Simulation and Verification of Synchronous Set Relations in Rewriting Logic

Camilo Rocha[1] and César Muñoz[2]

[1] University of Illinois at Urbana-Champaign
[2] NASA Langley Research Center

Abstract. This paper presents a mathematical foundation and a rewriting logic infrastructure for the execution and property verification of synchronous set relations. The mathematical foundation is given in the language of abstract set relations. The infrastructure consists of an order-sorted rewrite theory in Maude, a rewriting logic system, that enables the synchronous execution of a set relation provided by the user. By using the infrastructure, existing algorithm verification techniques already available in Maude for traditional *asynchronous* rewriting, such as reachability analysis and model checking, are automatically available to synchronous set rewriting. The use of the infrastructure is illustrated with an executable operational semantics of a simple synchronous language and the verification of temporal properties of a synchronous system.

1 Introduction

Synchronous set relations provide a natural model for describing the operational semantics of synchronous languages. Previous work by the authors [11] gives a *serialization procedure* for simulating the execution of synchronous set relations by *asynchronous* term rewriting. The synchronous execution of a set relation is a parallel reduction, where the terms to be reduced in parallel are selected according to some strategy. The serialization procedure has been used to provide the rewriting logic semantics of the Plan Execution Interchange Language (PLEXIL) [5], a synchronous plan execution language developed by NASA to support spacecraft automation [6].

Despite being generic, the serialization procedure proposed in [11] has to be coded by the user for each synchronous language. This paper extends that work in two ways. First, it generalizes the theoretical development of synchronous set relations by extending the notion of strategy to enable a larger set of synchronous transformations. Second, it introduces an infrastructure in Maude [4], a high-performance reflective language and system supporting asynchronous set rewriting, that implements on-the-fly a serialization procedure for a synchronous language provided by the user. These contributions allow for simpler and more succinct language specifications, and more general synchronous set relations.

Formally, a synchronous set relation is defined as the *synchronous closure* of an *atomic* relation with a given *strategy*. Two sets are synchronously related if

the first set can be transformed into the second set by parallel atomic transformations. The selection of the redexes in the source set is done by the strategy. Strategies can be defined using priorities, which solve conflicts arising from the overlapping of atomic transitions. Section 2 presents, in an abstract setting, definitions of synchronous set relations, strategies, and priorities.

The infrastructure presented in this paper uses the reflection capabilities of Maude's rewriting logic, which is succinctly described in Section 3.1. Maude supports set rewriting, i.e., rewriting modulo axioms such as associativity, commutativity, and identity. These features are well-suited for object-based concurrent systems. The infrastructure consists of a rewrite theory in Maude, defining a set of generic sorts and terms, the algebraic properties of the datatypes, and a set of functions and rewrite rules that support the synchronous execution of an atomic set relation. The infrastructure is described in sections 3.2 and 3.3.

As a direct advantage of using this infrastructure, all commands in Maude for rewrite theories such as its rewrite and search commands, and formal verification tools such as Maude's LTL Model Checker, are available for analyzing properties of synchronous set relations. Section 4 illustrates the use of the infrastructure by giving an executable semantics of a simple synchronous language with arithmetic expressions. Section 5 illustrates the use of Maude's LTL Model Checker for the verification of temporal properties of a synchronous set relation.

The infrastructure in Maude and the examples presented in this paper are available from `http://shemesh.larc.nasa.gov/people/cam/PLEXIL`.

2 Abstract Synchronous Set Relations

This section introduces the concepts of abstract set relations used in this paper.

Let $\mathcal{U}$ be a set whose elements are denoted $A, B, \ldots$ and let $\rightarrow$ be a binary relation on $\mathcal{U}$. An element $A \in \mathcal{U}$ is called a $\rightarrow$-*redex* if there exists $B \in \mathcal{U}$ such that the *pair* $\langle A\,;B\rangle \in \rightarrow$. The expressions $A \rightarrow B$ and $A \not\rightarrow B$ denote $\langle A\,;B\rangle \in \rightarrow$ and $\langle A\,;B\rangle \notin \rightarrow$, respectively. The *identity* relation and *reflexive-transitive closure* of $\rightarrow$ are defined as usual and denoted $\rightarrow^0$ and $\rightarrow^*$, respectively.

Henceforth, it is assumed that $\mathcal{U}$ is the family of all *nonempty* finite sets over an abstract and possibly infinite set T, i.e., $\mathcal{U} \subseteq \wp(T)$ and $\emptyset \notin \mathcal{U}$, and, therefore, $\rightarrow$ is a binary relation on finite sets of T. The elements of T will be denoted by lowercase letters $a, b, \ldots$. When it is clear from the context, curly brackets are omitted from set notation, e.g., $a, b \rightarrow b$ denotes $\{a, b\} \rightarrow \{b\}$. Because of this abuse of notation, the symbol ',' is overloaded to denote set union, e.g., if A denotes the set $\{a, b\}$, B denotes the set $\{c, d\}$, and D denotes the set $\{d, e\}$, notation $A, B \rightarrow B, D$ denotes $\{a, b, c, d\} \rightarrow \{c, d, e\}$.

The *parallel* relation $\rightarrow^{\|}$ of $\rightarrow$ is the relation defined as the parallel closure of $\rightarrow$, i.e., the set of pairs $\langle A\,;B\rangle$ in $\mathcal{U} \times \mathcal{U}$ such that $A \rightarrow^{\|} B$ if and only if there exist $A_1, \ldots, A_n$, (nonempty) pairwise disjoint subsets of A, and sets $B_1, \ldots, B_n$ such that $A_i \rightarrow B_i$ and $B = (A \setminus \bigcup_{1 \leq i \leq n} A_i) \cup \bigcup_{1 \leq i \leq n} B_i$.

This paper focuses on synchronous set relations. The synchronous relation of an abstract set relation $\rightarrow$ is defined as a subset of the parallel closure of

$\rightarrow$, where a given strategy selects elements from $\rightarrow$. Formally, a *$\rightarrow$-strategy* is a function s that maps an element $A \in \mathcal{U}$ into a set $s(A) \subseteq \wp(\rightarrow)$ such that if $\{\langle A_1 ; B_1\rangle, \ldots, \langle A_n ; B_n\rangle\} \in s(A)$, then $A_i \subseteq A$ and $A_i \rightarrow B_i$, for $1 \leq i \leq n$, and $A_1, \ldots, A_n$ are pairwise disjoint.

Definition 1 (Synchronous Relation). *Let s be a $\rightarrow$-strategy. The relation $\rightarrow^s$ denotes the set of pairs $\langle A ; B\rangle$ in $\mathcal{U} \times \mathcal{U}$ such that $A \rightarrow^s B$ if and only if $B = (A \setminus \bigcup_{1\leq i\leq n} A_n) \cup \bigcup_{1\leq i\leq n} B_n$, where $\{\langle A_1 ; B_1\rangle, \ldots, \langle A_n ; B_n\rangle\} \in s(A)$.*

Example 1. Let T be the set of distinct elements a, b, c, d, e, and the relation $\rightarrow = \{r_1, r_2, r_3\}$, where $r_1 = \langle a, b ; b, d\rangle$, $r_2 = \langle c ; d\rangle$, and $r_3 = \langle a, c ; e\rangle$. Let s_1, s_2, and s_3 be $\rightarrow$-strategies defined for $A = \{a, b, c, d\}$ as follows.

$$s_1(A) = \{\, \{r_2\}, \{r_3\} \,\}, \quad s_2(A) = \{\, \{r_1, r_2\} \,\}, \quad s_3(A) = \{\, \{r_1, r_2\}, \{r_3\} \,\}.$$

It holds that:

$$a, b, c, d \rightarrow^{s_1} a, b, d, \qquad a, b, c, d \rightarrow^{s_1} b, d, e, \qquad a, b, c, d \rightarrow^{s_2} b, d,$$
$$a, b, c, d \rightarrow^{s_3} b, d, \qquad a, b, c, d \rightarrow^{s_3} b, d, e.$$

Some strategies relevant to the operational semantics of synchronous languages are those strategies defined based on a priority. A *priority* $\prec$ for a relation $\rightarrow$ is a $\mathcal{U}$-indexed set $\prec = \{\prec_A\}_{A\in\mathcal{U}}$ with each $\prec_A$ a strict partial order on $\rightarrow \cap (\wp(A) \times \mathcal{U})$. Priorities can be used to decide between overlapping redexes.

Definition 2 (Saturation). *A set $\{\langle A_1; B_1\rangle, \ldots, \langle A_n; B_n\rangle\} \subseteq \rightarrow$ is* $\prec$-saturated for $A \in \mathcal{U}$ *(or* $\prec_A$-saturated*), with $\prec$ be a priority for $\rightarrow$, if and only if*

1. *the sets $A_1, \ldots, A_n$ are nonempty pairwise disjoint subsets of A,*
2. *each $\langle A_i ; B_i\rangle$ is such that for any $A' \rightarrow B'$ with $A' \subseteq A$ and $A' \cap A_i \neq \emptyset$, $\langle A_i ; B_i\rangle \not\prec_A \langle A' ; B'\rangle$, and*
3. *if there is $A' \rightarrow B'$ with $\langle A' ; B'\rangle \notin \{\langle A_1 ; B_1\rangle, \ldots, \langle A_n ; B_n\rangle\}$ and $A' \subseteq A$, then either*
   (i) there is $\langle A_j ; B_j\rangle$, for some $1 \leq j \leq n$, such that $A_j \cap A' \neq \emptyset$ or
   (ii) there is $A'' \rightarrow B''$ with $A'' \subseteq A$, $A'' \cap A' \neq \emptyset$, and $\langle A' ; B'\rangle \prec_A \langle A'' ; B''\rangle$.

A $\prec_A$-saturated set is a complete collection of non-overlapping redexes in a term $A \in \mathcal{U}$, where any overlapping is resolved by keeping $\prec$-maximal redexes. Note that the $\prec$-maximality tests in conditions (2) and (3) of Definition 2, are given with respect to *all* pairs $\langle A' ; B'\rangle$ in $\prec_A$, and hence $\prec_A$-saturation exclusively depends on the ordering of the finitely many subsets of $\rightarrow \cap (\wp(A) \times \mathcal{U})$.

Example 2. Recall the relation $\rightarrow = \{r_1, r_2, r_3\}$ and the set $A = \{a, b, c, d\}$ from Example 1. Let $\prec_A^1$ be such that $r_1 \prec_A^1 r_3$. It holds that the sets $\{r_2\}$ and $\{r_3\}$ are $\prec_A^1$-saturated. However, the set $\{r_1, r_2\}$ is not $\prec_A^1$-saturated because r_1 falsifies condition (2) in Definition 2 with witness r_3. Let $\prec_A^2$ be such that $r_3 \prec_A^2 r_1$. In this case, the only $\prec_A^2$-saturated set is $\{r_1, r_2\}$. The set $\{r_3\}$ is not $\prec_A^2$-saturated because r_3 falsifies condition (2) in Definition 2 with witness r_1. For $\prec_A^3 = \emptyset$, the sets $\{r_1, r_2\}$ and $\{r_3\}$ are the only $\prec_A^3$-saturated sets.

A maximal strategy defines the most general synchronous behavior of a relation, which is given by *all* saturated sets.

Definition 3 (Maximal Strategies). *Let $\prec$ be a priority for $\rightarrow$. A $\rightarrow$-strategy s is* $\prec$-maximal for $A \in \mathcal{U}$ *(or* $\prec_A$-maximal*) if and only if $s(A)$ is the collection of all $\prec_A$-saturated sets. A $\rightarrow$-strategy is $\prec$-maximal if and only if it is $\prec_A$-maximal for all $A \in \mathcal{U}$.*

Example 3. From examples 1 and 2, $\rightarrow$-strategies s_1, s_2, and s_3 are, respectively, $\prec_A^1$-maximal, $\prec_A^2$-maximal, and $\prec_A^3$-maximal.

Algorithm 1 witnesses the existence of maximal strategies, which are unique for a given relation $\rightarrow$ and a priority $\prec$ (for $\rightarrow$).

Theorem 1. *Let $\prec$ be a priority for $\rightarrow$. Then a $\prec$-maximal $\rightarrow$-strategy exists. Therefore, from Definition 3, the $\prec$-maximal $\rightarrow$-strategy is unique.*

Proof. It is proved that the existence of a $\prec$-maximal $\rightarrow$-strategy is witnessed by Algorithm 1, for any $A \in \mathcal{U}$ and priority $\prec$ for $\rightarrow$. First, the following are important and easy to prove remarks about Algorithm 1:

- all three loops (lines 3, 6, and 12) repeat finitely many times and all quantified conditions (lines 7 and 4) require finitely many comparisons because $A \in \mathcal{U}$ has finitely many elements; also the complexity of γ decreases with each iteration of the third loop, i.e., Algorithm 1 terminates,
- $\alpha = \rightarrow \cap (\wp(A) \times \mathcal{U})$ is finite and can be computed effectively,
- $\beta = \alpha \setminus \{\langle A' ; , B' \rangle \in \alpha \mid (\exists \langle A'' ; B'' \rangle \in \alpha) A' \cap A'' \neq \emptyset \wedge \langle A' ; B' \rangle \prec_A \langle A'' ; B'' \rangle\}$, i.e., β is the subset of α in which all conflicting pairs in α that are not maximal elements in $\prec_A$ have been omitted,
- $\sigma \subseteq \wp(\beta)$ is the collection of largest non-conflicting subsets of β, and
- if $C \in \sigma$, then for any nonempty $C' \subseteq (\beta \setminus C)$, $C \cup C' \notin \sigma$.

Let $D = \{\langle A_1 ; B_1 \rangle, \ldots, \langle A_n ; B_n \rangle\}$. It is enough to prove, for $A \in \mathcal{U}$ and priority $\prec$ for $\rightarrow$, that D is $\prec_A$-saturated if and only if $D \in \sigma$.

($\Longrightarrow$) If D is $\prec_A$-saturated, then $D \subseteq \alpha$ follows by definition. If $D \not\subseteq \beta$, then there is $\langle A_i ; B_i \rangle \in D$ satisfying $\langle A_i ; B_i \rangle \prec_A \langle A' ; B' \rangle$ for some $\langle A' ; B' \rangle \in \alpha$ with $A' \cap A_i \neq \emptyset$. But then, for D, $\langle A_i ; B_i \rangle$ violates condition (2) in Definition 2, a contradiction. Hence $D \subseteq \beta$. If $D \notin \sigma$, since $D \subseteq \beta$ and the $A_1, \ldots, A_n$ are pairwise disjoint by assumption, either there is a nonempty set $D' \subseteq \beta \setminus D$ such that $D \cup D' \in \sigma$ or there is nonempty set $D'' \subsetneq D$ such that $D'' \in \sigma$. If $D \cup D' \in \sigma$ and since D' is nonempty, any pair $\langle A' ; B' \rangle \in D'$ violates condition (3.ii) in Definition 2, contradicting the $\prec_A$-maximality of D. If $D'' \in \sigma$, then for any pair $\langle A'' ; B'' \rangle \in D \setminus D''$ the set $C = D'' \cup \{\langle A'' ; B'' \rangle\}$ falsifies the test in line 14 of Algorithm 1 and hence $C \in \sigma$. Since $D'' \in \sigma$ and $D'' \subsetneq C \in \sigma$, this contradicts the last remark aforementioned. Therefore, as desired, $D \in \sigma$.

```
Input  : A ∈ 𝒰 and priority ≺ for →.
Output: s(A), with s the ≺_A-maximal →-strategy.
1  begin
2      α, β, γ, σ ← ∅, ∅, ∅, ∅;
3      for A_i →-redex, A_i ⊆ A, and B_i such that A_i → B_i do
4          add ⟨A_i ; B_i⟩ to α;
5      end
6      for ⟨A_i ; B_i⟩ ∈ α do
7          if (∀⟨A' ; B'⟩ ∈ α) (A_i ∩ A') ≠ ∅ ⟹ ⟨A_i ; B_i⟩ ⊀ ⟨A' ; B'⟩ then
8              add ⟨A_i ; B_i⟩ to β;
9          end
10     end
11     γ ← {β};
12     while γ ≠ ∅ do
13         remove C from γ;
14         if (∃⟨A_i ; B_i⟩, ⟨A_j ; B_j⟩ ∈ C) with i ≠ j and A_i ∩ A_j ≠ ∅
15         then add C \ {⟨A_i ; B_i⟩} and C \ {⟨A_j ; B_j⟩} to γ;
16         else add C to σ;
17     end
18     return σ;
19 end
```

Algorithm 1: The $\prec$-maximal $\rightarrow$-strategy.

($\Longleftarrow$) If $D \in \sigma \subseteq \wp(\alpha)$, then $A_1, \ldots, A_n$ are pairwise disjoint $\rightarrow$-redexes, thus subsets, of A. Thus, condition (1) in Definition 2 is satisfied. For condition (2), since $D \in \sigma$, it follows that $D \subseteq \beta$. Hence, any $\langle A_i \,;\, B_i \rangle \in D$ satisfies condition (2) in Definition 2. For condition (3), assume there is $\langle A' \,;\, B' \rangle \in \alpha$ with $\langle A' \,;\, B' \rangle \notin D$. Then, either $\langle A' \,;\, B' \rangle \in (\beta \setminus D)$ or $\langle A' \,;\, B' \rangle \in (\alpha \setminus \beta)$. If $\langle A' \,;\, B' \rangle \in (\beta \setminus D)$, then $D \cup \{\langle A' \,;\, B' \rangle\} \notin \sigma$, as previously stated. However, $\langle A' \,;\, B' \rangle \in \beta$, so it must be the case that $A' \cap A_i \neq \emptyset$ for some $1 \leq i \leq n$. If $\langle A \,;\, B' \rangle \in (\alpha \setminus \beta)$, then $\langle A' \,;\, B' \rangle \prec_A \langle A'' \,;\, B'' \rangle$ for some $\langle A'' \,;\, B'' \rangle \in \alpha$. In either case, D satisfies condition (3) in Definition 2. Thus, D is $\prec_A$-saturated. □

The definitions of strategy and maximal strategy used in this paper are more general than those in [11, §2]. In that paper, the only possible nondeterminism in $\rightarrow^s$ arises from $\rightarrow$. In the formalization presented in this paper, as illustrated by strategies s_1 and s_3, the synchronous relation $\rightarrow^s$ can be nondeterministic even when the relation $\rightarrow$ is deterministic.

3 Synchronous Set Relations in Rewriting Logic

This section presents the infrastructure for specifying and executing in Maude a synchronous relation defined from a language $\mathcal{L}$.

3.1 A Brief Overview of Rewriting Logic

An *order-sorted signature* [2] is a triple $\Sigma = (S, \leq, F)$, where $(S, \leq)$ is a finite poset of sorts and F is a finite set of function symbols. Set $X = \{X_s\}_{s \in S}$ is an S-sorted family of disjoint sets of variables with each X_s countably infinite. The set of terms of sort s is denoted by $T_\Sigma(X)_s$ and the set of ground terms of sort s is denoted by $T_{\Sigma,s}$. It is assumed that for each sort s, $T_{\Sigma,s}$ is nonempty. Algebras $\mathcal{T}_\Sigma(X)$ and $\mathcal{T}_\Sigma$ denote the respective term algebras. The set of variables of a term t is written $vars(t)$ and is extended to sets of terms in the natural way. A term t is called *ground* if $vars(t) = \emptyset$. A *substitution* θ is a sorted map from a finite subset $dom(\theta) \subseteq X$ to $ran(\theta) \subseteq T_\Sigma(X)$ and extends homomorphically in the natural way. Substitution θ is called *ground* if $ran(\theta)$ is ground. Expression $t\theta$ denotes the application of θ to term t.

A Σ-*equation* is a sentence $t = u$ **if** *cond*, where $t = u$ is a Σ-*equality* with $t, u \in T_\Sigma(X)_s$, for some sort $s \in S$, and the *condition cond* is a finite conjunction of Σ-equalities. An *equational theory* is a pair (Σ, E) with order-sorted signature Σ and finite set of Σ-equations E. For a Σ-equation φ, the judgement $(\Sigma, E) \vdash \varphi$ states that φ can be derived from (Σ, E) by the deduction rules in [8]. In this case, it holds that φ is valid in all models of (Σ, E). An equational theory (Σ, E) induces the congruence relation $=_E$ on $T_\Sigma(X)$ defined for any $t, u \in T_\Sigma(X)$ by $t =_E u$ if and only if $(\Sigma, E) \vdash (\forall X)\, t = u$. The Σ-algebras $\mathcal{T}_{\Sigma/E}(X)$ and $\mathcal{T}_{\Sigma/E}$ denote the quotient algebras induced by $=_E$ over the algebras $\mathcal{T}_\Sigma(X)$ and $\mathcal{T}_\Sigma$. The algebra $\mathcal{T}_{\Sigma/E}$ is called the *initial algebra* of (Σ, E).

A Σ-*rule* is a sentence $\flat : t \Rightarrow u$ **if** *cond*, where $\flat$ is its *name*, $t \Rightarrow u$ is a Σ-*sequent* with $t, u \in T_\Sigma(X)_s$, for some sort $s \in S$, and the *condition cond* is a finite conjunction of Σ-equations. A *rewrite theory* is a tuple $\mathcal{R} = (\Sigma, E, R)$ with equational theory $\mathcal{E}_\mathcal{R} = (\Sigma, E)$ and a finite set of Σ-rules R. For $\mathcal{R} = (\Sigma, E, R)$ and $\flat$ a Σ-rule, the judgement $\mathcal{R} \vdash \flat$ states that $\flat$ can be derived from $\mathcal{R}$ by the deduction rules in [2]. In this case, it holds that $\flat$ is valid in all models of $\mathcal{R}$. For $\flat$ a Σ-equation, it can be proved that $\mathcal{R} \vdash \flat$ if and only if $\mathcal{E}_\mathcal{R} \vdash \flat$. A rewrite theory $\mathcal{R} = (\Sigma, E, R)$ induces the rewrite relation $\Rightarrow_\mathcal{R}$ on $T_{\Sigma/E}(X)$ defined for every $t, u \in T_\Sigma(X)$ by $[t]_E \Rightarrow_\mathcal{R} [u]_E$ if and only if there is a *one-step* rewrite proof $\mathcal{R} \vdash (\forall X)\, t \Rightarrow u$. Relations $\Rightarrow_\mathcal{R}$ and $\Rightarrow^*_\mathcal{R}$ respectively denote a one-step rewrite and an arbitrary length (but finite) rewrite in $\mathcal{R}$ from t to u. Model $\mathcal{T}_\mathcal{R} = (\mathcal{T}_{\Sigma/E}, \Rightarrow^*_\mathcal{R})$ is the *initial reachability model* of $\mathcal{R} = (\Sigma, E, R)$ [2].

The following conditions on a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ make rewriting with equations E and with rules R modulo E computable, and are assumed throughout this paper. First the set of equations E of $\mathcal{R}$ can be decomposed into a disjoint union $E' \uplus A$, with A a collection of axioms (such as associativity, and/or commutativity, and/or identity) for which there exists a *matching algorithm modulo* A producing a finite number of A-matching substitutions, or failing otherwise. The second condition is that the equations E' can be oriented into a set of *ground sort-decreasing*, *ground confluent*, and *ground terminating* rules $\overrightarrow{E'}$ modulo A. The expression $[\mathrm{can}_{\Sigma,E'/A}(t)]_A \in T_{\Sigma/A,s}$ will denote the E'-*canonical form* of $[t]_A$. The rules R in $\mathcal{R}$ are assumed to be *ground coherent* relative to the equations E' modulo A [14].

3.2 The Synchronous Language $\mathcal{L}$

Recall that definitions in Section 2 are given for an abstract set T, an abstract relation $\rightarrow$, and an abstract priority relation $\prec$. The language $\mathcal{L}$ is given by the user as an order-sorted rewrite theory $(\Sigma_{\mathcal{L}}, E_{\mathcal{L}}, R_{\mathcal{L}})$ that enables the definition of concrete mathematical objects $T_{\Sigma_{\mathcal{L}},Elem}$, $\rightarrow_{\mathcal{L}}$, and $\prec_{\mathcal{L}}$ that implement T, $\rightarrow$, $\prec$, respectively. The rewrite theory $(\Sigma_{\mathcal{L}}, E_{\mathcal{L}}, R_{\mathcal{L}})$ extends the rewrite theory (Σ, E, R), which provides an infrastructure with definitions of basic sorts and data structures that are suitable for specifying set rewriting systems. This rewrite theory exploits rewriting logic's reflection capabilities available in Maude to *soundly* and *completely* simulate the synchronous relation $\rightarrow^{s}_{\mathcal{L}}$, where s is the $\prec_{\mathcal{L}}$-maximal strategy for $\rightarrow_{\mathcal{L}}$.

The Set $T_{\Sigma_{\mathcal{L}},Elem}$. The set of ground terms $T_{\Sigma_{\mathcal{L}},Elem}$ of the rewrite theory $(\Sigma_{\mathcal{L}}, E_{\mathcal{L}}, R_{\mathcal{L}})$ implements the abstract set T of Section 2. The sort *Elem* represents *elements* in Σ having the form $\langle m \mid a_1 : e_1, \ldots, a_n : e_n \rangle$, where m is an identifier of sort *Eid* and $a_1 : e_1, \ldots, a_n : e_n$ is a *map* of sort *Map*. A map is a collection of *attributes*. An *attribute* is a pair $a : e$ where a is an attribute identifier of sort *Aid* and e is an expression of sort *Expr*. Attributes are a flexible way of defining the internal state of an element. Sorts *Aid* and *Eid* are declared as subsorts of *Expr*. The set $\mathcal{U}$ of Section 2 corresponds to the set of ground terms $T_{\Sigma_{\mathcal{L}},Ctx}$, where the sort *Ctx* represents sets of elements of sort *Elem*. A *context* is an element of sort *Ctx*. The sort *Val* is defined in Σ as a subsort of *Expr* and represents built-in values such as Boolean and numerical values. Function symbol $eval : Ctx \times Expr \longrightarrow Val$ is defined in Σ without any equational definition.

The user is free to extend the signature Σ in $\Sigma_{\mathcal{L}}$ with any syntax and subsorts for element identifiers, attribute identifiers, and expressions. However, it is assumed that attribute identifiers within a map and element identifiers within a context are unique. It is also assumed that the theory $(\Sigma_{\mathcal{L}}, E_{\mathcal{L}})$ includes a complete equational interpretation of *eval* for the set of expressions in $\Sigma_{\mathcal{L}}$.

The Relation $\rightarrow_{\mathcal{L}}$. The synchronous relation in Definition 1 is given for an abstract atomic relation $\rightarrow$. In a concrete language, such as $\mathcal{L}$, this relation represents atomic computational steps that are synchronously executed. For that reason, the concrete relation $\rightarrow_{\mathcal{L}}$ is called the *atomic relation*. As shown in [11], the atomic relation is usually parametric with respect to a context that, in this infrastructure, provides global information to the function *eval*. Henceforth, the atomic relation with respect to a context Γ of sort *Ctx* will be denoted $\overset{\Gamma}{\rightarrow}_{\mathcal{L}}$.

The atomic relation $\rightarrow_{\mathcal{L}}$ is specified in $R_{\mathcal{L}}$ through *atomic rules*.

Definition 4 (Atomic Rules). *Let $\Sigma_{\mathcal{L}}$ be an order-sorted signature extending Σ. An* atomic $\Sigma_{\mathcal{L}}$-rule *is a $\Sigma_{\mathcal{L}}$-rule $\flat : l \Rightarrow r$* **if** *cond such that:*

- *rule name $\flat$ has the form c–n, where c, the* component *of $\flat$, is an identifier, and n, the* rank *of $\flat$, is a natural number;*
- *l does not contain attribute identifier variables, i.e., $vars(l) \cap X_{Aid} = \emptyset$; and*

– *attribute names appearing in an element term in r are named for that same element term in l, i.e., if $\langle i \mid m' \rangle \in r$ and $(a : e') \in m'$, then there is $\langle i \mid m \rangle \in l$ such that $(a : e) \in m$ for some $e \in T_{\Sigma_\mathcal{L}}(X)_{Expr}$.*

An atomic $\Sigma_\mathcal{L}$-rule specifies transitions of contexts (possibly) constrained by a condition that may involve expressions in the syntax of $\mathcal{L}$. The component and rank of $\Sigma_\mathcal{L}$-rules are used to define the priority relation $\prec_\mathcal{L}$. The restriction on attribute identifier names and variables is to prevent the user from defining an atomic relation $\rightarrow_\mathcal{L}$ for which computing a $\rightarrow_\mathcal{L}$-reduction could be highly inefficient or even incorrect.

Definition 5 (Atomic Relation $\rightarrow_\mathcal{L}$). *Let $\mathcal{L} = (\Sigma_\mathcal{L}, E_\mathcal{L}, R_\mathcal{L})$ be a rewrite theory with $(\Sigma_\mathcal{L}, E_\mathcal{L})$ extending (Σ, E) and $R_\mathcal{L}$ a collection of atomic $\Sigma_\mathcal{L}$-rules with different names. For a rule $\flat : l \Rightarrow r$* **if** *$cond \in R_\mathcal{L}$, the (parametric) relation $\overset{\Gamma}{\rightarrow}_\flat$, with parameter $\Gamma \in T_{\Sigma_\mathcal{L}, Ctx}$, denotes the set of pairs $\langle A ; B \rangle$ in $T_{\Sigma_\mathcal{L}, Ctx} \times T_{\Sigma_\mathcal{L}, Ctx}$ such that there is a ground substitution $\theta : T_{\Sigma_\mathcal{L}}(X) \longrightarrow T_{\Sigma_\mathcal{L}}$ satisfying $cond\theta$, $A = l\theta$, and $B = r\theta$ in $\mathcal{L}$, where any expression is evaluated in Γ. The* atomic relation *$\rightarrow_\mathcal{L}$ is the indexed set $\{\overset{\Gamma}{\rightarrow}_\flat\}_{\Gamma \in T_{\Sigma_\mathcal{L}, Ctx}, \flat \in R_\mathcal{L}}$.*

In Definition 5, A, B, and Γ are ground terms of sort Ctx. Furthermore, the term B is a variant of A in which some expressions and attributes have been modified. In particular, A and B have the same number of elements with the same element and attribute identifiers. This means that the atomic relation does not delete or create elements or attributes in A. This restriction simplifies the technical development of (Σ, E, R). In any case, creation and deletion of elements and attributes can be encoded by using additional attributes. Also observe that, due to the syntactical restrictions of atomic rules in Definition 4, equational sentences $C\theta$, $A = l\theta$, and $B = r\theta$ can be checked in $(\Sigma_\mathcal{L}, E_\mathcal{L})$ because they are equational expressions that, although may depend on context Γ, do *not* depend on $R_\mathcal{L}$.

In general, the atomic relation $\rightarrow_\mathcal{L}$ and the rewrite relation $\Rightarrow_\mathcal{L}$ induced by the rewrite theory $\mathcal{L}$ do not coincide for ground context terms. In particular, $\rightarrow_\mathcal{L}$ is defined as the top-most application of the atomic rules, while $\Rightarrow_\mathcal{L}$ is defined as the congruence closure of those rules.

The Priority $\prec_\mathcal{L}$. For a given context Γ, the elements in $\overset{\Gamma}{\rightarrow}_\mathcal{L}$ can be regarded as tuples of the form $(A, B, c, m)_\Gamma$ as a shorthand for $A \overset{\Gamma}{\rightarrow}_{c\text{–}m} B$, with $c\text{–}m \in R_\mathcal{L}$. The set $\prec_\mathcal{L} = \{\prec_{\mathcal{L}(\Gamma)}\}_{\Gamma \in T_{\Sigma_\mathcal{L}, Ctx}}$ is defined automatically by the infrastructure:

$$(A', B', c', m')_\Gamma \prec_{\mathcal{L}(\Gamma)} (A, B, c, m)_\Gamma \quad \equiv \quad A \subseteq \Gamma \wedge A' \subseteq \Gamma \wedge c = c' \wedge m < m',$$

where $<$ is the usual order on natural numbers.

Lemma 1. *The indexed set $\prec_\mathcal{L}$ is a priority for $\rightarrow_\mathcal{L}$.*

Proof. It is enough to prove that $\prec_{\mathcal{L}(\Gamma)}$ is a strict partial order, for any $\Gamma \in T_{\Sigma_\mathcal{L}, Ctx}$. Irreflexivity of $\prec_{\mathcal{L}(\Gamma)}$ follows from the irreflexivity of $<$. Transitivity of

$\prec_{\mathcal{L}(\Gamma)}$ follows from the fact that if $(A'', B'', c'', m'')_\Gamma \prec_{\mathcal{L}(\Gamma)} (A', B', c', m')_\Gamma$ and $(A', B', c', m')_\Gamma \prec_{\mathcal{L}(\Gamma)} (A, B, c, m)_\Gamma$, then $A'' \subseteq \Gamma$, $A \subseteq \Gamma$, $c'' = c' = c$, and $m < m' < m''$. Therefore, $(A'', B'', c'', m'')_\Gamma \prec_{\mathcal{L}(\Gamma)} (A, B, c, m)_\Gamma$. □

The priority $\prec_{\mathcal{L}}$ is an indexed collection of strict partial orders. In particular, for each $\Gamma \in T_{\Sigma_{\mathcal{L}}, Ctx}$, priority $\prec_{\mathcal{L}(\Gamma)}$ compares two elements of $\rightarrow_{\mathcal{L}}$ if they are computed with the same context and they originate from atomic $\Sigma_{\mathcal{L}}$-rules having the same component. It assigns a higher priority to elements with smaller rank.

Rewrite theory (Σ, E, R) includes a function *max-strat* that computes the $\prec_{\mathcal{L}}$-maximal $\overset{\Gamma}{\rightarrow}_{\mathcal{L}}$-strategy, where $\Gamma \in T_{\Sigma_{\mathcal{L}}, Ctx}$ is the parameter of the relation $\rightarrow_{\mathcal{L}}$. That function implements Algorithm 1 of Section 2. It takes as input the language $\mathcal{L}$ and ground context Γ and returns the collection $s(\Gamma)$, where s is the $\prec_{\mathcal{L}}$-maximal $\rightarrow_{\mathcal{L}}$-strategy. The function *max-strat* is implemented in Maude using the meta-level capabilities of the system. Henceforth, the strategy s will denote the $\prec_{\mathcal{L}}$-maximal $\rightarrow_{\mathcal{L}}$-strategy as computed by *max-strat*.

3.3 Simulation of $\rightarrow^{s}_{\mathcal{L}}$

The set of Σ-rules R of the order-sorted rewrite theory (Σ, E, R) includes only one rule: for $l, r \in X_{Ctx}$, $T \in X_{Transition}$, and $S \in X_{TransitionSet}$

$$\begin{aligned} \mathit{sync} : \{l\} \Rightarrow \{r\} \quad & \textbf{if} \quad T, S := \mathit{max\text{-}strat}(\mathcal{L}, l) \\ & \wedge \quad r := \mathit{update}(l, T). \end{aligned}$$

This rule, along with the rules $R_{\mathcal{L}}$ provided by the user, implements the serialization algorithm defined in [11], which has been adapted to the notion of maximal strategy presented in this paper. Sort *Transition* denotes sets of pairs in $T_{\Sigma_{\mathcal{L}}}(X)_{Ctx}$ and sort *TransitionSet* denotes collections of transitions. Function *update* takes as inputs a ground context A and a ground transition term $C = \{\langle A_1 \,;\, B_1\rangle, \ldots, \langle A_n \,;\, B_n\rangle\}$, and computes the ground context $B = (A \setminus \bigcup_{1 \leq i \leq n} A_i) \cup \bigcup_{1 \leq i \leq n} B_i$.

It is noted that the rule *sync* acts on contexts that are syntactically wrapped by curly brackets, that is, terms of the form $\{A\}$ with A a ground context term. Those terms are of sort *SState*. The curly brackets operator prevents its context A to be directly rewritten by the user defined atomic rules in $R_{\mathcal{L}}$. The actual application of those rules is done by the function *update*.

Rule *sync* is *nondeterministic* because a ground substitution for l matching its condition depends on the choice of T, i.e., on *all* possible transitions computed by *max-strat*. However, there will be *exactly* one rewrite with *sync* for each transition.

Theorem 2. *Let $\mathcal{L} = (\Sigma_{\mathcal{L}}, E_{\mathcal{L}}, R_{\mathcal{L}})$ be an extension of (Σ, E, R). For $A, B \in T_{\Sigma_{\mathcal{L}}, Ctx}$, the following equivalence holds:*

$$\mathcal{L} \vdash \{A\} \Rightarrow \{B\} \qquad \equiv \qquad A \rightarrow^{s}_{\mathcal{L}} B,$$

where s denotes the $\prec_{\mathcal{L}}$-maximal $\rightarrow_{\mathcal{L}}$-strategy as computed by max-strat.

Proof. The key observation is that because *max-strat* computes the $\prec_{\mathcal{L}}$-maximal $\rightarrow_{\mathcal{L}}$-strategy s, the following equivalence holds:

$$C \in s(A) \qquad \equiv \qquad (\exists C' \in T_{\Sigma_{\mathcal{L}},\mathit{TransitionSet}})C, C' =_{E_{\mathcal{L}}} \mathit{max\text{-}strat}(\mathcal{L}, A).$$

($\Longrightarrow$) Since $\{A\}$ can be rewritten only by rule $\mathit{sync} \in R$, there is a ground substitution $\theta : X \longrightarrow T_{\Sigma_{\mathcal{L}}}$ satisfying $A =_{E_{\mathcal{L}}} l\theta$, $B =_{E_{\mathcal{L}}} r\theta$, $T\theta, S\theta =_{E_{\mathcal{L}}} \mathit{max\text{-}strat}(\mathcal{L}, l\theta)$, and $r\theta =_{E_{\mathcal{L}}} \mathit{update}(l\theta, T\theta)$. By the observation above, $T\theta \in s(A)$. Then, from the definition of *update*, it follows that $A \rightarrow^s_{\mathcal{L}} B$.

($\Longleftarrow$) If $A \rightarrow^s_{\mathcal{L}} B$, there is $C = \{\langle A_1 \,;\, B_1\rangle, \ldots, \langle A_n \,;\, B_n\rangle\} \in s(A)$ such that $B = (A \setminus \bigcup_{1\leq i\leq n} A_i) \cup \bigcup_{1\leq i\leq n} B_i$. By the observation above and the definition of *update*, there is $C' \in T_{\Sigma_{\mathcal{L}},\mathit{TransitionSet}}$ such that $C, C' =_{E_{\mathcal{L}}} \mathit{max\text{-}strat}(\mathcal{L}, A)$ and $B =_{E_{\mathcal{L}}} \mathit{update}(A, C)$. Then substitution θ satisfying $A =_{E_{\mathcal{L}}} l\theta$ witnesses $\mathcal{L} \vdash \{A\} \Rightarrow \{B\}$.

□

One key advantage of this approach is that, while it offers support for the execution of a synchronous relation $\rightarrow^s_{\mathcal{L}}$, it does that by simulating $\rightarrow^s_{\mathcal{L}}$ using the standard asynchronous semantics of Maude. Therefore, all commands available in Maude for executing and verifying rewrite relations are directly available for $\rightarrow^s_{\mathcal{L}}$. Sections 4 and 5 illustrate these features with practical examples.

4 Executable Semantics of a Simple Synchronous Language

Module *SMAUDE* implements in Maude the rewrite theory (Σ, E, R) presented in Section 3. This section illustrates the use of *SMAUDE* by giving the small-step semantics of a simple synchronous language with arithmetic expressions.

Consider a language that consists of two kinds of elements: *memory* elements $\mathtt{Mem}(m, v)$ and *assignment* elements $l{:=}e$, where m, l denote memory names, v denotes a numerical value, and e denotes an arithmetic expression. Arithmetic expressions are recursively formed using memory names, numerical values, and expressions of the form $e_1 + e_2$, where e_1 and e_2 are arithmetic expressions. In this case, set T consists of all elements having the form $\mathtt{Mem}(m, v)$ or $m{:=}v$.

The small-step semantics of the language requires the definition of an evaluation function *eval* that takes as inputs a context Γ, which is a set of elements T, and an arithmetic expression e. It is inductively defined on expressions:

$$\mathit{eval}(\Gamma, e) = \begin{cases} v & \text{if } e \text{ is the numerical value } v, \\ v & \text{if } e \text{ is the memory name } m \text{ and } \mathtt{Mem}(m, v) \in \Gamma, \\ v_1 + v_2 & \text{if } e \text{ has the form } e_1 + e_2,\ v_i = \mathit{eval}(\Gamma, e_i) \text{ for } i \in \{1, 2\}. \end{cases}$$

The (parametric) atomic relation $\rightarrow$ of the language is defined for a context Γ by $A \xrightarrow{\Gamma} B$ if and only if $A \subseteq \Gamma$, $A = \{\mathtt{Mem}(m, v), l{:=}e\}$, $B = \{\mathtt{Mem}(m, u), l{:=}e\}$, and $u = \mathit{eval}(A, e)$, for some memory name m, values v and u, and expression

e. The semantic relation of the language is the relation $\xrightarrow{\Gamma}{}^{s}$, where s is the $\prec$-maximal $\xrightarrow{\Gamma}$-strategy, Γ is a ground context, and $\prec$ is the empty priority.

Example 4. Let $\Gamma = \{\mathtt{Mem}(x,3), \mathtt{Mem}(y,4), x{:=}y, y{:=}x\}$. Then:

$$\mathtt{Mem}(x,3), \mathtt{Mem}(y,4), x{:=}y, y{:=}x \xrightarrow{\Gamma}{}^{s} \mathtt{Mem}(x,4), \mathtt{Mem}(y,3), x{:=}y, y{:=}x.$$

This language is specified by the Maude module *SIMPLE*, which includes system module *SMAUDE*:

$$\begin{array}{llll} a : Nat \to Eid & body : \to Aid & Nat \leq Val & \\ x : \to Eid & mem : \to Aid & + : Expr \times Expr \to Expr & \text{Memory ele-} \\ y : \to Eid & to : \to Aid & & \end{array}$$

ments use constructors x and y for element identifiers and have attribute *mem* as their only attribute. Assignment elements use constructors a for element identifiers and have attributes *body* and *to* as their only attributes. In the syntax of *SIMPLE*, memory element $\mathtt{Mem}(x,v)$ and an assignment element $x{:=}e$ are represented, for instance, by elements $\langle x \mid mem : v\rangle$ and $\langle a(1) \mid to : x, body : e\rangle$, respectively. Built-in natural numbers are values of the language. Evaluation of expressions is given equationally following the definition of *eval*.

Atomic rule r–1 specifies the atomic relation of the language:

$$r\text{–}1 : \langle I \mid mem{:}N\rangle\langle J \mid body{:}E, to{:}I\rangle \Rightarrow \langle I \mid mem{:}eval(E)\rangle.$$

The specification of atomic rules is slightly different to the usual specification of rules in rewriting logic. First, in the lefthand side of an atomic rule, it is sufficient to only mention the attributes involved in the atomic transition. In this case, *SMAUDE* will complete each lefthand side term by automatically adding a variable of sort *Map*, unique for each element, before any matching is performed. Second, in the righthand side of an atomic rule, it is sufficient to only mention the elements and the attributes that can change in the atomic step. In this case, *SMAUDE* updates in the current state *only* the attributes of the elements occurring in the righthand side of the rule, while keeping the other ones intact. So, in atomic rule r–1, the only attribute that can change is attribute *mem* of the memory element. Note also that in the righthand side of r–1 a unary version of function *eval*, without mention to any particular context, is used; *SMAUDE* will automatically extend it to its binary counterpart, for the given context, when computing function *max-strat*.

The context Γ in Example 4, written in the syntax of *SIMPLE*, is

$$\langle x \mid mem{:}3\rangle\langle y \mid mem{:}4\rangle\langle a(1) \mid to{:}x, body{:}y\rangle\langle a(2) \mid to{:}y, body{:}x\rangle.$$

Maude's *search* command can be used to compute, for instance, the one-step synchronous semantic relation of the language in Example 4 from context Γ:

```
Maude> search { Gamma } =>1 X:SState .
search in SIMPLE : { Gamma } =>1 X:SState .
Solution 1 (state 1)
states: 2  rewrites: 514 in 53ms cpu (54ms real) (9655 rewrites/second)
X:SState --> {< x | mem : 4 > < y | mem : 3 >
              < a(1) | body : y, to : x > < a(2) | body : x, to : y > }
No more solutions.
```

5 Verification of Synchronous Relations

This section illustrates the use of Maude's LTL Model Checker for the verification of properties of a synchronous relation.

Consider a system of *clocks* keeping track of hours and minutes. Each clock is modeled in rewrite theory *CLOCKS* by two elements, one displaying hours and the other displaying minutes:

$$\begin{array}{lll} h : Nat \rightarrow Eid & hour : \rightarrow Aid & min : Nat \rightarrow Expr \\ m : Nat \rightarrow Eid & min : \rightarrow Aid & Nat \leq Val \end{array}$$

Hour elements use constructor h for element identifiers and have attribute *hour* as its single attribute. Minute elements use constructor m for element identifiers and have attribute *min* as its single attribute. Natural numbers are used as values for the attributes. The n-th clock is represented by the hour element with element identifier $h(n)$ and the minute element with element identifier $m(n)$. Attribute *min* of a minute element $m(n)$ can be accessed by evaluating expression *min(n)*. A clock displaying 9:15, written in the syntax of *CLOCKS*, is

$$\langle h(1) \mid hour : 9 \rangle \; \langle m(1) \mid min : 15 \rangle .$$

The following clock transitions are of interest:

(i) if $hour = 11$ and $min = 59$, then set $hour = 0$ and $min = 0$;
(ii) if $hour < 11$ and $min = 59$, then increment *hour* in one unit and set $min = 0$;
(iii) if $hour < 11$ and $min < 59$, then increment *min* in one unit.

These transitions are intuitively coded via priorities in rewrite theory *CLOCKS*. The behavior of the system is modeled by defining a priority such that redexes of the form (i) have the highest priority and the ones of the form (iii) the lowest. The following are the atomic rules of *CLOCKS*, for $C, M, N \in X_{Nat}$:

$$\begin{array}{lll} cl\text{–}1 : & \langle h(C) \mid hour : 11 \rangle & \\ & \langle m(C) \mid min : 59 \rangle \; \Rightarrow & \langle h(C) \mid hour : 0 \rangle \langle m(C) \mid min : 0 \rangle \\ & & \\ cl\text{–}2 : & \langle h(C) \mid hour : N \rangle \; \Rightarrow & \mathit{if}\ eval(min(C)) == 59 \\ & & \mathit{then}\ \langle h(C) \mid hour : s(N) \rangle \\ & & \mathit{else}\ \langle h(C) \mid hour : N \rangle\ \mathit{fi} \\ & & \\ cl\text{–}3 : & \langle m(C) \mid min : N \rangle \; \Rightarrow & \mathit{if}\ N == 59 \\ & & \mathit{then}\ \langle m(C) \mid min : 0 \rangle \\ & & \mathit{else}\ \langle m(C) \mid min : s(N) \rangle\ \mathit{fi} \end{array}$$

In *CLOCKS*, *resetting* a clock (i.e., rule *cl*–1) has higher priority than exclusively increasing the hour (i.e., rule *cl*–2) or the minute (i.e., rule *cl*–3) of a clock. Rule *cl*–1 uses matching for detecting when a clock needs to be reset. In the righthand side of rule *cl*–2 the evaluation of expression *min*(C) will yield the minute value of clock C, freeing the lefthand side of the rule from explicitly mentioning the minutes element. Because of this, rules *cl*–2 and *cl*–3 can never

overlap and therefore can be executed in parallel. As a final remark, observe that the priorities of the last two rules can be switched without altering the behavior of the system, since their lefthand sides can never overlap.

Two temporal properties that the synchronous relation of *CLOCKS* must satisfy is that clocks are *always synchronized* and that each clock is *reset* infinitely often. These two properties are specified by propositions $\Pi = \{sync, reset\}$. Using the syntax of Maude's LTL Model Checker and for variables $C, C', H, M \in X_{Nat}$ and $\Gamma \in X_{Ctx}$, the propositions Π are defined in the equational theory *CLOCKS-PREDS* as follows:

$$sync : Nat \times Nat \rightarrow Prop \qquad reset : Nat \rightarrow Prop \qquad SState \leq State$$

$$\{\Gamma\} \models sync(C, C') = \begin{cases} true & \text{if } \langle h(C) \mid hour : H \rangle \langle m(C) \mid min : M \rangle \subseteq \Gamma \\ & \wedge \langle h(C') \mid hour : H \rangle \langle m(C') \mid min : M \rangle \subseteq \Gamma, \\ false & \text{otherwise.} \end{cases}$$

$$\{\Gamma\} \models reset(C) = \begin{cases} true & \text{if } \langle h(C) \mid hour : 0 \rangle \langle m(C) \mid min : 0 \rangle \subseteq \Gamma, \\ false & \text{otherwise.} \end{cases}$$

The subsort declaration $SState \leq State$ tells Maude's LTL Model Checker that the semantics of propositions Π (each with sort *Prop* –provided by the model checker) is to be defined on sort *SState*. Two clocks are synchronized if their hour values and minute values are the same; otherwise they are not synchronized. A clock is reset if its hour and minute values are 0.

Consider the following state *init* in the signature of *CLOCKS*

$$\{\langle h(1) \mid hour : 0 \rangle \langle m(1) \mid min : 0 \rangle \langle h(2) \mid hour : 0 \rangle \langle m(2) \mid min : 0 \rangle\},$$

with two clocks, both displaying 0:00. The two temporal properties aforementioned that the synchronous relation of *CLOCKS* must satisfy, are formally specified for state *init* as follows:

$$\mathcal{K}^{\Pi}_{CLOCKS}, init \models \Box sync(1, 2),$$
$$\mathcal{K}^{\Pi}_{CLOCKS}, init \models \Box\Diamond reset(1) \ \wedge \ \Box\Diamond reset(2),$$

where $\mathcal{K}^{\Pi}_{CLOCKS} = (T_{\Sigma/E, SState}, \Rightarrow_{CLOCKS}, L_{\Pi})$ is the Kripke structure associated to the initial reachability model $\mathcal{T}_{CLOCKS}$, with topsort *SState*, and predicates Π (see [4] for details on how $\mathcal{K}^{\Pi}_{CLOCKS}$ is associated to $\mathcal{T}_{CLOCKS}$).

First observe that the set of clock states reachable from *init* is finite and, therefore, each property specification problem is decidable. The first property specification asserts that clocks 1 and 2 are always synchronized, and the second property specification asserts that each clock is reset infinitely often.

By using Maude's LTL Model Checker, the following results are obtained:

```
Maude> red modelCheck(init, [] sync(1,2)) .
reduce in CLOCKS-PREDS : modelCheck(init, []sync(1, 2)) .
rewrites: 124946 in 6023ms cpu (6023ms real) (20744 rewrites/second)
result Bool: true

Maude> red modelCheck(init, ([] <> reset(1)) /\ ([] <> reset(2))) .
reduce in CLOCKS-PREDS : modelCheck(init, []<> reset(1) /\ []<> reset(2)) .
rewrites: 125514 in 6810ms cpu (6812ms real) (18428 rewrites/second)
result Bool: true
```

6 Conclusion

Rewriting logic has been used previously as a test bed for specifying and animating synchronous rewrite relations. M. AlTurki and J. Meseguer [1] have studied the rewriting logic semantics of the language Orc, which includes a synchronous reduction relation. T. Serbanuta *et al.* [13] and C. Chira *et al.* [3] define the execution of P-systems with structured data with continuations. The focus of the former is to use rewriting logic to study the (mainly) non-deterministic behavior of Orc programs, while the focus of the latter is to study the relationship between P-systems and the existing continuation framework for enriching each with the strong features of the other. D. Lucanu [7] studies the problem of the interleaving semantics of concurrency in rewriting logic for synchronous systems from the perspective of P-systems. More recently, T. Serbanuta [12] advances the rewriting-based framework $\mathbb{K}$ with resource sharing semantics that enables some kind of synchronous rewriting. J. Meseguer and P. Ölveczky [9] present a formal specification of the *physically asynchronous logically synchronous* architectural pattern as a formal model transformation that maps a synchronous design, together with performance bounds on the underlying infrastructure, to a formal distributed real-time specification that is semantically equivalent to the synchronous design.

The work presented in this paper is closely related to those works in that it presents techniques for specifying and executing synchronous rewrite relations. However, the work presented here is a first milestone towards the development of *symbolic* techniques for the analysis of synchronous set relations. In particular, the authors strongly believe that the infrastructure presented in Section 3 can be extended with rewriting and narrowing based techniques, in the style of [10], to obtain a deductive approach for verifying symbolic safety properties, such as invariance or race conditions, of synchronous set relations. Another feature that distinguishes this work from related work is the idea of priorities as an instrument to control nondeterminism of synchronous relations. Of course, in some cases priorities can be encoded in the condition of rewrite rules, but the treatment here seems more convenient and simpler for the end-user. One interesting exercise would be to study how best to implement this feature in the framework $\mathbb{K}$ and for real-time specifications in rewriting logic.

The contribution of this paper to rewriting logic research is the implementation of general synchronous set relations via asynchronous set rewrite systems. This work extends previous work reported in [11] by giving an on-the-fly implementation of the serialization procedure for rewrite theories that supports execution and verification of more general synchronous set relations. The framework exploits rewriting logic's reflective capabilities, and its implementation in Maude, to soundly and completely simulate the synchronous relation associated to an atomic relation and a maximal strategy specified by atomic rules. This work also generalizes the concept of priority, so that more general synchronous set relations are supported both theoretically and in the Maude infrastructure. A priority, as treated in this work, allows for nondeterministic synchronous relations even when the atomic relation is deterministic. In [11], the only possible

nondeterminism in a synchronous relations arises from its atomic relation. A direct benefit to the user from using the infrastructure presented in this paper, is the wealth of Maude's *ground* analysis tools for rewrite theories such as its rewrite and search commands, and its LTL Model Checker.

Although the framework is illustrated with simple examples, it is currently being used to specify an executable semantics in Maude of the Plan Execution Interchange Language (PLEXIL) [5], an open source synchronous language developed by NASA to support autonomous spacecraft operations. This specification enables the application of formal verification techniques available in Maude, such as model-checking and reachability analysis, to PLEXIL programs.

The Maude infrastructure presented in this work is a first prototype of the theoretical developments. Future work includes the development of a wider range of case studies stressing the infrastructure's capabilities; it is also important to streamline the algorithms and data structures in the infrastructure. Future work in the area of deductive analysis will study symbolic reachability analysis techniques in rewriting logic for synchronous set relations. More specifically, adapting the rewriting and narrowing based techniques developed in [10], seems promising for the analysis of safety properties of synchronous set relations.

Acknowledgments. The authors would like to thank the anonymous referees for their comments, which helped to improve the paper. This work is supported by NASA's Autonomous Systems and Avionics Project, Software Verification Algorithms. The first author has been partially supported by NSF grant CCF 09-05584 and by the National Aeronautics and Space Administration at Langley Research Center under Research Cooperative Agreement No. NNL09AA00A awarded to the National Institute of Aerospace.

References

1. M. AlTurki and J. Meseguer. Reduction semantics and formal analysis of Orc programs. *Electronic Notes in Theoretical Computer Science*, 200(3):25 – 41, 2008. Proceedings of the 3rd International Workshop on Automated Specification and Verification of Web Systems (WWV 2007).
2. R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. *Theoretical Computer Science*, 360(1-3):386–414, 2006.
3. C. Chira, T. F. Serbanuta, and G. Stefanescu. P systems with control nuclei: The concept. *Journal of Logic and Algebraic Programming*, 79(6):326 – 333, 2010. Membrane computing and programming.
4. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.
5. G. Dowek, C. Muñoz, and C. Rocha. Rewriting logic semantics of a plan execution language. *Electronic Proceedings in Theoretical Computer Science*, 18:77–91, 2010.
6. T. Estlin, A. Jónsson, C. Păsăreanu, R. Simmons, K. Tso, and V. Verna. Plan Execution Interchange Language (PLEXIL). Technical Memorandum TM-2006-213483, NASA, 2006.

7. D. Lucanu. Strategy-based rewrite semantics for membrane systems preserves maximal concurrency of evolution rule actions. *Electronic Notes in Theoretical Computer Science*, 237:107 – 125, 2009. Proceedings of the 8th International Workshop on Reduction Strategies in Rewriting and Programming (WRS 2008).
8. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *WADT*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1997.
9. J. Meseguer and P. Ölveczky. Formalization and correctness of the pals architectural pattern for distributed real-time systems. In J. Dong and H. Zhu, editors, *Formal Methods and Software Engineering*, volume 6447 of *Lecture Notes in Computer Science*, pages 303–320. Springer Berlin / Heidelberg, 2010.
10. C. Rocha and J. Meseguer. Proving safety properties of rewrite theories. Technical report, University of Illinois at Urbana-Champaign, 2010. http://hdl.handle.net/2142/17407.
11. C. Rocha, C. Muñoz, and G. Dowek. A formal library of set relations and its application to synchronous languages. *Theoretical Computer Science*, 412(37):4853–4866, 2011.
12. T. Serbanuta. *A Rewriting Approach to Concurrent Programming Language Design and Semantics*. PhD thesis, University of Illinois at Urbana-Champaign, December 2010. https://www.ideals.illinois.edu/handle/2142/18252.
13. T. Serbanuta, G. Stefanescu, and G. Rosu. Defining and executing p systems with structured data in k. In D. Corne, P. Frisco, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 374–393. Springer Berlin / Heidelberg, 2009.
14. P. Viry. Equational rules for rewriting logic. *Theoretical Computer Science*, 285(2):487–517, 2002.