Timed rewriting logic with an application to object-based specification

Piotr Kosiuczenko\textsuperscript{a,b,*}, Martin Wirsing\textsuperscript{b}

\textsuperscript{a}Instytut Matematyki, Politechnika Warszawska, pl. Politechniki 1, Warszawa, Poland
\textsuperscript{b}Institut für Informatik, Ludwig-Maximilians-Universität München, Leopoldstr. 11 b, D-80802 München, Germany

Abstract

In this paper timed rewriting logic is presented and its application to the specification of real-time object-oriented systems is shown by an example.

Time rewriting logic (TRL) is an extension of Meseguer’s rewriting logic. The functional and the static properties of a system are described by algebraic specifications, whereas the behaviour of a process is described by nondeterministic term rewriting where each rewriting step is labelled by a time stamp or a time interval.

Thus our approach is similar to timed transition systems and can be seen as a generalization of timed automata combined with algebraic specifications. The approach is illustrated by several examples, such as clocks, time out and timer.

As the main application we present Timed Maude, an object-based specification language for real-time concurrent systems. Timed Maude is a timed variant of Meseguer’s language Maude which is based on rewriting logic. The algebraic specification part and the module part of Maude are kept unchanged in Timed Maude, only concurrent rewriting is replaced by TRL.

© 1997 Elsevier Science B.V.

Keywords: Real-time systems; Algebraic specification; Object-oriented systems; Term rewriting systems; Timed rewriting

1. Introduction

The main goal of timed rewriting logic is to extend algebraic specification techniques and tools to dynamic systems and in particular to real-time systems.

Algebraic specification techniques have proved to be useful and well-suited for describing complex data structures and the functional aspects of a software system (see e.g. [27, 28]). But lately it appeared that the existing algebraic specification
techniques are insufficient when applied to dynamic systems. Also the question appeared how to apply such powerful tools to dynamic systems or real-time systems.

There are many approaches that extend algebraic techniques to deal with dynamic systems ranging from operational ones to stream-processing functions and temporal logics [3]. Among the operational ones two of them seem to be particularly well-suited: Astesiano's SMoLCS approach based on algebraic transition systems [2] and Meseguer's concept of rewriting logic [15]. SMoLCS has been used for specifying and prototyping many different kinds of concurrent systems and rewriting logic has been applied for describing uniformly many different formalisms such as Petri Nets, Actors, CCS, and for designing the object-oriented parallel programming language Maude [16]. The latter is an object-oriented extension of OBJ [8]. In contrast to many other object-oriented languages it supports concurrency and multiple inheritance (see [30,18] for details).

In this paper we propose a timed extension of Rewriting Logic called Timed Rewriting Logic and use it for defining a timed variant of Maude called Timed Maude.

Timed Rewriting Logic extends algebraic specification techniques and allows one to reason about time elapse in real-time systems. This is done in the lines of studies considering processes as terms and proofs as behaviours of a process. We add timing constraints to rewrite rules for dealing with processes happening in real-time. Terms represent the states (or phases) of a system and timed rewrites model transitions in time. Every time-dependent rewrite step is labelled with a time stamp or a time interval. The basic rules of rewriting calculus are extended with time labels as follows:

- Transitivity yields the addition of the time elapses.
- Congruence and replacement are modelled by synchronous composition. This allows us to enforce uniform time elapse in all components of a system.
- Reflexivity is dropped since we are going to express also hard real-time constraints.

For specifications of soft real-time systems we can add particular reflexivity axioms.

Synchronous composition combined with irreflexivity induces maximal parallelism, which means that no component of a process can stay idle. As a consequence Timed Rewriting Logic allows one to describe the behaviour of dynamic systems in time, to reason about time elapse and to describe hard as well as soft real-time systems.

Timed Rewriting Logic gives a framework that generalizes timed automata [1] and timed transition systems [9]. In contrast, to both these approaches it includes algebraic specification techniques. The functional and the static properties of a system is described by algebraic specifications, whereas the dynamic behaviour of the system is modelled by transitions.

Since reflexivity is dropped our logic is not just a timed version of Meseguer's Rewriting Logic [15]. Reflexivity would not allow to describe hard real-time systems since it would not allow any enforcement of actions in a given time period. In other words, the system could stay idle for an arbitrary long time period.

Timed Rewriting Logic is also different from Timed CSP [23], Timed LOTOS [22] and Timed Process Algebra [4] since deliberately we do not abstract from states (see
also [21]). Moreover, in contrast to these approaches TRL focuses on true parallelism and not on interleaving semantics.

In this paper we study syntax and semantics of TRL and prove some basic properties concerning the existence of initial models and the decidability of finite timed rewriting systems.

As the main application we present Timed Maude, an object-oriented specification language for real-time systems. Timed Maude is a timed variant of Maude where the algebraic specification part and the module part of Maude are kept unchanged, only concurrent rewriting is replaced by TRL.

The paper is organized as follows.

In Section 2 the basic definitions of signature, algebra, term algebra, equational specification are given and some well-known facts are stated. The rules of Meseguer's rewriting logic are presented.

In Section 3 we add timing constraints to rewrite rules for dealing with processes happening in real-time. In Section 3.1 timed rewriting logic (TRL) and its deduction rules are introduced. The definition of TRL is based on the notion of an archimedean monoid. This notion summarises minimal requirements which must be satisfied by a monoid modelling time. A (labelled) timed rewrite rule has the form $t_1 \rightarrow g r \rightarrow t_2$ and means informally that the term $t_1$ rewrites to the term $t_2$ in time $r$ by applying the rule labelled with $g$. A timed rewrite specification extends an equational specification by a set of labelled timed rewrite rules. Several simple examples of timed rewriting specifications including timers and time outs are given in Section 3.2. In Section 3.4 a semantics of TRL based on the notion of functional dynamic algebra is presented. It corresponds to the semantics of rewriting logic given by Meseguer [15]. The soundness and completeness of TRL is shown. As for equational specifications completeness follows from the existence of initial models of TRL-specifications. In Section 3.5 we prove that for any finite timed rewrite specification with decidable equational theory, the validity of any TRL-formula is decidable.

In Section 4 we present first an example in a form that is used in current pragmatic object-oriented design methods (cf. e.g. [10]). Then we introduce shortly Timed Maude and give a Timed Maude specification of the example. Finally, in Section 5 we conclude the paper with some remarks on possible extensions of TRL.

2. Basic definitions

2.1. Signatures, structures and formulas

A many sorted (algebraic) signature $\Sigma$ is a pair $(S, F)$, where $S$ is a set of sorts and $F$ is a set of function symbols. Given $s_1, \ldots, s_n, s \in S$, to each function symbol $f$ a type $s_1, \ldots, s_n \rightarrow s$ is associated. $s$ is called the range of $f$. A many sorted relational signature $\Sigma$ is a triple $(S, F, P)$, where $(S, F)$ is an algebraic signature and $P$ is a set of relational symbols with associated arities of the form $(s_1, \ldots, s_n)$, where $s_1, \ldots, s_n \in S$. 
A (total) $\Sigma$-algebra $A = ((A_s)_{s \in S}, (f_A)_{f \in F})$ over a signature $\Sigma = (S, F)$ consists of a family of carrier sets $(A_s)_{s \in S}$, and a family of (total) functions $(f_A)_{f \in F}$ such that $f^A : A_{s_1} \times \ldots \times A_{s_n} \to A_s$ if $f$ has type $s_1, \ldots, s_n \to s$. A $\Sigma$-structure over a relational signature $\Sigma = (S, F, P)$ is a triple $A = ((A_s)_{s \in S}, (f_A)_{f \in F}, (p_A)_{p \in P})$, where $((A_s)_{s \in S}, (f_A)_{f \in F})$ is an $(S, F)$-algebra and where for any relational symbol $p \in P$ with arity $(s_1, \ldots, s_n)$ $p_A$ is a subset of the cartesian product: $A_{s_1} \times \ldots \times A_{s_n}$.

Given an algebraic signature $\Sigma = (S, F)$ and an arbitrary $S$-sorted family $X = (X_s)_{s \in S}$ of sets $X_s$, $T(\Sigma, X) = ((T(\Sigma, X)_s)_{s \in S}, (f)_{f \in F})$ denotes the $\Sigma$-term algebra freely generated by $X$. An element $t \in T(\Sigma, X)_s$ is called $\Sigma$-term of sort $s$ with variables in $X$. A term without variables is called ground term. We write $t(x_1, \ldots, x_n)$ for denoting a term $t$ which contains at most the variables $x_1, \ldots, x_n$. $t(t_1, \ldots, t_n)$ denotes the term obtained from $t(x_1, \ldots, x_n)$ by simultaneous substitution of $t_i$ for $x_i$. Let $\rho : X \to X$ be a family of renamings of variables $\rho_s : X_s \to X_s$ for $s \in S$. We can extend $\rho$ to (a family of) renamings of variables of terms (denoted by the same name) $\rho : T(\Sigma, X) \to T(\Sigma, X)$ defined by $\rho_s(t(x_1, \ldots, x_n)) = \rho(t_1(x_1), \ldots, \rho_s(x_n))$, where $x_i$ is of sort $s_i$ for $i = 1, \ldots, n$.

In the following, definitions and theorems are often formulated for one sorted algebras in order to avoid overloading the paper by technicalities. By adding appropriate indices for the sorts the definitions and theorems can easily be extended to the many sorted case.

If $A$ is a $\Sigma$-algebra then a valuation $v : X \to A$ is a family of mappings $v_s : X_s \to A_s$. For any $\Sigma$-term $t$, the corresponding interpretation function or term function $t_A : (X \to A) \to A$ is defined inductively as follows: Let $v : X \to A$ be any valuation.

1. If $t$ is a variable $x$, then $x_A(v) = \text{def } v(x)$.
2. If $t$ is of the form $f(t_1, \ldots, t_n)$, then $t_A(v) = \text{def } f_{t_A}(t_1_A(v), \ldots, t_n_A(v))$.

The $\Sigma$-algebra structure of term functions $\text{Ter}_X(A)$ over $A$ and $X$ consists of

1. the carrier sets $(\text{Ter}_X(A)_s)_{s \in S}$ defined by $\text{Ter}_X(A)_s = \text{def } \{ t_A : t \in T(\Sigma, X)_s \}$,
2. the operations $(f_{\text{Ter}_X(A)}(t_1, \ldots, t_n))(v) = \text{def } f(t_1_A(v), \ldots, t_n_A(v))$.

An atomic $\Sigma$-formula is either an equation $t_1 = t_2$ or a literal $p(t_1, \ldots, t_n)$ with $t_1, \ldots, t_n \in T(\Sigma, X)$. A $\Sigma$-algebra $A$ satisfies $t_1 = t_2$ (in symbols $A \models t_1 = t_2$) iff $t^A_1(v) = t^A_2(v)$ for all valuations $v : X \to A$ iff $t^A_1 = t^A_2$. Similarly for any relational symbol $p \in P$, $A \models p(t_1, \ldots, t_n)$ iff $(t_1^A(v), \ldots, t_n^A(v)) \in p^A$ for all valuations $v : X \to A$.

The following is a well-known fact.

Fact 2.1.1. Let $X$ be an infinite set and let $\phi$ be an equation. Then

1. $A \models \phi$ iff $\text{Ter}_X(A) \models \phi$.
2. The algebra of term functions $\text{Ter}_X(A)$ is isomorphic to $\text{Ter}_X(\text{Ter}_X(A))$.

2.2. Equational specifications and rewriting logic

An equational specification is a pair $(\Sigma, E)$ consisting of a signature $\Sigma$ and a set $E$ of $\Sigma$-equations.
The following is a version of the Birkhoff calculus [5], which is sound and complete.

0. **Reflexivity**: For each \( t \in T(\Sigma, X) \),

\[
\overline{t} = t
\]

1. **Transitivity**: For each \( t_1, t_2, t_3 \in T(\Sigma, X) \)

\[
\frac{t_1 = t_2, \quad t_2 = t_3}{t_1 = t_3}
\]

2. **Symmetry**: For each \( t_1, t_2 \in T(\Sigma, X) \)

\[
\frac{t_1 = t_2}{t_2 = t_1}
\]

3. **Replacement**: For each \( t_0, t_1, \ldots, t_n, u_0, u_1, \ldots, u_n \in T(\Sigma, X) \)

\[
\frac{t_0 = u_0, \quad t_1 = u_1, \ldots, t_n = u_n}{\overline{t_0(t_1, \ldots, t_n)} = \overline{u_0(u_1, \ldots, u_n)}}
\]

Given an equational specification \((\Sigma, E)\) and a (possibly) infinite set \( L \) of labels, a (labelled) rewrite rule is a literal \( p_g(t_1, t_2) \) written as \( t_1 \rightarrow g \rightarrow t_2 \), where \( g \) is a label from \( L \) and where \( t_1, t_2 \) are \( \Sigma \)-terms in \( T(\Sigma, X) \) of the same sort.\(^2\) Informally, this means that \( t_1 \) rewrites to \( t_2 \) by applying the rule labelled with \( g \). The label \( g \) can be understood as the name of the rewrite rule or – as it is customary in labelled transition systems – as (the name of) an action.

A rewrite specification extends \((\Sigma, E)\) with a set of labelled rewrite rules. Thus, a (labelled) rewrite specification is a presentation of a usual theory with equations and literals as axioms where the literals are rewrite rules indexed by labels. Formally, a (labelled) rewrite specification \( R \) is a 4-tuple \( R = (\Sigma, E, L, RW) \), where \( \Sigma \) is a signature, \( E \) is a set of \( \Sigma \)-equations and \( RW \) is a set of literals (contained in the family of literals \( (p_g(t_1, t_2))_{g \in L} \)). For expressing deductions we introduce rewrite rules extending the set of labels inductively. Formally, we define the set \( G \) of composite labels inductively as follows:

\[
\begin{align*}
G_0 &= \text{def} L \cup \{\text{id}\}, \\
G_{n+1} &= \text{def} G_n \cup \{g_1; g_2 : g_1, g_2 \in G_n\} \\
&\quad \cup \{l(g_1, \ldots, g_m) : l \in L \text{ and } g_i \in G_n \text{ for } i = 1, \ldots, m\}, \\
G &= \text{def} \cup G_n.
\end{align*}
\]

A label \( g \in G \) is called **atomic** iff \( g \in L \cup \{\text{id}\} \).

\( \text{id} \) is a special label, which will appear in reflexivity axioms.

---

\(^2\)Meseguer uses the notation \( g : t_1 \rightarrow t_2 \). We rather follow the convention of labelled transition systems where an action is written in infix notation.
The following deduction system for rewriting logic is equivalent to Meseguer's rewriting logic. A rewrite specification \( \mathcal{R} \) entails a literal \( t_1 \xrightarrow{g} t_2 \) (written \( \mathcal{R} \vdash t_1 \xrightarrow{g} t_2 \)) if and only if \( t_1 \xrightarrow{g} t_2 \) can be obtained from the axioms \( E \) and \( RW \) by using the axioms and rules of the Birkhoff calculus and the following four deduction rules for rewriting:

0. **Reflexivity (Ref)**: For each \( t \in T(\Sigma, X) \),

\[
\text{id} \xrightarrow{t} t
\]

1. **Transitivity (T)**: For each \( t_1, t_2, t_3 \in T(\Sigma, X), g_1, g_2 \in G \)

\[
\frac{t_1 \xrightarrow{g_1} t_2, t_2 \xrightarrow{g_2} t_3}{t_1 \xrightarrow{g_1; g_2} t_3}
\]

2. **Replacement (Rpl)**: For each \( t_0, t_1, \ldots, t_n, u_0, u_1, \ldots, u_n \in T(\Sigma, X), l \in G_0, g_1, \ldots, g_n \in G \)

\[
\frac{t_0 \xrightarrow{l} u_0, t_1 \xrightarrow{g_1} u_1, \ldots, t_n \xrightarrow{g_n} u_n}{t_0(t_1, \ldots, t_n) \xrightarrow{l(g_1, \ldots, g_n)} u_0(u_1, \ldots, u_n)}
\]

3. **Compatibility with = (Comp)**: For each \( t_1, t_2, u_1, u_2 \in T(\Sigma, X), g \in G \)

\[
\frac{t_1 = u_1, u_1 \xrightarrow{g} u_2, u_2 = t_2}{t_1 \xrightarrow{g} t_2}
\]

We say that \( \mathcal{R} \) entails \( t_1 \xrightarrow{g} t_2 \) if and only if there exists a condition \( g \in G \) such that \( \mathcal{R} \vdash t_1 \xrightarrow{g} t_2 \).

The first three rules 0, 1, 2 are equivalent to the classical rules for reflexivity, transitivity, congruence and substitution. The congruence rule can be obtained from the replacement rule by using the reflexivity axiom \( f(x) \xrightarrow{id} f(x) \) for the function symbol \( f \). Substitution is also a specialization of the replacement rule by choosing reflexivity rules \( t_i \xrightarrow{id} t_i \) for the substitutions \([t_i/x_i] \). On the other hand, the replacement rule can be obtained by an iterated combination of substitution, congruence and transitivity. Rule 3 ensures the compatibility of the rewriting relations with the equality relation.

The replacement rule above is a slightly generalized version of the replacement rule of Meseguer. Like Meseguer's rule, it is particularly well-suited to describe the dynamic behaviour of systems which evolve concurrently. Concurrent rewriting coincides with deduction.

A rewrite specification extends \((\Sigma, E)\) with a set of labelled rewrite rules. Thus, a (labelled) rewrite specification is a presentation of a usual theory with equations and literals as axioms where the literals are rewrite rules indexed by labels. Formally,
a (labelled) rewrite specification $\mathcal{R}$ is a 4-tuple $\mathcal{R} = (\Sigma, E, L, RW)$, where $\Sigma$ is a signature, $E$ is a set of $\Sigma$-equations and $RW$ is a set of literals (contained in the family of literals $(p_g(t_1, t_2))_{g \in L}$).

3. Timed rewriting logic and its rules

Time is modelled abstractly by archimedean monoids:

Let $R_+ = (R_+, +, 0, \geq)$ be a monoid with a partial ordering relation $\geq$ such that 0 is the least element.

(1) $R_+$ is called archimedean monoid iff $+$ is a monotone operation (see [5]) and for every non-zero element $r_1$ of $R_+$ and for every element $r_2$ of $R_+$, $nr_1 > r_2$ holds for some natural number $n$ (where $nr_1 = r_1 + \cdots + r_1$).

(2) A sequence $\{r_i\}_{i \in N}$ diverges to infinity iff for every $r \in R_+$ there is a natural number $n$ such that $r_n > r$.

(3) Let $R_+$ be countable. $R_+$ is decidable, iff the operation $+$ is (total) recursive and the relation $\geq$ is decidable.

The archimedean property is needed in order to exclude the so-called Zeno paradox. We do not require the time domain to be linearly ordered since we have in mind also systems with distributed clocks where time is modelled by vectors of time values (see for example [25]). For other abstract models of time see for example [12, 21].

The definition of archimedean monoid covers discrete and dense time since the structure of natural numbers and the structure of all non-negative rational numbers with addition are both archimedean monoids. These two monoids are decidable. As an example of an archimedean monoid let us consider the following specification of

**Natural numbers.** Consider an algebra $N = (N, s, +, 0)$, where $s : N \to N$, $+ : N \times N \to N$, and where 0 is a constant. $N$ is the standard model of arithmetic of natural numbers iff $N$ is initial in the class of all algebras satisfying the following axioms: $x + y = y + x$, $0 + x = x$, $s(x) + y = s(x + y)$.

To specify the relation $\geq$ between natural numbers we assume a carrier set $B$ of boolean values to be given and that true and false are different boolean constants. Then the operation $\geq : N \times N \to B$ can be axiomatized in the following way:

$$(x \geq 0) = \text{true}, (s(x) \geq s(y)) = (x \geq y), (0 \geq s(x)) = \text{false}.$$  

(We write $x \geq y$ instead of $\geq (x, y)$.)

In the following we fix a particular archimedean monoid $R_+$ and assume an equational axiomatization $SP_{\text{Time}} = (\Sigma_{\text{Time}}, E_{\text{Time}})$ of $R_+$ to be given. The signature $\Sigma_{\text{Time}}$ includes the signature $\Sigma_{\text{AM}} = \text{def}([\text{Time}, \text{Bool}], \{0, +, \geq\})$ of archimedean monoids. $R_+$ is the carrier set of the sort Time and $\geq$ is a boolean function symbol corresponding to the relation $\geq$. Equational axiomatizations of $\Sigma_{\text{AM}}$ exist, e.g., for the structures of natural numbers. For other archimedean monoid structures, one can
always construct an infinitary equational specification as follows: we extend the signature of archimedean monoids by adding a constant symbol \( r \) for each \( r \in R_+ \) (where \( r \) denotes \( r \)). Then \( E_{\text{Time}} \) is the diagram of \( R_+ \):

\[
E_{\text{Time}} = \{ t_1 = t_2 : t_1, t_2 \in T(\Sigma_{\text{AM}}, \{ r : r \in R_+ \}) \text{ and } t_1 = t_2 \text{ holds in } R_+ \}
\]

Any ground \( \Sigma_{\text{Time}} \)-term \( t \) is equal to a constant \( r \), where \( r \in R_+ \). Thus in the following, we identify the elements of \( R_+ \) with ground \( \Sigma_{\text{Time}} \)-terms and write \( r \) for \( r \).

Moreover, we assume that a specification \( SP(R_+) = (\Sigma(R_+), E(R_+)) \), of an application domain is given which extends \( SP_{\text{Time}} \). The signature \( \Sigma(R_+) \) consists of the union of the signature \( \Sigma_{\text{Time}} \) with the sorts \( S_0 \) and the function symbols \( F_0 \) of the application domain.

The set of equational axiom \( E(R_+) \) consists of the union of the axioms \( E_{\text{Time}} \) with the axioms \( E_0 \) of the application domain:

\[
(\Sigma(R_+), E(R_+)) = \{ (S_0, F_0) \cup \Sigma_{\text{Time}}, E_0 \cup \Sigma_{\text{Time}} \}.
\]

In some applications, it is necessary to consider a special function symbol “age”. This symbol can be introduced for each sort \( s \in S_0 \) so that \( \Sigma(R_+) \) contains the symbol

\[
\text{age: } s, \text{ Time } \to s
\]

where \( \text{age}(t, r) \) informally expresses that the term \( t \) has aged by \( r \) time units. The function \( \text{age} \) is axiomatized with the equation \( \text{age}(\text{age}(t, r_1), r_2) = \text{age}(t, r_1 + r_2) \) and timed rewrite rules. Then \( F_0 \) contains the age symbol for every \( s \in S_0 \).

The set of labels of timed rewriting logic is defined in the following way (cf. [15]):

\[
G_0 = \text{def } L,
\]

\[
G_{n+1} = \text{def } G_n \cup \{ g_1, g_2 : g_1, g_2 \in G_n \}
\]

\[
\cup \{ l(g_1, \ldots, g_m) : l \in L, g_i \in G_n \text{ for } i = 1, \ldots, m \},
\]

\[
G = \text{def } \bigcup G_n.
\]

A (labelled) timed rewrite rule is a literal \( p_s(t_1, r, t_2) \) written as \( t_1 - g \ r \to t_2 \), where \( r \in R_+ \), \( g \) is a label from \( G \) and \( t_1, t_2 \) are \( \Sigma \)-terms in \( T(\Sigma(R_+), X) \) of the same sort. Informally, this means that \( t_1 \) rewrites to \( t_2 \) in time \( r \) by applying the rule labelled with \( g \). As before the label \( g \) can be understood as the name of the rewrite rule or as (the name of) an action.

A timed TRL-rewrite specification extends \( (\Sigma(R_+), E(R_+)) \) with a set of labelled timed rewrite rules. Thus, a (labelled) timed rewrite specification is a presentation of a usual theory with equations and literals as axioms where the literals are timed rewrite rules indexed by labels. Formally, a TRL-specification \( \mathcal{T}R \) (also called (labelled) timed rewrite specification) is a 4-tuple

\[
\mathcal{T}R = (\Sigma(R_+), E(R_+), L, RW),
\]
where $\Sigma(R_+)$ is a signature containing $\Sigma_{\text{Time}}$, $E(R_+)$ is a set of $\Sigma$-equations containing $E_{\text{Time}}$ and $\text{RW}$ is a subset of family of literals $\mathcal{L}^e = \{p_1(t_1, r, t_2); \ t_1, t_2 \in T(\Sigma, X), r \in R_+\}_{i \in L}$.

3.1. Basic rules of TRL

The basic rules of rewriting calculus (see e.g. [15]) are extended with time stamps as follows:

- **Transitivity** yields the addition of the time elapses. If $t_1$ evolves in time $r_1$ to $t_2$ and $t_2$ evolves in time $r_2$ to $t_3$ then $t_1$ evolves in time $r_1 + r_2$ to $t_3$.
- **Replacement** is modelled by synchronous replacement: Let $t_0(t_1, \ldots, t_n)$ and $u_0(u_1, \ldots, u_n)$ be composite terms and let $x_{i_1}, \ldots, x_{i_k}$ be the intersection of the (flexible) variables of $t_0$ and $u_0$. A composite term $t_0(t_1, \ldots, t_n)$ evolves in time $r$ to the term $u_0(u_1, \ldots, u_n)$ if all its components do this in time $r$, that is if $t_0$ evolves to $u_0$ and if $t_j$ evolves to $u_j$ for $j = i_1, \ldots, i_k$. We do not require anything for $t_j$ or $u_j$ with $j \neq i_1, \ldots, i_k$ since the corresponding variables occur only in one of the terms $t_0$ or $u_0$. This rule allows us to enforce uniform time elapse in all components of a system.

The uniform time elapse is a major requirement (and obstacle) in designing TRL. An important feature of TRL is the philosophical assumption of an absolute time (but not of a global clock which would synchronize all processes), which allows to reason about change in time. Synchronous replacement combined with irreflexivity induces maximal parallelism, which means that no component of a process can stay idle.
- **Timed compatibility** is just the compatibility of the equality relation with the ternary timed rewriting relations.
- **Renaming of variables** is a rule which ensures that timed rewriting is independent of the particular names of the variables.

1. **Timed transitivity** (TT). For each $t_1, t_2, t_3 \in T(\Sigma, X), g_1, g_2 \in G, r_1, r_2 \in R_+$

$$
\begin{align*}
& t_1 - g_1 r_1 \rightarrow t_2, t_2 - g_2 r_2 \rightarrow t_3 \\
& t_1 - g_1, g_2 r_1 + r_2 \rightarrow t_3
\end{align*}
$$

2. **Synchronous replacement** (SR). Let $\{x_{i_1}, \ldots, x_{i_k}\} = \text{FV}(t_0) \cap \text{FV}(u_0)$ be the intersection of the free variables of $t_0$ and $u_0$. For each $t_0, t_1, \ldots, t_n, u_0, u_1, \ldots, u_n \in T(\Sigma, X), l \in L, g_{i_1}, \ldots, g_{i_k} \in G, r \in R_+$

$$
\begin{align*}
& t_0 - l r \rightarrow u_0, \ t_{i_1}, \ldots, g_{i_l} r \rightarrow u_{i_1}, \ldots, t_{i_k} - g_{i_k} r \rightarrow u_{i_k} \\
& t_0(t_1, \ldots, t_n) - l(g_{i_1}, \ldots, g_{i_k}) r \rightarrow u_0(u_1, \ldots, u_n)
\end{align*}
$$

3. **Timed compatibility** with $= (TC)$. For each $t_1, t_2, u_1, u_2 \in T(\Sigma, X), r_1, r_2 \in R_+, g \in G$

$$
\begin{align*}
& t_1 = u_1, r_1 = r_2, u_1 - g r_1 \rightarrow u_2, u_2 = t_2 \\
& t_1 - g r_2 \rightarrow t_2
\end{align*}
$$
4. Renaming of variables (RN). Let $\rho : X \to X$ be a renaming of variables. For each $t_1, t_2 \in T(\Sigma, X)$, $g \in G$, $r \in R_+$

$$\frac{t_1 - gr \to t_2}{\rho(t_1) - g r \to \rho(t_2)}$$

A timed rewrite specification $\mathcal{T} = (\Sigma(R_+), E(R_+), L, RW)$ entails a literal $t_1 - gr \to t_2$ (written $\mathcal{T} \models t_1 - gr \to t_2$) if and only if $t_1 - gr \to t_2$ can be obtained from the axioms $E(R_+)$ and RW by using the axioms and rules of equational logic (e.g. of the Birkhoff calculus [5]) and the deduction rules 1-4 above for times rewriting.

3.2. Examples

3.2.1. Pedestrian lights

Consider pedestrian lights where the lights can be only in two states: red and green. The light switches from red to green after 1 min and from green to red after 2 min. We can specify this situation in the following way:

The signature $(S_0, F_0)$ consists of one sort "State" with two 0-ary function symbols red, green: $\to$ State, $R_+$ is the set of natural numbers. We introduce two labels $r_g$ and $g_r$ with the axioms

$$\text{red} - r_g 1 \to \text{green}, \quad \text{green} - g_r 2 \to \text{red}.$$

for specifying the admissible state changes from red to green and from green to red.

3.2.2. Clock

A clock changes dynamically with the elapse of time. Thus, we define a clock as a unary function symbol clock from sort $Time$ to the sort Clockstate.

$$\text{clock} : Time \to \text{Clockstate}$$

with the set of axioms \{clock$(r) - c r_1 \to \text{clock}(r + r_1)$: $r, r_1 \in R_+$\}

3.2.3. Lights with a clock

The lights can be combined in parallel with a clock as follows: We introduce a sort "Conf" for the combined states of lights and clocks and a binary function symbol

$$\mid : \text{State} \times \text{Clockstate} \to \text{Conf}$$

with the axiom

$$x | y \mid \text{par} r \to x | y.$$ 

As an example application we derive the following one-step concurrent rewrite

$$\text{par}(r_g, c):
\begin{align*}
\frac{x | y \mid \text{par} 1 \to x | y, \text{red} - r_g 1 \to \text{green}, \text{clock}(5) - c 1 \to \text{clock}(5 + 1)}{
\text{red} | \text{clock}(5) - \text{par}(r_g, c) 1 \to \text{green} | \text{clock}(5 + 1)}
\end{align*}$$
3.2.4. Time out

Let $\mathcal{R} = (\Sigma(R_+), E(R_+), L, \Lambda x)$ be a timed rewrite specification where $E(R_+) = E_{\text{time}}$ and let us suppose that $p \in L$ is the expected distinguished atomic action to be done on a given state $t$ of sort $s$. Furthermore, assume that this action should be done in less than $r_0$ seconds and should change the state $t$ to the state $t'$. If this does not happen, then the system should change to the state $t_1$ of sort $s$.

To model this situation we enrich our term signature $\Sigma(R_+)$ adding new ternary operation symbols $\text{TO} : s, s, \text{Time} \to s$ (for each $s \in S_0$) and a new action symbol $d$. The set $\Lambda x$ is extended by adding the set of axioms $\{ \text{TO}(t, t_1, r) - pr' \to t' : t - pr' \to t' \in \Lambda x \text{ and } r \geq r' \}$ for the label $p$ and the following sets of axioms describing the time out:

\[
\{ \text{TO}(t, t_1, r) - dr' \to \text{TO}(t, t_1, r'') : r' + r'' = r \},
\]

\[
\text{TO}(t, t_1, 0) - d0 \to t_1 \}.
\]

3.2.5. Timer

Timers can be used to control the time elapse in a system or to delay processes.

We define a timer by a unary function symbol from sort $\text{Time}$ to a new sort “Timer”, within the TRL framework. In contrast to a clock a timer counts downwards which leads to the following axiomatization:

\[
\{ \text{timer}(r_1 + r) - \rho r \to \text{timer}(r_1) : r, r_1 \in R_+ \}.
\]

Observe, that $\text{timer}(0)$ cannot be rewritten by a positive time delay $r > 0$.

3.3. An extension of TRL

If the execution times of the rules for the components of a system are different then the synchronous replacement rule cannot be applied directly. We achieve the synchronization of the timing by the introduction of the “age” function. The following aging rule allows to split every action into two components: the first is time elapse, the second is the proper action. It corresponds to the paradigm accepted by some researchers that actions take no time and that there is a special time-elapse action (see for example [9]).

5. Aging rules (Age). For each $t_1, t_2, u_1, u_2 \in T(\Sigma, X)$, $r_1, r_2 \in R_+$, $l \in L$

\[
\begin{align*}
\tag{a} r_1 + r_2 = r, t_1 - lr \to t_2 & \quad \Rightarrow \quad t_1 - \text{age}(t_1, r_1) \\
\tag{b} r_1 + r_2 = r, t_1 - lr \to t_2 & \quad \Rightarrow \quad \text{age}(t_1, r_1) - lr_2 - t_2
\end{align*}
\]

The term $\text{age}(t_1, r_1)$ in the rule above specifies that $r_1$ time units have elapsed. The rules 5(a) and 5(b) can be understood as expressing waiting for a synchronization event $l$. 
3.4. Semantics and properties of TRL

In this section we present the notions of functional dynamic algebra. One can say that functional dynamic algebra is a model for TRL describing a behaviour "locally".

In a functional dynamic algebra, terms (possibly with variables) are interpreted by term functions (see Section 2.1). The term functions correspond exactly to the equational classes of terms considered by Meseguer (see [15]) since any term function can be viewed as a congruence class of terms, i.e. an equational class of term functions of a given algebra. Timed rewriting steps are interpreted by ternary relations over term functions. Thus, a functional dynamic algebra is a term functions algebra $\text{Ter}_x(A)$ over a given algebra $A$ together with interpretations for the ternary relation symbols $P_g$ with $g \in G$.

For defining the truth of a formula in a functional dynamic algebra we first give interpretations for timed rewrite relations $P_g$ with labels $g \in G$. The following definition introduces a hierarchy of relations, which is further used in the definition of functional dynamic algebra.

**Definition 3.4.1.** Let $\Sigma = (S, F)$ be an algebraic signature containing $\Sigma_{\text{Time}}$, $X$ an $S$-sorted family of infinite sets of variables, $L$ be a set of labels, $P = \{P_g: g \in G\}$ the corresponding set of ternary relation symbols and let $A$ be a $\Sigma$-algebra. We assume that age does not belong to $L$. For every $g \in G$ we define inductively the relation $\gamma_g$ which is the interpretation of $P_g$. 

1. Let for $l \in L (= G_0)$, $\gamma_l \subseteq \text{Ter}_x(A) \times R \times \text{Ter}_x(A)$ be an arbitrary relation closed under renaming of variables.
2. For any composite $g \in G$ the relation $\gamma_g \subseteq \text{Ter}_x(A) \times R \times \text{Ter}_x(A)$ is defined as follows:

   $$\gamma_{g_1; g_2} = \{ (t_1, r, t_2^3): \text{there exist } t_2, r_1, r_2 \text{ such that}\}
   \begin{align*}
   & (t_1^1, r_1, t_2^1) \in \gamma_{g_1} \quad \& \quad (t_2^2, r_2, t_3^2) \in \gamma_{g_2} \\
   \text{and } r_1 + r_2 = r\}.
   \end{align*}

   $$

   $$\gamma_{(g_1, \ldots, g_k)} = \{ (t_0^A(t_1^A, \ldots, t_n^A), r, u_0^A(u_1^A, \ldots, u_k^A)); (t_0^A, r, u_0^A) \in \gamma_{g_0} \text{ for } g_0 = l \text{ and for } i = 0, i_1, \ldots, i_k,\}
   \text{where } \{i_1, \ldots, i_k\} \text{ is the intersection of the free variable sets of } t_0 \text{ and } u_0.
   $$

3. $\Gamma_n(X) = \{ \gamma_g: g \in G_n \}$ is the set of relations associated with the set of labels $G_n$ for each $n \in N$.

4. $\Gamma(X) = \bigcup \Gamma_n(X)$

   We often write $\Gamma$ instead of $\Gamma(X)$, when it does not cause any ambiguity. If the symbol age occurs, then we can extend the above definition in a natural way using a fixpoint construction.
Definition 3.4.2. Let $\Sigma$, $A$, $\Gamma_0$ and $\Gamma$ be as above and let $t_0, \ldots, t_n, u_0, \ldots, u_n, r, r_1, r_2, r \in R^+$.  
(1) A functional dynamic $(S, F, L)$-algebra over $A$ is a relation $(S, F, L)$-structure $(\text{Ter}_X(A), \Gamma_0)$  
(2) A fractional dynamic algebra (DA) is a functional dynamic algebra over some algebra.  
(3) If $A$ is a term algebra $\mathcal{T}(\Sigma, X)$, then we call such a pair a timed term rewriting system (TTRS).  
(In this case it is the pair $(\mathcal{T}(\Sigma, X), \Gamma_0)$. Moreover, $\text{Ter}_X(A) = \text{Ter}_X(\mathcal{T}(\Sigma, X))$ is isomorphic to $\mathcal{T}(\Sigma, X)$, and we can identify the term functions with terms.)  
(4) A literal $t_1 - gr + t_2$ is true in $(\text{Ter}_X(A), \Gamma)$ (written $(\text{Ter}_X(A), \Gamma) \models t_1 - gr + t_2$) iff $(t_1^A, t_2^A) \in \gamma_g$, i.e. the relation $\gamma_g$ corresponding to $g$ contains $(t_1^A, t_2^A)$.  
To simplify notation we will often write $(A, \Gamma_0)$ instead of $(\text{Ter}_X(A), \Gamma_0)$.  

Theorem 3.4.3. The rules of TRL and the rules for the special symbol are sound with respect to the above semantics.  

Proof. Follows directly from the definition above by structural induction on the form of the labels. $\square$  

Morphisms of functional dynamic algebras are homomorphisms of relational structures:  

Definition 3.4.4. Let $(A, \Gamma_0)$ and $(B, \Gamma_0')$ be functional dynamic algebras. A morphism from $(A, \Gamma_0)$ and $(B, \Gamma_0')$ is a homomorphism $h : \text{Ter}_X(A) \rightarrow \text{Ter}_X(B)$ such that  

$$h(t^A) = t^B \text{ and } \{(h(t^A), r, h(t^A)); (t^A, r, t^A) \in \gamma_g \} \subseteq \gamma'_g \text{ for each } g \in G.$$  

In other words, a morphism must preserve the algebraic as well the relational structure. It can be seen as a special case of the notion of simulation (see for example [13, 29]).  

In algebraic specification theory initial and free models play a fundamental role. Due to the notion of morphism we can construct initial models of timed rewrite specifications.  

Given $\mathcal{R} = (\Sigma(R^+), E(R^+), L, \text{RW})$ and an $S$-sorted set $X$ of variables. Let us define the initial functional dynamic algebra as follows: $\mathcal{F} = \text{def} (\text{Ter}_X(F), \Gamma_0)$, where $F = \text{def} F(X)$ is the free algebra for the class $\text{Alg}(\Sigma(R^+), E(R^+))$ of all models of $(\Sigma(R^+), E(R^+))$ over $X$ and where for each $l \in L$ $\gamma_l$ is defined by  

$$\gamma_l = \text{def} \{(t^F, r, t^F); t_1 - lr \rightarrow t_2 \in \text{RW}\}.$$  

Theorem 3.4.5. Let $\mathcal{R} = (\Sigma(R^+), E(R^+), L, \text{RW})$ be a timed rewrite specification and $\mathcal{F}$ the functional dynamic algebra as defined above. Then $\mathcal{F}$ is an initial
model of TR, i.e.
(1) $F$ is a model of TR and
(2) for any model $(\text{Ter}_x(A), I'_0)$ of TR there exists a unique morphism from $F$ to $(\text{Ter}_x(A), I'_0)$

**Proof.** (1) By definition the free algebra $F$ satisfies the axioms $E(R_+)$. The same holds for $I'_0$ and RW. Thus, $F = (\text{Ter}_x(F), I'_0)$ is a model of TR.

(2) Let $(\text{Ter}_x(A), \Gamma'(X))$ be an arbitrary model of TR.

As mentioned in Fact 2.1.1 $\text{mx}(F)$ is isomorphic with $\gamma'$. It is an elementary fact from universal algebra, that there is exactly one homomorphism $h: F \rightarrow \text{Ter}_x(A)$ such that $h(t^F) = t^d$ for all terms $t$. This is a morphism (see Definition 3.4.4). Indeed:

$$h(p) = \{h(tf), r, h(t') : t_1 \rightarrow lr \rightarrow t_2 \in RW\}$$

$$= \{(t_1^d, r^d, t_2^d) : t_1 \rightarrow lr \rightarrow t_2 \in RW\} \subseteq \gamma'_f,$$ since $(A, \Gamma'(X)) \models RW$.

The rest follows from the fact that the composition of relations is monotoneic (see Definition 3.4.1) and the fact that $h(\gamma_g) = \{h(t^F), r, h(t^F) : (t_1^F, r, t^F_2) \in \gamma_g\}$.

Thus there exists a unique morphism from $F$ to $(\text{Ter}_x(A), \Gamma'(X))$. \(\square\)

**Corollary 3.4.6.** Let $\mathcal{RA}$ and $F$ be as in Theorem 3.4.5. Then for all equations $t_1 = t_2$ and all literals $t_1 - g r \rightarrow t_2$ with $t_1, t_2 \in T(\Sigma, X), r \in R_+, g \in G$ the following holds:

(1) $F \models t_1 = t_2$ if and only if $\mathcal{RA} \models t_1 = t_2$,

(2) $F \models t_1 - g r \rightarrow t_2$ if and only if $\mathcal{RA} \models t_1 - g r \rightarrow t_2$.

**Proof.** (1) According to Fact 2.1.1 an equation is true in a functional dynamic algebra $F$ if it is true in the corresponding free algebra $F$. An equation is true in a free algebra $F$ iff it semantically follows from the corresponding set of equations (see for example [5]).

The proof of (2) follows from Theorem 3.4.5 (2) which asserts that for any model $(\text{Ter}_x(A), \Gamma'(X))$ of TR there exists a unique morphism from $F$ to $(\text{Ter}_x(A), \Gamma'(X))$. \(\square\)

**Corollary 3.4.7.** (Soundness and completeness of TRL) Let $\mathcal{RA} = (\Sigma(R_+), E(R_+), L, RW)$ be a timed rewrite specification. Then for all equations $t_1 = t_2$ and all literals $t_1 - g r \rightarrow t_2$ with $t_1, t_2 \in T(\Sigma, X), r \in R_+, g \in G$ the following holds:

(1) $\mathcal{RA} \models t_1 = t_2$ if and only if $\mathcal{RA} \vdash t_1 = t_2$,

(2) $\mathcal{RA} \models t_1 - g r \rightarrow t_2$ if and only if $\mathcal{RA} \vdash t_1 - g r \rightarrow t_2$.

**Proof.** (1) It is a well-known fact that $E(R_+) \models t_1 = t_2$ iff $E(R_+) \vdash t_1 = t_2$, but this is equivalent to $\mathcal{RA} \vdash t_1 = t_2$ iff $E(R_+) \vdash t_1 = t_2$, but this is equivalent to $\mathcal{RA} \vdash t_1 = t_2$ iff $\mathcal{RA} \vdash t_1 = t_2$. The rest follows from the fact that the composition of relations is monotoneic (see Definition 3.4.1) and the fact that $h(\gamma_g) = \{h(t^F), r, h(t^F) : (t_1^F, r, t^F_2) \in \gamma_g\}$.

Thus there exists a unique morphism from $F$ to $(\text{Ter}_x(A), \Gamma'(X))$. \(\square\)
(2) The 'if' part follows from Theorem 3.4.3. For the "only if" part we consider the initial functional dynamic algebra \( F = (\text{Ter}_b(F), \Gamma_0) \). Moreover, by a simple induction one can prove that for all \( g \in G \),

\[
(t^1_1, r, t^2_2) \in \gamma_g \text{ if and only if } F \vdash t_1 - g r \rightarrow t_2
\]

which implies (2). \( \square \)

The statements (1) and (2) of the theorem above express the soundness and completeness of TRL. Note that due to the use of term functions satisfaction of literals in the initial model \( F \) is equivalent with validity in \( F \) (which is in contrast to initial models of equational theories where this result holds only for ground literals).

3.5. A decidability result

In this section we study the question whether a formula \( \phi \) of the form \( t - g r \rightarrow t' \) is valid in an initial model \( F \). We show that under reasonable local finiteness assumptions on the set of axioms the validity of such formulas is decidable.

Let \( E(R_+) \) be a decidable equational theory. Since the corresponding signature \( \Sigma(R_+) \) is by definition supposed to be finite or countable we can introduce a Gödel numbering of the set of terms \( T(\Sigma(R_+), X) \). The equivalence relation corresponding to equational theory of \( (\Sigma(R_+), E(R_+)) \) is decidable, therefore we can recursively define a normal form \( \text{nf}(t) \) of a term by taking as representative of an equivalence class the term which has the smallest Gödel number.

**Theorem 3.5.1.** Let \( F = (\Sigma(R_+), E(R_+), L, RW) \) be a timed rewrite specification such that the equational theory of \( (\Sigma(R_+), E(R_+)) \) is decidable and \( L \) is a finite set. Assume that for every \( a \in R_+ \) and every \( t \in T(\Sigma(R_+)) \) the synchronization set

\[
S(t, a, l) = \text{def} \{ (r, \text{nf}(t_j)): t_1 - l r \rightarrow t_2 \in RW: a \geq r \text{ and } E(R_+) \vdash t = t_1 \}
\]

is finite and recursively given depending on \( t \) and \( a \).

Then for every formula \( \phi \) of the form \( t - g r \rightarrow t' \) it is decidable whether \( \phi \) is valid in \( F \).

**Proof.** We prove the theorem in a slightly stronger form (*):

For every \( g \in G \), for every \( a \in R_+ \) and for every term \( t \) there is a recursively given finite formal representation set \( B(t, a, g) = \text{def} \{ (\text{nf}(t_j), r_j): j = 1, \ldots, n \} \), such that if \( F \vdash t - g r \rightarrow t' \) and \( a \geq r \), then \( F \vdash t' = t_j \) and \( F \vdash r = r_j \) for some \( j \).

Then given a formula \( \phi \) of the form \( t - g r \rightarrow t' \) it is enough to check, whether \( (\text{nf}(t'), r) \in B(t, a, g) \). This is decidable, because the sets \( B(t, a, g) \) are recursively defined and the function \( \text{nf} \) is recursive.
Proof of (*). By induction on complexity of \( g \), i.e. on \( n \) where \( g \in G_n \).

Thus, we can define

1. \( g \) atomic:

   Then \( B(t, a, l) = \{ (\text{nf}(t), r) : t_1 - l r \to t_2 \text{ is an axiom, } a \geq r \text{ and } \text{nf}(t) = \text{nf}(t_1) \} \)

   for any atomic label \( l \). \( B(t, a, l) \) is finite.

2. Let \( g = g_1 \cdot g_2 \):

   Let \( B(t, a, g_1) = \{ (t_j, r_j) : j = 1, \ldots, n \} \) be the formal representation set which can be computed by induction hypothesis for \( g_1 \) and \( t \).

   By induction hypothesis a finite representation set \( B(\text{nf}(t_i), a, g_2) = \{ (t_i, r_i) : i = 1, \ldots, k \} \) can be computed for \( a, g_2 \) and for each \( t_j \) for \( j = 1, \ldots, n \).

   Using the \( (TT) \) rule for all possible compositions we compute a finite representation set \( B(t, a, g_1 \cdot g_2) \). It has the form \( \{ (t_j, r_j) : (t_j, r_j) \in B(\text{nf}(t_i), a, g_2), (t_j, r_j) \in B(t, a, g_1), a \geq r_j + r_j \text{ for some } r, r_j, r_j \text{ and } t_j \} \).

3. Let \( g = g_0(g_i, \ldots, g_k) \), where \( g_0 \in L \). We have to consider all decompositions of \( t \) of the form \( t_0(t_1, \ldots, t_n) \).

   Let \( B(t, a, g_i) = \{ (t_{ij}, r_{ij}) : j = 1, \ldots, n \} \) for \( i = 0, \ldots, n \) be the formal representation sets which can be computed by induction hypothesis for \( g_i \) and \( t_1 \). By induction hypothesis these sets are finite.

   Using the \( (SR) \) rule for all possible decompositions of \( t \) we compute a finite representation set \( B(t, a, g_0(g_i, \ldots, g_k)) \). \( \square \)

It can easily be seen that in case of \( R_+ = N \) the above theorem applies to the examples of pedestrian lights, clock, lights with a clock, timer, and time out.

**Corollary 3.5.2.** Let \( \mathcal{R} = (\Sigma(R_+), E(R_+), L, RW) \) be a timed rewrite specification such that the equational theory of \( (\Sigma(R_+), E(R_+)) \) is decidable and \( RW \) is a finite set of literals. Then for every formula \( \phi \) of the form \( t_1 - g r \to t_2 \) it is decidable whether \( \phi \) is valid in \( \mathcal{R} \).

4. Application: concurrent object-based specification with TRL

In this section we show how TRL can be applied for defining an object-based specification language for real-time systems. We base this approach on the language Maude [19] introduced by Meseguer. In our version of Maude (called here timed Maude) the algebraic specification part of that language is kept unchanged, only concurrent rewriting is replaced by TRL. This means also that in timed Maude inheritance is treated in the same way as in Maude [18] by means of subsorting.

In the following, we present first informally an object-based example in a form that is used in current pragmatic object-oriented design methods. Then we introduce shortly the object-oriented part of timed Maude and its module concept (for a more
detailed description of these concepts in Maude see [19]). As illustration a timed Maude specification of the example is given.

4.1. The recycling machine example

The recycling machine is a slightly simplified version of the running example in Jacobson’s book on OOSE (Object-Oriented Software Engineering [10]).

A recycling machine receives returning items (such as cans or bottles) from a customer. Descriptions of these items and the daily total of the returned items of all customers are stored in the machine. If the customer presses a receipt button he gets a receipt for all items he has returned before. The receipt contains a list of the returned items as well as the total return sum.

An abstract design of this machine can be given in OOSE with the help of an object diagram that describes the objects of the problem together with their attributes and interrelationships, and of an interaction diagram that describes the flow of exchanged messages:

A recycling machine (with class name RM) is an object with two attributes storing the daily total and the current list of items (represented in Fig. 1 by the object diagram on the left). The interaction diagram (Fig. 1 on the right) shows (abstractly) the interaction between the customer and the recycling machine. The customer (class name USR) can send two kinds of messages to the machine: return messages containing a returned item \( i \) and receipt messages asking for a receipt with the list of items \( l \) he has returned as well as the total return sum amount \( l \). The timings of the rewrite steps are indicated on the right of the messages.

4.2. Timed Maude

Timed Maude is a variant of Maude where rewriting is replaced by time rewriting. As in Maude, an (object) class is declared by an identifier and a list of attributes and their types. \text{OId} is the type of Maude identifiers reserved for all object identifiers, \text{CId} is the type of all class identifiers.

\[
\begin{align*}
\text{USR} & \quad \text{return}(i) \quad \text{5} \\
\text{RM} & \quad \text{receipt} \quad r \\
\text{RM} & \quad \text{print}(l, s) \quad 1 \\
\end{align*}
\]

where \( s = \text{amount}(l) \) and \( r = l + \text{length}(l) \)

Fig. 1. Object diagram and interaction diagram of the recycling machine.
The object-oriented concept in Maude is the object module. The declaration of an object module (keyword omod) consists of an import list (protecting, extending or using), a number of class declarations (class), message declarations (msg), variable declarations (var), rewrite rules (rl) and possibly equations (eq).

A message is a term that consists of the message's name, the identifiers of the objects the message is addressed to and, possibly, parameters (in mixfix notation). An object is represented by a term - more precisely by a tuple - comprising a unique object identifier, an identifier for the class the object belongs to and a set of attributes with their values. For example, the term \langle rm : RM | \text{total} : d, \text{cur} : l \rangle represents an object with object identifier \( \text{rm} \) belonging to the class \( \text{RM} \). The attribute "total" has value \( d \), the attribute "cur" has value \( l \).

A Timed Maude program makes computational progress by rewriting its global state (called "configuration"). A configuration is a multiset, or a bag, of objects and messages. The sorts Message and Object are considered as subsorts of the sort Configuration. Multiset union is expressed by juxtaposition. Formally, a configuration is a term of the form: \( m_1 \otimes \cdots \otimes m_k \otimes o_1 \otimes \cdots \otimes o_l \), where \( \otimes \) is the function symbol for multiset union. Usually, the symbol \( \otimes \) is dropped in configurations. (We write \( m_1 \cdots m_k o_1 \cdots o_l \) instead of \( m_1 \otimes \cdots \otimes m_k \otimes o_1 \otimes \cdots \otimes o_l \).) In composite labels of rewrite steps we use an infix notation for \( \otimes \).

A timed rewrite step transforms a configuration into a subsequent configuration. The rewrite rules are of the form \( t_1 \rightarrow g \Rightarrow t_2 \), where \( g \) is a label and where \( t_1 \) and \( t_2 \) are terms of sort configuration.

We assume two restricted reflexivity axioms:

\( t \rightarrow 0 \Rightarrow t \) for all terms \( t \),
\( c_1 \otimes c_2 \rightarrow r \Rightarrow c_1 \otimes c_2 \) for all \( r \in R_+ \), where \( c_1, c_2 \) are variables of sort Configuration.

The first axiom allows for interleaving of actions that take 0 time. The second allows for a truly parallel behaviour of objects and configurations.

Moreover, we assume as in (Untimed) Maude commutativity and associativity of multiset union:

\[ x \otimes y = y \otimes x, \quad (x \otimes y) \otimes z = x \otimes (y \otimes z). \]

It is worth noticing that, in general, we do not require reflexivity neither for objects nor for messages.\(^4\)

We use the Maude convention that those attributes are omitted whose values remain unchanged by the rule, e.g. the attribute total.

Timed Maude gives the possibility of controlling the execution times of the rules. The following is a design specification in timed Maude for the 'recycling machine'.

\(^4\) Otherwise we would not be able to express hard real-time constraints in Timed Maude.
omod RM_SPEC_T is
  protecting ITEM LIST.

  class USR.
  msg print : List DM OId → Msg.
  op r0 : → Time.
  op r1 : → Time.
  var usr rm : OId.
  var r : Time.
  var i : Item.
  var d l : List
  var s : DM.

  rl (usr : USR) → id r → (usr : USR).
  *** the user can be idle
  rl (usr : USR) → usr_ret r0 → (usr : US) return (i, rm).
  *** returning an item takes r0 time units
  rl (usr : USR) → usr_rec r1 → (usr : US) receipt (usr, rm).
  *** requiring a receipt takes r1 time units
  rl print(l, s, usr) (usr : USR | c : x) → print 1 → (usr : USR | c : x).
  *** print is just a dummy rule here

  class RM | total : List, cur : List.
  msg return : Item Old → Msg.
  msg receipt : Old Old → Msg.
  rl (rm : RM) → id r → (rm : RM).
  *** the recycling machine can be idle
  rl return(i, rm) (rm : RM | total : d, cur : l) → rm_ret 5 →
    (rm : RM | total : (i::d), cur : (i::l)).
  *** processing a returned item takes 5 time units
  rl receipt(usr, rm) (rm : RM | total : d, cur : nil) → rm_rec 1 →
  *** requiring a receipt takes r1 time units
  rl receipts(usr, rm) (rm : RM | total : d, cur : (i::l)) → rm_rec 1 + length(l) →
    (rm : RM | total : d, cur : nil) print(i::l, amount(i::l), usr).
  *** no receipt is given if no item has been returned, calculating
  *** the receipt depends on the number of returned items

endom

The module RM_SPEC_T uses two predefined specifications: ITEM (which defines the price of the items), LIST (for lists of items where nil stands for the empty list, i::l adds the item i to list l and amount(l) computes the sum of the prices of all items of l). RM_SPEC_T declares the class RM of recycling machine objects and the class USR which simulates a possible user of the recycling machine.
The class USR is used only for technical reasons, in particular for testing the recycling machine. Any user can be idle (rule id). Giving back an item takes \( r_0 \) time units (rule \text{usr\_ret} ), requiring that a receipt takes \( r_1 \) time units (rule \text{usr\_rec} ). As a consequence, any action of the user takes at least \( r_0 \) or \( r_1 \) time units.

Class RM has two attributes total and cur to store the daily total and the current list of items. Objects of class RM accept two kinds of messages: return\((i, rm)\) returns the item \( i \) to the machine \( rm \), receipt\((usr, rm)\) asks \( rm \) for sending a receipt to \( usr \). The print message print\((l, amount(l), usr)\) sends the receipt to the customer \( usr \) with the list of items \( l \) he has returned as well as the total return sum amount\((l)\). In case \( usr \) did not return any item, no print message is sent. Note that the attribute total is required in the informal specification but it is not needed for the interaction between \( usr \) and machine. The recycling machine \( rm \) waits (by rule id) any amount of time until a receipt or return message arrives. Then due to the absence of reflexivity (for messages) it has to react immediately by one of the rules \text{rm\_rec} or \text{rm\_ret}. The processing of a returned item takes a fixed amount of time (5 time units), whereas calculating the bill depends on the number of the returned items.

The following example shows the behaviour of a system where the object \( usr \) returns two items (the second one 5 time units after the first one) and then asks for a receipt. Each rewriting step is performed by applying the synchronous replacement rule with \( t_0 \) being multiset union.

Example 4.2.1. (\( r_0 = 5 \), \( r_1 = 5 \). \( i_1 \) and \( i_2 \) denote here the first and second returned item)

\[
\begin{align*}
\langle \text{usr} \rangle \langle \text{rm} | \text{cur: nil} \rangle & \rightarrow \text{usr\_ret} \otimes \text{id} 5 \\
\langle \text{usr} \rangle \text{return}(i_1, m) \langle \text{rm} | \text{cur: nil} \rangle & \rightarrow \text{usr\_ret} \otimes \text{rm\_ret} 5 \\
\langle \text{usr} \rangle \text{return}(i_2, m) \langle \text{rm} | \text{cur: (i}_1 :: \text{nil}) \rangle & \rightarrow \text{usr\_rec} \otimes \text{rm\_ret} 5 \\
\langle \text{usr} \rangle \text{receipt}(\text{usr}, \text{rm}) \langle \text{rm} | \text{cur: (i}_2 :: i_1 :: \text{nil}) \rangle & \rightarrow \text{id} \otimes \text{rm\_rec} 2 \\
\langle \text{usr} \rangle \langle \text{rm} | \text{cur: nil} \rangle \text{print}(i_2 :: i_1 :: \text{nil}, \text{amount}(i_2 :: i_1 :: \text{nil}), \text{usr}) & \rightarrow \text{print} \otimes \text{id} 1 \\
\langle \text{usr} \rangle \langle \text{rm} | \text{cur: nil} \rangle & \rightarrow \text{id} \otimes \text{rm\_rec} 2 \\
\end{align*}
\]

However, impatience causes deadlock: if the customer \( usr \) returns items faster than the machine can store them (e.g. each 3 time units), a deadlock situation occurs where none of the rules can be applied although some messages are not yet processed.

4.2.2. Example: \( r_0 = 3 \)

\[
\begin{align*}
\langle \text{usr} \rangle \langle \text{rm} | \text{cur: l} \rangle & \rightarrow \text{usr\_ret} \otimes \text{id} 3 \\
\langle \text{usr} \rangle \text{return}(i_1, m) \langle \text{rm} | \text{cur: l} \rangle & \rightarrow \text{usr\_ret} \otimes \text{age} 3 \\
\langle \text{usr} \rangle \text{return}(i_2, m) \text{age}(\text{return}(i_1, m) \langle \text{rm} | \text{cur: l} \rangle, 3) & \rightarrow \text{Deadlock!}
\end{align*}
\]

The deadlock occurs because a message must be processed by the recycling machine immediately after its arrival.
More generally, the recycling machine works correctly under the assumption that time intervals between the production of receipt or receive messages are larger than the processing times of the machine, i.e. if $\max(r_0, r_1) \geq 5$.

The situation where additional assumptions are needed for the correctness of the behaviour of a class or object arises often in distributed programming because of the lack of a global state and common knowledge. This leads to a proof methodology known as "assume-guarantee" where processes make assumptions about other processes (see e.g. [7]).

5. Concluding remarks

In this paper we have presented Timed Rewriting Logic, a logic analogous to Meseguer's Rewriting Logic which allows us to describe hard and soft real-time constraints. For practical applications we need further extensions of TRL: on one hand, it is necessary to generalize the form of the axioms to include conditional formulas and quantifications (in a similar way as [17]); a first simple approach to do this has already been sketched in Section 4. On the other hand, it is not enough to deal with single time instants; a next variant of Timed Rewriting Logic will be able to express rules with time intervals. Also in this case which is under current investigation we hope to get similar results on the existence of initial model and the decidability of timed rewrite steps.

One of the main reasons for choosing Rewriting Logic as the basis of our calculus is the elegant treatment of object-oriented design specifications with this logic expressed within the specification language Maude (e.g. [16–19]). In a similar way we intend to use TRL for specifying object-oriented real-time systems by defining Timed Maude as a combination of Maude (which is written in Rewriting Logic) with Timed Rewriting Logic.

Our approach is rather model based, and therefore similar to process algebras. In the future, we will try to find a more abstract (property oriented) language for specification and verification which would allow for proving liveness and safety of TRL specifications. We will also try to extend already existing methods of term rewriting theory for proving deadlock freeness, livelock freeness, and absence of Zeno behaviours.

References