in terms of rewrite theories, so that the concurrent systems on which the $LTLR$ properties are model checked can be specified at a very high level with rewrite rules. This paper answers a nontrivial challenge, namely, to be able to build a model checker to model check $LTLR$ formulas on rewrite theories with relatively little effort by reusing Maude's LTL model checker for rewrite theories. For this, the reflective features of both rewriting logic and its Maude implementation have proved extremely useful.

Keywords: temporal logic of rewriting, model checking, rewriting logic, reflective transformation

1 Introduction

In temporal logic and model checking one can distinguish two main camps: a *state-based* camp, in which all atoms in formulas are *state predicates* (e.g., LTL, CTL, and CTL^* [11]); and an *event-based* camp, where the formulas' atoms are *actions* or *events* (e.g., Hennesy-Milner's logic [22], or De Nicola and Vaadrager's A-CTL^* logic [35]). At the semantic level, state based formulas are evaluated on Kripke structures. Instead, action-based formulas are evaluated on labeled transition systems.

Some properties can be naturally expressed in state-based logics and are difficult to express in action-based logics, whereas the opposite is the case for other properties. This means that, when the property does not fit well a given logic, one has to "cook" in a possibly complex way both the system description (as a Kripke structure or a label transitions system depending on the logic's semantics) and the

[1] Email: kbae4@cs.uiuc.edu

[2] Email: meseguer@cs.uiuc.edu

1571-0661 © 2012 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2012.11.009

property in order to model check it in the given logic. The situation is even more challenging for *mixed* properties such as fairness properties (see the discussion [31]), where both state-based predicates and actions are involved.

Reflecting on this situation we can speak, as in [31], of *tandems*, each given by a pair $\mathcal{L}_S/\mathcal{L}_P$, where $\mathcal{L}_S$ is a formalism to specify *systems* (for example, the formalisms of Kripke structures or of label transition systems), and $\mathcal{L}_P$ is a formalism to describe *properties* (for example, some state-based or action-based temporal logic). The frequent need for "cooking" both the system and the property in both state-based and action-based tandems is then due to a lack of expressiveness in both cases.

1.1 An Example

To illustrate the lack of expressiveness, of either solely state-based or solely action-based systems, we use a variant of a simple parallel language, whose rewriting semantics is presented in [15,18], in which we can define Dekker's algorithm for mutual exclusion and then model check some of its properties. The parallel language supports processes that execute concurrently on a shared memory machine and communicate with each other through shared variables.

Dekker's algorithm has two processes with entirely symmetric code. Process 1 sets a Boolean variable `c1` to 1 to indicate that it wishes to enter its critical section. Process 2 does the same with variable `c2`. If one process, after setting its variable to 1 finds that the variable of its competitor is 0, then it enters its critical section right away. In case of a tie (both variable set to 1) the tie is broken using a variable `turn` that takes values in {1,2}. For example, the code of process 1 is a follows:

```
repeat
  'c1 := 1 ;
  while 'c2 = 1 do
    if 'turn = 2 then
      'c1 := 0 ;
      while 'turn = 2 do skip od ;
      'c1 := 1
    fi
  od ;
  crit ; 'turn := 2 ; 'c1 := 0 ; rem
forever
```

where fragments of code for the critical section and for the remaining part of the program are respectively abstracted as constants `crit` and `rem`[3].

Global states are modeled as pairs, with first component a set of processes, and second component a shared memory. In their Maude rewriting semantics such global states are instances of the pattern `{[I,R] | S, M}`, where `[I,R]` is one of the processes with `I` its process id and `R` its program code to be executed next by process, `|` is an associative-commutative parallel process composition operator, `S` is the remaining set of processes, and `M` is the shared memory. The language's operational semantics is then defined by rewriting rules for each language feature.

[3] We assume that `crit` is *terminating*, but `rem` may not be. See [15] for a "cooked" version of the example, and [1] for the Maude specification of both the example and is *LTLR* properties.

A property to be checked (in fact it fails, see Section 5) for Dekker's algorithm is the *strong fairness property* that executing infinitely often implies entering one's critical section infinitely often, expressed as the LTL-*like* formula

$$\Box\Diamond\ exec.p1 \Rightarrow \Box\Diamond\ in\text{-}crit.p1.$$

The predicate $in\text{-}crit.p1$ means that process 1 is in its critical section and can be defined easily by the equation

$$\texttt{\{[p1, crit ; R] | S, M\}} \models in\text{-}crit.p1 = true.$$

However, the above strong fairness formula is only "LTL-like" and not really an LTL formula, since $exec.p1$, which asserts the execution of process 1, *cannot be defined directly*, because there is no way to know which process has executed some statement using the current state description. To express $exec.p1$ we need to "cook" both the state representation and the operational semantics by adding a *third* component to the state, indicating the last executed process, and modifying the semantic rules to update such third component. After this "cooking," we can define $exec.p1$ as a state predicate by the equation

$$\texttt{\{[I, R] | S, M, p1\}} \models exec.p1 = true.$$

This is just a convoluted way to represent what is really an *action* ($exec.p1$) indirectly as a *state predicate*. Likewise, there are also natural properties in state-based system that would need a convoluted "cooking" to be represented as actions. For example, a predicate which is true when a process is in its `rem` part, say $in\text{-}rem.p1$, is nontrivial to define in an action-based system, while it is trivial to define in a state-based system by an equation similar to that for $in\text{-}crit.p1$.

In this paper we describe a new tandem *RewritingLogic*/*TLR**, first proposed in [31], where the need for cooking both the system and the property disappears. For example, the above strong fairness formula for Dekker can be expressed *directly* in *TLR**. We also present a Maude-based model checker for the *LTLR* sublogic of *TLR**, which extends with actions the *LTL* logic. The point is that rewriting logic [30] is more expressive than both Kripke structures and labeled transitions systems, since in a rewrite theory state predicates can be equationally specified, and rewrite rules are labeled. The logic *TLR**, called the *temporal logic of rewriting*, extends *CTL** with *spatial action patterns*, which are quite expressive, since they can *localize* a rewrite rule's action to a given context and a partial substitution.

The nontrivial challenge answered in this paper is to be able to build a model checker for the *RewritingLogic*/*LTLR* tandem with relatively little effort by reusing Maude's *LTL* model checker for rewrite theories. For this, the reflective features of both rewriting logic and its Maude implementation [15] have proved extremely useful. In essence, the new model checker design uses a result in [31] by which the model checking of a *LTLR* formula on a rewrite theory $\mathcal{R}$ can be reduced to the model checking of a translated *LTL* formula on a translated Kripke structure. Using reflection, this is here achieved by a reflective theory transformation associating each rewrite theory $\mathcal{R}$ with a new theory such that the given *LTLR* formula holds for $\mathcal{R}$ and a given initial state *if and only if* its *LTL* translation holds for the new theory.

The paper is organized as follows. Section 2 presents the necessary background and Section 3 provides the formal foundations, defining and proving correctness of the reflective construction. Section 4 explains the reflective design of a *LTLR* model checker based on the foundations. Section 5 illustrates the use of the model checker with an example; and Section 6 discusses related work and presents conclusions.

2 Rewriting Logic and the Temporal Logic of Rewriting

This section explains the concepts of *RewritingLogic* and TLR^*, constituting the *RewritingLogic*/TLR^* tandem, which is the semantic infrastructure of the model checker described in this paper.

2.1 Rewriting Logic

A rewrite theory is a formal specification of a concurrent system with static states and concurrent transitions between the states. More precisely, a *rewrite theory* is a triple $\mathcal{R} = (\Sigma, E, R)$ [4] such that:

- (Σ, E) is a many-sorted equational theory. [5] The initial algebra $T_{\Sigma/E}$ defined by the equational theory (Σ, E) defines the states of the system specified by $\mathcal{R}$.
- R is a collection of rewrite rules of the form $l : q \longrightarrow r$, with l a label (which can be duplicated for several rules), q and r Σ-terms of the same sort, and such that the set of variables $vars(r)$ is a subset of the variables $vars(q)$. These rewrite rules define concurrent transitions between states.

More precisely, each state is modeled as an E-equivalence class $[t]_E$ of ground terms, and rewriting happens *modulo* E; that is, rewriting E-equivalence classes $[t]_E$ representing states, not just terms t. A *one-step rewrite* $[t]_E \longrightarrow_{\mathcal{R}} [t']_E$ exists in $\mathcal{R}$ iff there exists $u \in [t]_E$ such that u can be rewritten to v using some rule $l : q \longrightarrow r$ in R in the standard way, [6] denoted $u \longrightarrow_R v$, and we furthermore have $v \in [t']_E$.

The most useful rewrite theories satisfy additional executability conditions, because for arbitrary E and R, whether $[t]_E \longrightarrow^*_{\mathcal{R}} [t']_E$ is *undecidable* in general. A rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R)$ is *computable* if the following conditions hold:

(i) Equality modulo A is decidable, and there exists a *matching algorithm modulo* A, producing a finite number of A-matching substitutions or failing otherwise, that can implement rewriting in A-equivalence classes.

(ii) $(\Sigma, E \cup A)$ is *ground terminating and confluent modulo* A [16]. That is: (i) there are no infinite sequences of rewritings with E module A; and (ii) for each $[t]_A \in T_{\Sigma/A}$ there is a *unique* A-equivalence class $[can_{E/A}(t)]_A \in T_{\Sigma/A}$ called

[4] This definition can be extended to the more general rewrite theories in [3,15], which uses a more expressive equational logic, *conditional* rewrite rules, and *frozen* function operators.

[5] There are various possibilities for the equational theory (Σ, E) such as unsorted, many-sorted, order-sorted, or even membership equational logic. However, to keep the exposition as simple as possible, we will assume that (Σ, E) is a many-sorted equational theory.

[6] See [16] for basic notation on term rewriting. Positions in a term are denoted as strings of nonzero natural numbers and represent tree positions when the term is parsed as a tree. Two useful notions are that of a subterm of a given term t at a given position p, denoted $t|_p$, and of replacement in t of such a subterm by another term u at position p, denoted $t[u]_p$. For example, in the term $t = x + ((z + 0) + y)$, the subterm at position 2.1 is $z + 0$, and the replacement $t[z]_{2.1}$ is the term $x + (z + y)$.

the *E-canonical form* of $[t]_A$ modulo A such that the last term, which cannot be further rewritten with E module A, of any terminating sequence beginning at $[t]_A$ is necessarily $[can_{E/A}(t)]_A$.

(iii) The rules R are *ground coherent* relative to the equations E modulo A [37]. That is, if $[t]_A$ is rewritten to $[t']_A$ by a rule l in R, $[can_{E/A}(t)]_A$ is also rewritten to $[t'']_A$ by the *same* rule l such that $[can_{E/A}(t')]_A = [can_{E/A}(t'')]_A$

In addition, to make the integration of rewriting logic and TLR^* smoother, we define the class $RWTh_0$ of rewrite theories as follows.

Definition 2.1 $\mathcal{R} \in RWTh_0$ when the following condition is satisfied:

- $\mathcal{R}$ is computable and has a sort *State* as its chosen sort of states.
- If $\mathcal{R}$ has a sort named *Prop*, then it must also have a sort named *Bool* with constants *true* and *false*, and an operator $\_ \models \_ : State \times Prop \rightarrow Bool$.[7]
- The subsignature $\Pi \subseteq \Sigma$ of its *state predicate symbols* is the set of all operators in Σ of the form $p : A_1 \times \cdots \times A_n \rightarrow Prop$, $n \geq 0$. $A_1, \ldots, A_n$ are called the *parameter sorts* of the atomic state predicate p.
- $\mathcal{R}$ is *deadlock-free*, that is, there are no finite sequences
$$[t_1]_A \longrightarrow_{\mathcal{R}} [t_2]_A \ldots [t_n]_A \longrightarrow_{\mathcal{R}} [t_{n+1}]_A$$
such that $[t_{n+1}]_A$ cannot be further rewritten (i.e., it is a "deadlock state"). This is not at all a strong restriction, since, as explained in [15,33], any rewrite theory $\mathcal{R}$ whose rules do not have rewrites in their conditions can be transformed into a semantically equivalent theory $\widehat{\mathcal{R}}$ that is deadlock-free.

Proof Terms and Computations

The inference rules of rewriting logic derive all concurrent computations in the system specified by $\mathcal{R}$ [3,30]. That is, given two states $[u], [v] \in T_{\Sigma/E\cup A}$, one can *reach* $[v]$ from $[u]$ by some possibly complex concurrent computation if and only if one can *prove* $\mathcal{R} \vdash [u] \longrightarrow^+ [v]$[8]. In rewriting logic any such complex computation reaching $[v]$ from $[u]$ is witnessed by a *proof term* [30,31], say λ, written $\mathcal{R} \vdash \lambda : [u] \longrightarrow^+ [v]$.

Proof terms are identified modulo natural equations making any proof term λ always equivalent to an interleaving description as a sequential composition $\gamma_1; \ldots; \gamma_k$ of one-step proof terms γ_i [3,30,31], which have a very simple algebraic description.

Definition 2.2 Given rewrite proof $R \vdash \gamma : [u] \longrightarrow_{\mathcal{R}} [v]$ using a rewrite rule $l : q \longrightarrow r \in R$, a *one-step* proof term γ has the form $t[l(\phi)]_p$, where $t \in [u]$, p is a position in t where the rule is applied, and $\phi = \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$ is a substitution such that $t|_p = \phi(q)$, $t[\phi(r)]_p \in [v]$, where $x_1, \ldots, x_n$ are variables in q.

In the above definition, if t is an E/A-canonical term, say $t = can_{E/A}(u)$, we say that $can_{E/A}(u)[l(\phi)]_p$ is a *canonical* one-step proof term [31]. Canonical one-step

[7] *Prop* is the designated sort of atomic state predicates, and $\models$ is the function defining whether a given state satisfies a given state predicate.

[8] $[u] \longrightarrow^+ [v]$ denotes a combination of one or more sequential compositions of concurrent rewrites.

rewrite proofs are the key ingredient to arrive at our desired notion of *computation*, on which the truth of TLR^* path formulas will be evaluated. Before defining computation, let us define two useful sets of canonical proof terms. First, the set $(Can_{\Sigma/E,A})_{State}$ of all A-equivalence classes of the form $[can_{E/A}(t)]_A$, where t is a ground Σ-term of sort $State$. Second, we can define the set $CanPTerms^1(\mathcal{R})$ of all one-step canonical proof terms in $\mathcal{R}$. In a computable rewrite theory, any proof term is always semantically equivalent to a canonical one [31].

Definition 2.3 An *infinite computation* in $\mathcal{R} \in RWTh_0$ is a pair of functions (π, γ), with $\pi : \mathbb{N} \longrightarrow (Can_{\Sigma/E,A})_{State}$ and $\gamma : \mathbb{N} \longrightarrow CanPTerms^1(\mathcal{R})$ such that for all $n \in \mathbb{N}$, $\pi(n) \xrightarrow{\gamma(n)} \pi(n+1)$ is a canonical one-step rewrite proof in $\mathcal{R}$. Graphically,

$$\pi(0) \xrightarrow{\gamma(0)} \pi(1) \xrightarrow{\gamma(1)} \pi(2) \ldots \pi(n) \xrightarrow{\gamma(n)} \pi(n+1) \ldots$$

$Comp(\mathcal{R})^\infty$ denotes the set of infinite computations in $\mathcal{R}$, and for each $[t] \in (Can_{\Sigma/E,A})_{State}$, $Comp(\mathcal{R})^\infty_{[t]}$ denotes the infinite computations starting at $[t]$, that is, those computations (π, γ) such that $\pi(0) = [t]$. Given an infinite computation (π, γ) and a number $i \in \mathbb{N}$, $(\pi, \gamma)^i$ denotes the suffix of (π, γ) beginning at position i, that is, the pair of functions $(\pi \circ s^i, \gamma \circ s^i)$ with s the successor function, s^0 the identity function, and $s^{n+1} = s \circ s^n$.

Spatial Actions

Spatial actions are the action atoms of TLR^*. They generalize one-step proof terms, which can be thought of as *ground-instantiated spatial actions*. Spatial actions describe *patterns*, which in general specify not just a single one-step proof term, but a possibly infinite set of such proof terms. Roughly speaking, we can think of spatial actions as "one-step proof terms with variables," but they are slightly more general than that as we explain below.

Let Ω be the subsignature of constructors[9] and L be the set of labels labeling rules in R, and assume that $\Omega \cap L = \emptyset$. The signature $\Omega(L)$ extends Ω by adding:

- fresh sorts Top and $Subst$
- an associative and commutative operator $\_;\_ : Subst\ Subst \to Subst$.
- for each rewrite rule $l : q \longrightarrow r$ in R with q, r of sort B, and with variables $x_1, \ldots, x_n$ in q having sorts $B_1, \ldots, B_n$:
  · operators $l : Subst \to B$ and $x_1\backslash\_ : B_1 \to Subst, \ldots, x_n\backslash\_ : B_n \to Subst$
  · a constant l of sort B and an operator $top : B \to Top$,

Let X be a many-sorted set of variables with an infinite set of variables for each sort in Ω. Consider the algebras: (i) $T_{\Omega(L)/A}(X)$ of A-equivalence classes of $\Omega(L)$-terms with variables in X; and (ii) $T_{\Omega/A}(X)$ of A-equivalence classes of Ω-terms with variables in X. Also, assume that the substitution ϕ has the form $x_1\backslash u_1; \cdots; x_n\backslash u_n$.

[9] $\Omega \subseteq \Sigma$ is the subsignature of *constructors* associated with the ground confluent and terminating (modulo A) theory $(\Sigma, E \cup A)$, where $f \in \Omega$ iff there is a ground term t s.t. f is a function symbol in $[can_{E/A}(t)]$

Definition 2.4 $\mathcal{R}$'s spatial action patterns $SP(\Omega, L) \subset T_{\Omega(L)/A}(X)$ are defined by:

- for each $l \in L$, $[l]_A, [top(l)]_A \in SP(\Omega, L)$
- $[l(\phi)]_A \in SP(\Omega, L)$ if $l \in L$, $[l(\phi)]_A \in T_{\Omega(L)/A}(X)$, $u_1, \ldots, u_n \in T_{\Omega/A}(X)$
- $[top(l(\phi))]_A \in SP(\Omega, L)$ if $l \in L$, $[top(l(\phi))]_A \in T_{\Omega(L)/A}(X)$, $u_1, \ldots, u_n \in T_{\Omega/A}(X)$
- $[v[l]_p]_A \in SP(\Omega, L)$ if p is not the empty (top) position, $l \in L$, $[v[l]_p]_A \in T_{\Omega(L)/A}(X)$, and $v \in T_{\Omega/A}(X)$.
- $[v[l(\phi)]_p]_A \in SP(\Omega, L)$ if p is not the empty (top) position, $l \in L$, $[v[l(\phi)]_p]_A \in T_{\Omega(L)/A}(X)$, and $v, u_1, \ldots, u_n \in T_{\Omega/A}(X)$.

$SP(\Omega, L)$ defines $\Omega(L)$-terms that are spatial action patterns. Note that $CanPTerms^1(\mathcal{R}) \subseteq SP(\Omega, L)$, so that any canonical one-step proof term is a ground version of some spatial action pattern.

An action pattern of the form l describes a rule labeled l that can be applied *anywhere*. An action pattern $l(\phi)$ allows l to also be applied anywhere, but constrains the variable instantiation related to rule l to be itself a further instance of ϕ. Action patterns of the form $top(l(\phi))$ are needed to cover the case where l is applied at the top of the term. The most fully spatial patterns are those of the form $v[l(\phi)]_p$ with v a nonempty context and p a position. The *instance-of* relation between a spatial action pattern and a proof term captures these meanings of spatial action patterns. Let $[u]_A \preceq_A [v]_A$ iff there is a many-sorted substitution θ such that $[u]_A = [\theta(v)]_A$ [10]. In addition, for substitution terms, let us define $[\phi]_A \preceq_A [\varphi]_A$ iff for all $[x_i \backslash v_i]_A \in [\varphi]_A$, there exists $[x_i \backslash u_i]_A \in [\phi]_A$ such that $[u_i]_A \preceq_A [v_i]_A$.

Definition 2.5 The *instance-of* relation between a canonical one-step proof term γ and a spatial action pattern $\delta \in SP(\Omega, L)$, denoted $\gamma \sqsubseteq_A \delta$, is a slight variant of the $\preceq_A$ relation defined as follows:

- $[v[l(\phi)]_p]_A \sqsubseteq_A [l]_A$
- $[v[l(\phi)]_p]_A \sqsubseteq_A [l(\varphi)]_A$ iff $[\phi]_A \preceq_A [\varphi]_A$
- $[v[l(\phi)]_p]_A \sqsubseteq_A [w[l]_{p'}]_A$ iff $[v[l(\_)]_p]_A \preceq_A [w[l(\_)]_{p'}]_A$
- $[v[l(\phi)]_p]_A \sqsubseteq_A [w[l(\varphi)]_{p'}]_A$ iff $[v[l(\_)]_p]_A \preceq_A [w[l(\_)]_{p'}]_A$ and $[\phi]_A \preceq_A [\varphi]_A$
- $[l(\phi)]_A \sqsubseteq_A [top(l)]_A$
- $[l(\phi)]_A \sqsubseteq_A [top(l(\varphi))]_A$ iff $[\phi]_A \preceq_A [\varphi]_A$.

Reflection and Metalevel Computation

Rewriting logic is reflective in a precise mathematical way [12], namely, there is a finitely presented rewrite theory $\mathcal{U}$ that is *universal* in the sense that we can represent in $\mathcal{U}$ any finitely presented rewrite theory $\mathcal{R}$ (including $\mathcal{U}$ itself) as a term $\overline{\mathcal{R}}$, any terms t, t' in $\mathcal{R}$ as terms $\overline{t}, \overline{t'}$, and any pair $(\mathcal{R}, t)$ as a term $\langle \overline{\mathcal{R}}, \overline{t} \rangle$, in such a way that we have the following equivalence

$$\mathcal{R} \vdash t \longrightarrow^* t' \ \Leftrightarrow\ \mathcal{U} \vdash \langle \overline{\mathcal{R}}, \overline{t} \rangle \longrightarrow^* \langle \overline{\mathcal{R}}, \overline{t'} \rangle$$

[10] This is a *decidable* relation by our assumption that there is an A-matching algorithm

Since $\mathcal{U}$ is representable in itself, we can achieve a "reflective tower" with an arbitrary number of levels of reflection [12,13]:

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

Key functionality of the universal theory $\mathcal{U}$ can be controlled by *descent function*[15,14]. For example, in Maude, given $\mathcal{R} = (\Sigma, E \cup A, R)$, several key constructs of $\mathcal{U}$ are defined by the following descent functions:

- $\texttt{metaReduce}(\overline{\mathcal{R}}, \overline{t})$ metarepresents the E/A-canonical form of term t in $\mathcal{R}$. If $\mathcal{R}$ is computable, then $[t]_{E/A} = [t']_{E/A}$ iff $\texttt{metaReduce}(\overline{\mathcal{R}}, \overline{t}) = \texttt{metaReduce}(\overline{\mathcal{R}}, \overline{t'})$.
- $\texttt{metaMatch}(\overline{\mathcal{R}}, \overline{t}, \overline{t'})$ metarepresents the instance-of modulo A relation between terms: $t' \preceq_A t$ iff $\texttt{metaMatch}(\overline{\mathcal{R}}, \overline{t}, \overline{t'}) = \mathit{true}$.
- $\texttt{metaXapply}(\overline{\mathcal{R}}, \overline{t}, l, m)$ [11] metarepresents the m-th one-step rewrite of term t by rule l in $\mathcal{R}$. $\mathcal{R} \vdash [v[l(\phi)]_p]_A : t \rightarrow t'$ iff $\exists m$ s.t. $\texttt{metaXapply}(\overline{\mathcal{R}}, \overline{t}, l, m) = (\overline{t'}, \overline{v}[]_p, \overline{\phi})$.

2.2 The Linear Temporal Logic of Rewriting

TLR* is a family of logics parameterized by the spatial actions $\mathit{SP}(\Omega, L)$ and the signature of atomic propositions Π. The most general of these logics is *TLR**, a generalization of the state-based *CTL** logic that allows both spatial actions and state predicates in formulas. It contains various important sublogics of interest, which appear as special cases (see [31]). For our model checking purposes here, where we are interested in extending with support for spatial actions the Maude*LTL* model checker for rewrite theories described in [15], we focus on *LTLR*, the sublogic generalizing *LTL*, which is parameterized as $\mathit{LTLR}(\mathit{SP}(\Omega, L), \Pi)$ by the spatial actions $\mathit{SP}(\Omega, L)$ and the signature of state predicates Π. The following is the syntax for *LTLR* in BNF-like style, where, instead of using *CTL**-like notation, we adopt the implicitly universally path quantified *LTL* notation [12]:

$$\delta : \mathit{SP}(\Omega, L),\; p : \mathit{Prop}(\Pi),\; \varphi, \varphi' : \mathit{LTLR}(\mathit{SP}(\Omega, L), \Pi)$$

$$\mathit{LTLR}(\mathit{SP}(\Omega, L), \Pi) : \; \delta \mid p \mid \neg\varphi \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi' \mid \bigcirc\varphi \mid \varphi \mathcal{U} \varphi' \mid \varphi \mathcal{R} \varphi' \mid \varphi \mathcal{W} \varphi' \mid \Diamond\varphi \mid \Box\varphi$$

Smaller and useful sublogics of $\mathit{TLR}^*(\mathit{SP}(\Omega, L), \Pi)$ can be obtained by restricting the atomic propositions and/or spatial actions allowed. That is, we can define sublogics of *TLR** parameterized by a subset $W \subseteq \mathit{SP}(\Omega, L)$ of spatial actions, and a subset $\Delta \subseteq \mathit{Prop}(\Pi)$ of atomic propositions. Specifically, the sublogic $\mathit{LTLR}(W, \Delta) \subseteq \mathit{LTLR}(\mathit{SP}(\Omega, L), \Pi)$ is defined by the set-theoretic formula $\mathit{LTLR}(W, \Delta) = \{\varphi \in \mathit{LTLR}(\mathit{SP}(\Omega, L), \Pi) \mid \mathit{sp}(\varphi) \subseteq W \;\wedge\; \mathit{prop}(\varphi) \subseteq \Delta\}$, where $\mathit{sp}(\varphi)$ denotes the set of spatial action subformulas of φ, and $\mathit{prop}(\varphi)$ denotes the set of atomic proposition subformulas. In the particular, when $W = \emptyset$, and $\Delta = \mathit{Prop}(\Pi)$, we obtain specializations to the *state-based logics LTL*; that is,

[11] This definition of `metaXapply` describes only part of the actual function. See [15] for the full definition.

[12] We assume that all state predicate constants and function symbols are constructors, i.e., that there is a subsignature containment $\Pi \subseteq \Omega$, and then define the set $\mathit{Prop}(\Pi)$ of *atomic propositions* as the set of ground terms $\mathit{Prop}(\Pi) = T_{\Omega_{\mathit{Prop}}}$.

$\mathit{LTL}(\Pi) = \mathit{LTLR}(\emptyset, \Pi)$. For other specializations of this kind to various state-based and action-based logics see [31].

The semantics of formula $\varphi \in \mathit{LTLR}(\mathit{SP}(\Omega, L), \Pi)$ is given by the satisfaction relation $\mathcal{R}, [t] \models \varphi$, where $\mathcal{R} \in \mathit{RWTh}_0$ has subsignatures of constructors Ω and of state predicates Π, and $[t]$ is a state (i.e., an A-equivalence class $[t]_A$ in E-canonical form modulo A and of sort *State*, where $E \cup A$ are the equations in $\mathcal{R}$). By definition, $\mathcal{R}, [t] \models \varphi$ holds if and only if for each infinite computation $(\pi, \gamma) \in \mathit{Comp}(\mathcal{R})^{\infty}_{[t]}$ the path satisfaction relation $\mathcal{R}, (\pi, \gamma) \models \varphi$ holds. Since one can express all of $\mathit{LTLR}(\mathit{SP}(\Omega, L), \Pi)$ in terms of $\mathit{SP}(\Omega, L)$, $\mathit{Prop}(\Pi)$, and the basic connectives $\top$, $\neg$, $\vee$, $\bigcirc$, and $\mathcal{U}$, it is enough to define the semantics for the atoms and for those connectives. Since *LTLR* generalizes *LTL*, the semantic definitions are entirely similar to those for *LTL* (see, e.g., [27]). The key new addition is the semantics of *spatial actions*; the relation $\mathcal{R}, (\pi, \gamma) \models \delta$ holds if and only if the proof term $\gamma(0)$ of a current computation is an instance of a spatial action pattern δ.

The path satisfaction relation is defined inductively as follows:

- $\mathcal{R}, (\pi, \gamma) \models \top$
- $\mathcal{R}, (\pi, \gamma) \models p \;\Leftrightarrow\; \mathit{can}_{E/A}(\pi(0) \models p) = \mathit{true}$
- $\mathcal{R}, (\pi, \gamma) \models \delta \;\Leftrightarrow\; \gamma(0) \sqsubseteq_A \delta$
- $\mathcal{R}, (\pi, \gamma) \models \neg\varphi \;\Leftrightarrow\; \mathcal{R}, (\pi, \gamma) \not\models \varphi$
- $\mathcal{R}, (\pi, \gamma) \models \varphi \vee \varphi' \;\Leftrightarrow\; \mathcal{R}, (\pi, \gamma) \models \varphi$ or $\mathcal{R}, (\pi, \gamma) \models \varphi'$
- $\mathcal{R}, (\pi, \gamma) \models \bigcirc\varphi \;\Leftrightarrow\; \mathcal{R}, (\pi, \gamma)^1 \models \varphi$
- $\mathcal{R}, (\pi, \gamma) \models \varphi \mathcal{U} \varphi' \Leftrightarrow \exists k \in \mathbb{N}\, s.t.\, \mathcal{R}, (\pi, \gamma)^k \models \varphi' \wedge \forall 0 \leq i < k \;\; \mathcal{R}, (\pi, \gamma)^i \models \varphi$

At the syntactic level, we have seen that *LTLR* contains *LTL*. Similar containments exist for pure action logics [31]. This generality has a counterpart at the semantic level: both Kripke structures and labeled transition systems can be seen as very simple special cases of rewrite theories [31].

3 Reflective Reduction to State-Based Temporal Logics

To make possible the use of standard CTL^* (or *LTL*) model checkers to verify TLR^* (or *LTLR*) properties of finite-state systems specified by rewrite theories, we need to show that there is a pair of model and formula transformations faithfully mapping the tandem $\mathit{RewritingLogic}/\mathit{TLR}^*$ to the tandem $\mathit{Kripke}/\mathit{CTL}^*$. After briefly describing how such a mapping is mathematically defined in terms of Kripke structures in Section 3.1 (summarizing results in [31,32]), we give in Section 3.2 a new construction of the same mapping at the level of rewrite theories, as a reflective rewrite theory transformation and prove the correctness of this new construction. Section 3.2 provides the theoretical foundations for the reflective design and implementation of the Maude *LTLR* model checker described in Section 4.

3.1 Reduction to State-Based Temporal Logics

The faithful mapping of tandems we seek is a mapping parametric in W

$$(\mathcal{K}_W, \widetilde{(\_)}) : \mathit{RewritingLogic}/\mathit{TLR}^*(W, \Pi) \longrightarrow \mathit{Kripke}/\mathit{CTL}^*(\Pi \cup W)$$

with W a finite set of spatial actions[13] in the given rewrite theory $\mathcal{R}$, and Π the subsignature of state predicates in $\mathcal{R}$.

Definition 3.1 Given $\mathcal{R} \in \mathit{RWTh}_0$ and a finite set $W \subseteq \mathit{SP}(\Omega, L)$, the construction $\mathcal{K}_W$ maps the rewrite theory $\mathcal{R}$ to the following Kripke structure[14] $\mathcal{K}_W(\mathcal{R})$:

(i) Its set of states is $(\mathit{Can}_{\Sigma/E,A})_{\mathit{State}} \times \mathcal{P}(W)$

(ii) Its transition relation is defined by the equivalence: $([t], U) \longrightarrow ([t'], V)$ iff there is a canonical one-step rewrite proof $[t] \xrightarrow{\gamma} [t']$ in $\mathcal{R}$, and V is the set
$$\mathit{act}_W(\gamma) = \{\delta \in W \mid \gamma \sqsubseteq_A \delta\}.$$

(iii) Its set of atomic propositions is the set $\mathit{Prop}(\Pi) \cup W$, and the labeling function maps a state $([t], U)$ to the set of atomic propositions $\mathcal{L}_{\mathcal{R}}([t]) \cup U$, where, by definition, $\mathcal{L}_{\mathcal{R}}([t]) = \{p \in \mathit{Prop}(\Pi) \mid \mathit{can}_{E/A}([t] \models p) = \mathit{true}\}$.

By the above definition, $([t], U) \models \delta$ if and only if a spatial action pattern $\delta \in U$, where $[t]$ is a state of $\mathcal{R}$ and $U \subseteq W$. Since the condition (ii) asserts that U is the set of *all* spatial action patterns of which the one-step proof term of a current computation is an instance, this coincides with the semantics of $\mathcal{R}$, $(\pi, \gamma) \models \delta$ defined in Section 2.2.

Definition 3.2 Given a formula $\varphi \in \mathit{TLR}^*(W, \Pi)$ we can map it to the formula $\widetilde{\varphi} \in \mathit{CTL}^*(W \cup \Pi)$ by systematically replacing each occurrence of a spatial action $\delta \in W$ in φ by the formula $\mathbf{X}\delta$.

The construction $\mathcal{K}_W(\mathcal{R})$, with the above formula translation $\widetilde{(\_)}$, defines a mapping of tandems $(\mathcal{K}_W, \widetilde{(\_)}) : \mathit{RewritingLogic}/\mathit{TLR}^*(W, \Pi) \longrightarrow \mathit{Kripke}/\mathit{CTL}^*(\Pi \cup W)$. This mapping is a *faithful* mapping of tandems preserving the satisfaction relations $\models$ in $\mathit{TLR}^*(W, \Pi)$ and $\models_{\mathit{CTL}^*}$ in $\mathit{CTL}^*(\Pi \cup W)$. This is shown by the following theorem proved in detail in [31], where a complexity-theoretic analysis of the reduction is also given.

Theorem 3.3 *Given a rewrite theory $\mathcal{R} \in \mathit{RWTh}_0$ and a finite $W \subseteq \mathit{SP}(\Omega, L)$, for each state $[t]$ in $\mathcal{R}$, $U \subseteq W$, and $\varphi \in \mathit{TLR}^*(W, \Pi)$, the following equivalence holds:*

$$\mathcal{R}, [t] \models \varphi \Leftrightarrow \mathcal{K}_W(\mathcal{R}), ([t], U) \models_{\mathit{CTL}^*} \widetilde{\varphi}$$

3.2 The $\mathcal{K}_W$ Construction as a Reflective Theory Transformation

The $\mathcal{K}_W(\mathcal{R})$ construction maps each rewrite theory in RWTh_0 to a Kripke structure. However, as pointed out in [31], it is very useful to decompose the mapping $\mathcal{R} \mapsto$

[13] Since any TLR^* formula ξ only involves a finite set $\mathit{sp}(\xi)$ of spatial actions as subformulas, it is always sufficient to consider formulas in $\mathit{TLR}^*(W, \Pi)$ with W *finite*.

[14] Recall that a *Kripke structure* on a set AP of atomic propositions is a triple $\mathcal{K} = (A, R, \mathcal{L})$, with A a set of states, $R \subseteq A \times A$ a total transition relation, and $\mathcal{L} : A \longrightarrow \mathcal{P}(AP)$ a labeling function assigning to each state $a \in A$ the set $\mathcal{L}(a) \subseteq AP$ of the atomic propositions that hold in a.

$\mathcal{K}_W(\mathcal{R})$ into a rewrite theory transformation $\mathcal{R} \mapsto \mathcal{R}_W$ followed by the general construction $\mathcal{R} \mapsto \mathcal{K}(\mathcal{R})$, spelled out in detail in [15,18], which maps each $RWTh_0$ to its underlying Kripke structure. That is, we can decompose the $\mathcal{K}_W(\mathcal{R})$ construction as $\mathcal{K}_W(\mathcal{R}) = \mathcal{K}(\mathcal{R}_W)$. This is particularly useful for tool building purposes, since the construction $\mathcal{R} \mapsto \mathcal{K}(\mathcal{R})$ is already automated in Maude's LTL model checker [15]. So, "essentially" all we need to do is to automate the theory transformation $\mathcal{R} \mapsto \mathcal{R}_W$. Here is where the reflective properties of rewriting logic summarized in Section 2.1 and efficiently supported in Maude by the `META-LEVEL` module become extremely useful. In what follows we describe in detail and prove correct a reflective, parametric construction for the $\mathcal{R} \mapsto \mathcal{R}_W$ transformation in which the parameter $\mathcal{R}$ is metarepresented as a term $\overline{\mathcal{R}}$ in the universal rewrite theory $\mathcal{U}$. We can then obtain the $\mathcal{R} \mapsto \mathcal{R}_W$ construction as a reflective parametric rewrite theory $\mathcal{U}_{\_}(\_)$ which, when instantiated with parameters $\overline{W}$ and $\overline{\mathcal{R}}$, yields a theory $\mathcal{U}_W(\mathcal{R})$ that extends $\mathcal{U}$ and provides a correct realization of the theory $\mathcal{R}_W$. We explain in detail the parametric $\mathcal{U}_W(\mathcal{R})$ construction in what follows.

Note that, in particular, the $\mathcal{U}_W(\mathcal{R})$ construction has to metarepresent the instance-of relation $\sqsubseteq_A$, between a one-step proof term and a spatial action pattern. In the `META-LEVEL` built-in implementation of $\mathcal{U}$'s sorts and descent functions, this can be achieved using the `metaMatch` descent function, which meta-represents the instance-of relation between terms. Since `metaMatch` is parametric on each rewrite theory, its metarepresentation is also parametric on the given rewrite theory. Let $\overline{\sqsubseteq}_{\overline{\mathcal{R}}}$ denote the metarepresentation of the instance-of relation between the meta representations of a one-step proof term and a spatial action pattern in the equational theory $(\Omega(L), A)$ associated with the rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R)$ in Section 2.1. Then, for a one-step proof term γ and a spatial action pattern δ, we have $\gamma \sqsubseteq_A \delta$ iff $\overline{\gamma} \; \overline{\sqsubseteq}_{\overline{(\Omega(L),A)}} \; \overline{\delta}$. The $\mathcal{U}_W(\mathcal{R})$ construction contains the universal theory $\mathcal{U}$ and can then be defined as follows.

Definition 3.4 The rewrite theory $\mathcal{U}_W(\mathcal{R})$ is the following parametric extension of the universal theory $\mathcal{U}$, in which $\mathcal{R} = (\Sigma, E \cup A, R)$ and $W \subset SP(\Omega, L)$ are metarepresented as *data parameters* $\overline{\mathcal{R}}$ and $\overline{W}$:

- $\mathcal{U} \subset \mathcal{U}_W(\mathcal{R})$, where $\mathcal{U}_W(\mathcal{R})$ includes all the descent functions in the `META-LEVEL` module as well as a descent function for the instance-of relation $\overline{\gamma} \; \overline{\sqsubseteq}_{\overline{(\Omega(L),A)}} \; \overline{\delta}$.
- $\mathcal{U}_W(\mathcal{R})$ has sorts *State*, *Prop*, and *Bool* with constants *true* and *false*, where the ground terms of sort *State* are pairs $(\overline{t}, \overline{U})$, with t a term of sort *State* in $\mathcal{R}$, and $U \subseteq W$ (that is, $\overline{U}$ is a meta-term that uses a set union associative and commutative operator to represent a finite set of action patterns contained in W).
- $\mathcal{U}_W(\mathcal{R})$ has also an $\overline{act}$ operator with $\overline{act}(\overline{W}, \overline{\gamma}) = \{\overline{\delta} \in \overline{W} \mid \overline{\gamma} \; \overline{\sqsubseteq}_{\overline{\mathcal{R}}} \; \overline{\delta}\}$.
- There is a single conditional rewrite rule *tr* in $\mathcal{U}_W(\mathcal{R})$ such that $tr : (\overline{t}, \overline{U}) \longrightarrow (\texttt{metaReduce}(\overline{\mathcal{R}}, \overline{t'}), \overline{V})$ iff there exists a rule label l in $\mathcal{R}$ and a natural number m such that $\texttt{metaXapply}(\overline{\mathcal{R}}, \overline{t}, l, m) = (\overline{t'}, \overline{v[]_p}, \overline{\phi}) \wedge \overline{V} = \overline{act}(\overline{W}, \overline{v[l(\phi)]_p})$. The rule's condition can be equationally expressed using "matching conditions" with extra variables (see [15]). Note that in particular this means that we have a one-step rewrite proof term $\mathcal{R} \vdash v[l(\phi)]_p : t \longrightarrow t'$.

- The ground terms of sort *Prop* are precisely the metarepresentations of: (i) either the atomic propositions of $\mathcal{R}$, or (ii) the elements of W.
- There is a labeling operator $\_ \models \_ : State \times Prop \to Bool$ with conditional equations such that:
  - $(\overline{t}, \overline{U}) \models \overline{\delta} = \text{true}$ iff $\delta \in U$
  - $(\overline{t}, \overline{U}) \models \overline{p} = \text{true}$ iff `metaReduce`$(\overline{\mathcal{R}}, \overline{t \models p}) = \overline{true}$, i.e., $can_{E/A}(t \models p) = true$.

The theory $\mathcal{U}_W(\mathcal{R})$ defined above is meta-level description for $\mathcal{K}_W(\mathcal{R})$, and so the definition is closely related to Definition 3.1. The correctness of the $\mathcal{U}_W(\mathcal{R})$ construction is expressed by the following proposition.

Proposition 3.5 *Given a rewrite theory $\mathcal{R} \in RWTh_0$ and a finite set $W \subseteq SP(\Omega, L)$, for each state $[t]$ in $\mathcal{R}$ and $U \subseteq W$, the following conditions hold:*

(i) *states $([t], U)$ of $\mathcal{K}_W(\mathcal{R})$ are in one-to-one correspondence with ground terms of sort State of the form $(\overline{can_{E/A}(t)}, \overline{U})$ in $\mathcal{U}_W(\mathcal{R})$.*

(ii) *For each atomic proposition α of $\mathcal{K}_W(\mathcal{R})$, $([t], U) \models \alpha$ iff $(\overline{can_{E/A}(t)}, \overline{U}) \models \overline{\alpha} = true$ in $\mathcal{U}_W(\mathcal{R})$.*

(iii) *There is a transition $([t], U) \longrightarrow^* ([t'], V)$ in $\mathcal{K}_W(\mathcal{R})$ iff $\mathcal{U}_W(\mathcal{R}) \vdash (\overline{can_{E/A}(t)}, \overline{U}) \longrightarrow (\overline{can_{E/A}(t')}, \overline{V})$.*

Proof (Sketch)

(i) By construction, the terms of sort *State* in the reflective theory $\mathcal{U}_W(\mathcal{R})$ extending $\mathcal{U}$ are pairs $(\overline{can_{E/A}(t)}, \overline{U})$ metarepresenting states $([t], U)$ of $\mathcal{K}_W(\mathcal{R})$. This metarepresentation is one-to-one because of the computability assumptions about $\mathcal{R}$ (which include confluence and termination of E modulo A).

(ii) There are two cases when $([t], U) \models \alpha$ is true in $\mathcal{K}_W(\mathcal{R})$; $\alpha \in U$, or $can_{E/A}([t] \models \alpha) = true$. Since $\alpha \in U$ iff $\overline{\alpha} \in \overline{U}$ and $can_{E/A}([t] \models \alpha) = true$ iff `metaReduce`$(\overline{\mathcal{R}}, \overline{can_{E/A}(t) \models \alpha}) = \overline{true}$, in both cases, by the above definition, $([t], U) \models \alpha$ iff $(\overline{can_{E/A}(t)}, \overline{U}) \models \overline{\alpha}$.

(iii) If a transition $([t], U) \longrightarrow ([t'], V)$ exists in $\mathcal{K}_W(\mathcal{R})$, then, by definition, there exists a one-step rewrite $[t] \xrightarrow{\gamma} [t']$ in $\mathcal{R}$ and $V = act_W(U)$. But, by the definition of $\mathcal{U}_W(\mathcal{R})$, this is equivalent to the existence of a one-step rewrite $tr : (\overline{t}, \overline{U}) \longrightarrow$ (`metaReduce`$(\overline{\mathcal{R}}, \overline{t'}), \overline{V})$ which can happen if and only if there exists a rule label l in $\mathcal{R}$ and a natural number m such that `metaXapply`$(\overline{\mathcal{R}}, \overline{t}, l, m) = (\overline{t'}, \overline{v[]_p}, \overline{\phi})$ and $\overline{V} = act(\overline{W}, \overline{v[l(\phi)]_p})$.

□

In addition to the $\mathcal{U}_W(\mathcal{R})$ construction, we need to deal with the metarepresentation of the *LTL* formulas $\widetilde{\varphi}$ associated to *LTLR* formulas.

Definition 3.6 Given a formula φ, $\widetilde{\widetilde{\varphi}}$ is the same as $\widetilde{\varphi}$, except that each atomic proposition (resp. a spatial action pattern) is replaced by its meta-representation.

4 Reflective Design of an LTLR Model Checker

Maude supports rewriting based LTL model checking for any computable rewrite theory $\mathcal{R}$ [15,18], that is, Maude automates the $\mathcal{R} \mapsto \mathcal{K}(\mathcal{R})$ construction and provides an *LTL* model checker for the underlying Kripke structure $\mathcal{K}(\mathcal{R})$. Therefore, the reduction method described in Section 3.1 to reduce model checking of *LTLR* formulas to that of *LTL* formulas can be used, thanks to the reflective $\mathcal{U}_W(\mathcal{R})$ construction described in Section 3.2, to reduce the model checking of $\mathcal{R}$ with respect to a formula $\varphi \in LTLR(W, \Pi)$ to performing *LTL* model checking on the rewrite theory $\mathcal{U}_W(\mathcal{R})$ with respect to the formula $\widetilde{\overline{\varphi}}$.

In a practical model checking system implementation, however, we want to hide all metalevel representations from the user; therefore the reflective $\mathcal{U}_W(\mathcal{R})$, while being the core of the model checker, is not what the user directly interacts with. In particular, we should support model checking commands in which the *LTLR* formula and the initial state are specified *at the object level*. This requires the design of a suitable user interface in addition to the internal reflective commands that perform the actual model checking. All this can be achieved, as we explain in this section, by extending the Full Maude language [17] using reflective methods.

4.1 Constructing the LTLR Syntax with Spatial Action Patterns

In the *LTLR* syntax, the atomic propositions and the spatial action patterns in formulas are not fixed: they depend parametrically on the given computable rewrite theory $\mathcal{R}$. In particular, spatial action patterns can be uniquely characterized as soon as $\mathcal{R}$ is specified. The idea is that given $\mathcal{R} = (\Sigma, E, R)$ with a subsignature of constructor Ω and rule labels L, we can define a map

$$\texttt{TLR} : \mathcal{R} \longrightarrow \mathcal{R} \cup SP(\Omega, L),$$

and this map can be metarepresented as an equationally defined function

$$\overline{\texttt{TLR}} : \overline{\mathcal{R}} \longrightarrow \overline{\mathcal{R}} \cup \overline{SP(\Omega, L)}$$

in an extension of the `META-LEVEL` module. The function $\overline{\texttt{TLR}}$ can be integrated into Full Maude [17], and by extending Full Maude's parser the user can then specify at the object level *LTLR* properties using the parametric *LTLR* syntax in $\texttt{TLR}(\mathcal{R})$. The remaining thing we need to do is to define $\overline{SP(\Omega, L)}$ as a rewrite theory. It is not hard, since we can just use the definition of $\Omega(L)$ and Definition 2.4. The slightly tricky part is to make actions $v[l(\phi)]_p$ include only one basic action term $l(\phi)$. It is archived by adding the operators

$$\begin{aligned}
o : \mathit{Action}\text{-}A_1 \times A_2 \times A_3 \times \cdots \times A_n &\to \mathit{Action}\text{-}A \\
o : A_1 \times \mathit{Action}\text{-}A_2 \times A_3 \times \cdots \times A_n &\to \mathit{Action}\text{-}A \\
\vdots \qquad\qquad\qquad & \;\;\vdots \qquad \vdots \\
o : A_1 \times A_2 \times \cdots \times A_{n-1} \times \mathit{Action}\text{-}A_n &\to \mathit{Action}\text{-}A
\end{aligned}$$

for each operator $o : A_1 \times \cdots \times A_n \to A \in \Omega$ with $n \geq 1$, where $\mathit{Action}\text{-}A_i$ is a related action sort for each sort A_i. Since to yield a spatial action pattern, operators in Ω can have only one action sub-pattern, the uniqueness of a basic action term $l(\phi)$ as a sub-term is automatically guaranteed.

4.2 Reflective LTLR Model Checker Design

Given a computable rewrite theory $\mathcal{R} \in RWTh_0$, we are now able to define *LTLR* properties using $\mathtt{TLR}(\mathcal{R}) \cup \Pi$, where Π is the set of atomic propositions in $\mathcal{R}$. The next step is to transform the *LTLR* properties into *LTL* properties. This is accomplished by the map $\overline{\varphi} \mapsto \widetilde{\overline{\varphi}}$ associated to Definition 3.6, which can be constructed by reflection using the metarepresented theory $\overline{\mathtt{TLR}}(\overline{\mathcal{R}}) \cup \overline{\Pi}$. Then, when a user gives to the *LTLR* Model Checker system extending Full Maude the command

(tlr check(t, φ) .)

the system performs the followings tasks: (i) t and φ are parsed and metarepresented as $\bar{t}$ and $\overline{\varphi}$; (ii) the set $\overline{W} = \overline{sp(\varphi)}$ is computed, and the translation $\overline{\varphi} \mapsto \widetilde{\overline{\varphi}}$ is performed; (iii) the Maude *LTL* model checker is invoked at the metalevel on the theory $\mathcal{U}_W(\mathcal{R})$ with initial state $(\bar{t}, \overline{\emptyset})$ and *LTL* formula $\widetilde{\overline{\varphi}}$; and (iv) the output of the model checking results (either `true` or a counterexample) are provided to the user at the object level using Full Maude's meta-pretty printing features. By the results in Section 3, we are then guaranteed (assuming correctness of the Maude and Full Maude implementations and of the reflective model checker implementation) that the model checker system will answer `true` to a model checking command (tlr check(t, φ) .) if and only if $\mathcal{R}, t \models \varphi$.

5 The Example Revisited

This section illustrates the use of the above *LTLR* model checker with Dekker's algorithm example explained in Section 1.1[15]. A similar definition of this parallel language and the algorithm appeared in [15,18], but since only *LTL* model checking was available in [15,18], there this required manual "cooking" both the rewrite theory and the formulas, as explained in Subsection 1.1. Instead, here everything can be specified and model checked in the most natural way. The code of algorithm and the system description are the same as before. The global states are of the form `{[I, R] | S, M}`, with `[I,R]` a process with id `I` and code `R`, `S` the set of remaining processes, and `M` the memory. The rewrite rules defining the language's operational semantics are all labeled with the same label `stmt`. Hence, the spatial action pattern `stmt(I\ p1)` asserts the execution of process `p1`. The predicates *in-crit* and *in-rem*, discussed in Section 1.1, are equationally defined as follows.

```
(mod DEKKER-CHECK is protecting TLR[DEKKER] .
    subsort MachineState < State .
    ops in-crit in-rem : Pid -> Prop .
    op exec : Pid -> Action .

    var M : Memory . vars R : Program . var S : Soup . vars J : Pid .

    eq exec(J) = [stmt(I\ J)] .
    eq {[J, crit ; R] | S, M} |= in-crit(J) = true .
```

[15] For a collection of other examples and the code of the *LTLR* model checker see the web page [1].

```
    eq {[J, R] | S, M} |= in-crit(J) = false [owise] .
    eq {[J, rem ; R] | S, M} |= in-rem(J) = true .
    eq {[J, R] | S, M} |= in-rem(J) = false [owise] .
endm)
```

We can then check that the strong fairness property fails, and the model checker returns a counterexample. The result is translated to the same format as that of the LTL model checker.

```
Maude> (tlr check(initial, []<> exec(1) -> []<> in-crit(1)) .)
result : counterexample({{[1,repeat 'c1 := 1 ; while 'c2 = 1 do
   if 'turn = 2 then 'c1 := 0 ;  while 'turn = 2 do skip od ; ...
```

Instead, the (somewhat subtle) weaker fairness property satisfied by Dekker's algorithm can be verified as follows.

```
Maude> (tlr check(initial,  []<> exec(1) /\ []<> exec(2) ->
    []<> ~ in-rem(1) -> []<> in-crit(1)) .)
result : true
```

If a given property is a LTL formula, the model checker do LTL model checking without transformation, such as the *mutual exclusion property* like the following:

```
Maude> (tlr check(initial, [] ~ (in-crit(1) /\ in-crit(2))) .)
ltl-result : true
```

6 Related Work and Conclusions

There is much related work on both state-based and action-based logics (see [31] for a more thorough discussion). Related temporal logics include: (i) state-based logics; (ii) action-based logics; and (iii) mixed logics supporting both actions and state predicates. Well-known state-based logics such as *LTL*, *CTL*, and CTL^* (see, e.g., [27], [11]), are all special cases of TLR^*. Faithful translations to TLR^* from well-known action-based logics such as Hennessy-Milner logic [22] and *A-*CTL^* [35] are defined in detail in [31]. The mixed action-state logic *SE-LTL* in [8,9] can also be viewed as special case of the TLR^*. The Spatial Logic for Concurrency of Caires and Cardelli [5,6] is a state-based spatial modal logic for process calculi in the π-calculus spirit with spatial features used only for *state predicates* (but see the logic in [4], which has an action-labeled diamond). Other action-based temporal and modal logics are discussed in the survey paper [29], including the modal μ-calculus [25] (μL), which is in some ways more powerful than TLR^*, but lacks spatial action patterns. Among other logics supporting both actions and state predicates we find several extensions of either *A-*CTL^* or *A-CTL* such as, e.g., [2,19,21,36]. Three other approaches proposing mixed logics with both state-predicates and actions are: (i) the extension of the *SE-LTL* in [8,9] to a universally path quantified logic involving ω-regular expressions [7]; (ii) the *ESTL* logic of events and states for Petri nets of [24]; and (iii) the Kripke modal transition systems of [23], and their use in the verification of safety and liveness properties in the context of the modal μ-calculus. The work most closely related to TLR^* is that on $VLRL$ [20,28]. The $VLRL$ solution was less general and did not consider model checking aspects. Two other logics that combine actions and state-based formulas are the UNITY logic of Chandy and Misra [10], and Misra's logic for Seuss [34]; however, actions as such do not appear in temporal logic formulas, which remain state-based. Methodologically and technically, Lamport's Temporal Logic of Actions (TLA) [26] and TLR^* are very different. In TLA, there is no division of labor between system and property specification logics: TLA plays both roles simultaneously [26]. Also, actions in TLA are interpreted as binary relations between states, so that one cannot distinguish between two actions having the same outcomes from a given state.

In conclusion, after reviewing background on rewriting logic and the temporal logic of rewriting we have presented the formal foundations of the Maude $LTLR$ model checker and explained its reflective implementation as an extension of the Full Maude language. We have also presented an example illustrating the use of the model checker. The $LTLR$ model checker presented here is a useful prototype that can be used for both teaching and research and that, as mentioned, is available at [1]. The theoretical complexity results given in [31] mean that in practice the growth in the number of states is reasonable. Also, thanks to the efficient implementation of reflective features in Maude, our experience with examples suggest that the performance of the $LTLR$ model checker is acceptable, so that it can be used in both teaching and experimental research. In particular, for pure LTL formulas it has essentially the same performance as that of Maude's LTL model checker, which is quite good.

We believe, however, that a greater efficiency, obviating even the need for any growth in the state space of the given rewrite theory $\mathcal{R}$, and avoiding any reflective computations can be achieved by a new, native $LTLR$ model checking algorithm. However, the design of such an algorithm and its implementation in Core Maude are both nontrivial tasks than have to be left for future research.

References

[1] Bae, K., *Tlr model checker source code*, Available at http://www.cs.uiuc.edu/homes/kbae4/tlr/ (2008).

[2] Beek, M., A. Fantechi, S. Gnesi and F. Mazzanti, *An action/state-based model-checking approach for the analysis of communication protocols for Service-Oriented Applications*, in: *Proc.* FMICS (2008), to appear.

[3] Bruni, R. and J. Meseguer, *Semantic foundations for generalized rewrite theories.*, Theoretical Computer Science **360** (2006), pp. 386–414.

[4] Caires, L., *Behavioral and spatial observations in a logic for the pi-calculus*, in: *Foundations of Software Sceince and Computation Structures* (2004), pp. 72–87.

[5] Caires, L. and L. Cardelli, *A spatial logic for concurrency (part I).*, Information and Computation **186** (2003), pp. 194–235.

[6] Caires, L. and L. Cardelli, *A spatial logic for concurrency - II.*, Theoretical Computer Science **322** (2004), pp. 517–565.

[7] Chaki, S., E. Clarke, O. Grumberg, J. Ouaknine, N. Sharygina, T. Touili and H. Veith, *State/event software verification for branching-time specifications*, in: *Proc. 5th Intl. Conf. on Integrated Formal Methods (IFM'05)* (2005), pp. 53–69.

[8] Chaki, S., E. Clarke, J. Ouaknine, N. Sharygina and N. Sinha, *State/event-based software model checking*, in: *Proc. 4th Intl. Conf. on Integrated Formal Methods (IFM'04)* (2004), pp. 128–147.

[9] Chaki, S., E. Clarke, J. Ouaknine, N. Sharygina and N. Sinha, *Concurrent software verification with states, events, and deadlocks*, Formal Aspects of Computing **17** (2005), pp. 461–483.

[10] Chandy, K. M. and J. Misra, "Parallel Program Design: A Foundation," Addison-Wesley, 1988.

[11] Clarke, E. M., O. Grumberg and D. A. Peled, "Model Checking," MIT Press, 2001.

[12] Clavel, M., "Reflection in Rewriting Logic: Metalogical Foundations and Metaprogramming Applications," CSLI Publications, 2000.

[13] Clavel, M., F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet and J. Meseguer, *Metalevel computation in Maude*, *Proc. 2nd Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, Vol. 15, North Holland, 1998.

[14] Clavel, M., 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** (2002), pp. 187–243.

[15] Clavel, M., F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet and C. Talcott, "All About Maude – A High-Performance Logical Framework," LNCS **4350**, Springer, 2007.

[16] Dershowitz, N. and J.-P. Jouannaud, *Rewrite systems*, in: J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, North-Holland, 1990 pp. 243–320.

[17] Durán, F. and J. Meseguer, *Maude's module algebra*, Science of Computer Programming **66** (2007), pp. 125–153.

[18] Eker, S., 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* (2002).

[19] Fantechi, A., S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese and F. Tiezzi, *A model checking approach for verifying COWS specifications*, in: *Proc.* FASE 2008 (2008), to appear.

[20] Fiadeiro, J., N. Martí-Oliet, T. Maibaum, J. Meseguer and I. Pita, *Towards a verification logic for rewriting logic*, in: D. Bert, C. Choppy and P. D. Mosses, editors, *Recent Trends in Algebraic Development Techniques, WADT'99*, Springer LNCS **1827** (2000), pp. 438–458.

[21] Gnesi, S. and F. Mazzanti, *A Model Checking Verification Environment for UML Statecharts*, in: *Proceedings XLIII AICA Annual Conference, University of Udine - AICA 2005*, 2005.
URL http://fmt.isti.cnr.it/WEBPAPER/gmaica2005.pdf

[22] Hennessy, M. and R. Milner, *Algebraic laws for nondeterminism and concurrency*, Journal of the Association for Computing Machinery **32** (1985), pp. 137–172.

[23] Huth, M., R. Jagadeesan and D. Schmidt, *Modal transition systems: A foundation for three-valued program analysis*, in: *Proc. 10th European Symposium on Programming (ESOP'01)* (2001), pp. 155–169.

[24] Kindler, E. and T. Vesper, *ESTL: A temporal logic for events and states*, in: *Proc. 19th Intl. Conf. on Application and Theory of Petri Nets (ICATPN'98)* (1998), pp. 365–384.

[25] Kozen, D., *Results on the propositional mu-calculus*, Theoretical Computer Science **27** (1983), pp. 333–354.

[26] Lamport, L., *A temporal logic of actions*, ACM Trans. on Prog. Lang. and Systems **16** (1994), pp. 872–923.

[27] Manna, Z. and A. Pnueli, "The Temporal Logic of Reactive and Concurrent Systems – Specification," Springer-Verlag, 1992.

[28] Martí-Oliet, N., I. Pita, J. L. Fiadeiro, J. Meseguer and T. S. E. Maibaum, *A verification logic for rewriting logic.*, J. Log. Comput. **15** (2005), pp. 317–352.

[29] Mateescu, R., *Logiques temporelles basées sur actions pour la vérification des systèmes asynchrones*, Technique et Science Informatiques **22** (2003), pp. 461–495, also, INRIA Report 5032, Dec. 2003.

[30] Meseguer, J., *Conditional rewriting logic as a unified model of concurrency*, Theoretical Computer Science **96** (1992), pp. 73–155.

[31] Meseguer, J., *The temporal logic of rewriting*, Technical Report UIUCDCS-R-2007-2815, CS Dept., University of Illinois at Urbana-Champaign (2007), revised November 2007.

[32] Meseguer, J., *The temporal logic of rewriting: A gentle introduction*, LNCS **5065** (2008), to appear in Festschrift in honor of Ugo Montanari.

[33] Meseguer, J., M. Palomino and N. Martí-Oliet, *Equational abstractions*, in: *Procs. of CADE'03*, LNCS (2003).

[34] Misra, J., "A Discipline of Multiprogramming," Springer-Verlag, 2001.

[35] Nicola, R. D. and F. W. Vaandrager, *Action versus state based logics for transition systems*, in: *Semantics of Systems of Concurrent Processes*, LNCS **469** (1990), pp. 407–419.

[36] Pecheur, C. and F. Raimondi, *Symbolic model checking of logics with actions*, in: *Workshop on Model Checking and Artificial Intelligence (MOCHART)* (2006), pp. 113–128.

[37] Viry, P., *Equational rules for rewriting logic*, Theoretical Computer Science **285** (2002), pp. 487–517.