Specification of real-time and hybrid systems in rewriting logic ⋆

Peter Csaba Ölveczky [a,b,1] and José Meseguer [a]

[a] *Computer Science Laboratory, SRI International, Menlo Park, USA*

[b] *Dept. of Informatics, University of Bergen, Norway*

Dedicated with affection to Miklós Ölveczky.

Abstract

This paper explores the application of rewriting logic to the executable formal modeling of real-time and hybrid systems. We give general techniques by which such systems can be specified as ordinary rewrite theories, and show that a wide range of real-time and hybrid system models, including object-oriented systems, timed automata [3], hybrid automata [2], timed and phase transition systems [25], and timed extensions of Petri nets [1,35], can indeed be expressed in rewriting logic quite naturally and directly. Since rewriting logic is executable and is supported by several language implementations, our approach complements property-oriented methods and tools less well suited for execution purposes, and can be used as the basis for symbolic simulation and formal analysis of real-time and hybrid systems [38]. The relationships with the timed rewriting logic approach of Kosiuczenko and Wirsing [22] are also studied.

Key words: Rewriting logic; Maude; Executable formal specification; Real-time systems; Hybrid systems; Timed Petri nets; Real-time object-oriented systems

⋆ Supported by DARPA through Rome Laboratories Contract F30602-97-C-0312, by DARPA and NASA through Contract NAS2-98073, by Office of Naval Research Contract N00014-96-C-0114, and by National Science Foundation Grant CCR-9633363.

1 Supported by The Norwegian Research Council.

1 Introduction

This paper explores the application of rewriting logic to the *executable formal modeling* of real-time and hybrid systems. The general conceptual advantage of using a logic instead of using a specific model is that many different models can be specified in the same logic, each in its own terms, rather than by means of possibly awkward translations into a fixed model. The advantages of using rewriting logic as a semantic framework for concurrency models has been amply demonstrated (see the surveys [30,32]). This work shows that a number of well-known models of real-time and hybrid systems can likewise be naturally specified in the rewriting logic framework.

Thus, rewriting logic can be used to specify many different formal models of such systems in a unified logic. But, since rewriting logic is executable, and is supported by implementations of the rewriting logic languages Maude [12], ELAN [9], and CafeOBJ [16], these models can be executed and can be formally analyzed in a variety of ways. This is in contrast to the most well-known formal methods tools for real-time and hybrid systems such as Kronos [44], STeP [41,7,27], and UPPAAL [23]. These are model checking tools which require the user to specify both the system and the formal properties the system should satisfy. The tools then try to check whether the system satisfies a given abstract property. However, these tools are not well suited for directly executing the system itself. The same can be said about HyTech [20], which takes a hybrid system description with some parameters unspecified, and returns the concrete values of the parameters which would make the system satisfy some given property. Of course, different tools have different important strengths of their own. The point is that executable specification methods and tools can complement those strengths in new ways.

To see how rewriting logic complements more abstract specification formalisms such as temporal logic as well as more concrete, automaton-based ones, one can think of it as covering an *intermediate operational level*, that can substantially help in bridging the gap between more abstract, property-oriented, specifications and actual implementations by providing:

- a precise mathematical model of the system (the initial model [29]) against which more abstract specifications can be proved correct by means of inductive theorem proving, model checking, and other techniques;
- support for other useful techniques of automated or semi-automated formal reasoning and analysis at the rewriting logic and equational logic levels, such as coherence [43], confluence, and strategy-based formal analysis;
- support for executable specification, symbolic simulation, and a wide range of formal analyses;
- the possibility of generating correct implementations from specifications by

theory transformations and code generation techniques.

Using the results in this paper, the capabilities for symbolic simulation are indeed supported by Real-Time Maude [38], a language and tool for the symbolic simulation and formal analysis of real-time and hybrid systems.

In this paper, we show that ordinary rewrite theories are sufficient to specify real-time systems in a natural way. Essentially, all we need is to include in the specification a *Time* data type satisfying appropriate equational properties. However, it is sometimes useful to highlight the real-time aspect by making explicit the duration information for some rewrite rules. We formalize this idea in Section 2 by means of *real-time rewrite theories*; but we show that, by adding an explicit clock, they are reducible to ordinary rewrite theories in a way that preserves all the expected properties.

The naturalness of the specification method, and its smooth integration with rewriting logic's support for object-oriented specification, is explored and illustrated with examples in Section 3, in which we also address the question of how generally and naturally rewriting logic can be used to express a variety of real-time and hybrid system models. We show in detail how, besides object-oriented real-time systems, a wide range of such models, including timed automata [3], hybrid automata [2], timed and phase transition systems [25], and timed extensions of Petri nets [1,35], can indeed be expressed in rewriting logic quite naturally and directly.

The first important research contribution exploring the application of rewriting logic to real-time specification has been the work of Kosiuczenko and Wirsing on *timed rewriting logic* (TRL) [22], an extension of rewriting logic where the rewrite relation is labeled with time stamps. TRL has been shown well-suited for giving object-oriented specifications of complex hybrid systems such as the steam-boiler [36], and has also been applied to give semantics to the SDL telecommunications specification language [42]. In fact, rewriting logic object-oriented specifications in the Maude language [12] have a natural extension to TRL object-oriented specifications in *Timed Maude* [22,36].

The approach taken here is different. As already mentioned, we argue that real-time systems can be specified in ordinary rewriting logic, and that reasoning about their behavior does not require a special inference system of their own, such as the one proposed in TRL. Even when special notation highlighting real-time aspects—such as that provided by real-time rewrite theories—is used, we show that this can essentially be regarded as syntactic sugar. This has the conceptual advantage of remaining within a simpler theoretical framework, and the practical advantage of being able to use the existing language implementations of rewriting logic to execute specifications. Therefore, it seems both conceptually and practically useful to study the relationships between our approach and TRL. We do so in Section 4, where we show that there is a map of logics $\mathcal{M} : TRL \longrightarrow RWL$ sending each TRL specification to a corresponding rewrite theory in such a way that logical entailment is preserved. However, the translated theory $\mathcal{M}(\mathcal{T})$ can in general prove additional sentences. This is due to some intrinsic conceptual differences between both formalisms that our analysis reveals.

1.1 Prerequisites on rewriting logic and Maude

We assume familiarity with the basic concepts of rewriting logic as presented in [29,32]. We recall here only the most basic notions that we shall use. Rewriting logic specifications are *rewrite theories* of the form $\mathcal{R} = (\Sigma, E, L, R)$, where (Σ, E) is an equational theory, L is a set of labels, and R is a collection of *labeled rewrite rules* of the form

$$[l] : t \longrightarrow t' \ \textbf{if} \ \bigwedge_{i=1}^{n} u_i \longrightarrow v_i \wedge \bigwedge_{j=1}^{m} w_j = w_j',$$

with $l \in L$, which is implicitly universally quantified by the variables appearing in the Σ-terms t, t', u_i, v_i, w_j, and w_j'. For most specifications in this paper the equational theory (Σ, E) will be assumed to be *order-sorted* [17], that is, the set of sorts comes equipped with a partial order relation, with $s \leq s'$ interpreted as subset inclusion $A_s \subseteq A_{s'}$ in a model A. Furthermore, operation symbols can be *subsort overloaded* (as for example the addition symbol $+$ for naturals, integers, and rationals). Such overloaded operators are required to yield the same result for the same arguments, regardless of the overloaded operator that is applied. In addition, the order-sorted signature Σ is assumed to be *coherent* [17]. This ensures that all terms have a least sort and that the poset of sorts has good properties. In Section 3.6.2 we give a rewriting logic specification that uses an even more expressive equational logic, namely, *membership equational logic* [31].

We make frequent use of the *initial model* construction $\mathcal{T}_\mathcal{R}$ associated to a rewrite theory $\mathcal{R}$, in which rewrite proofs $\alpha : t \longrightarrow t'$, derivable from the rules in $\mathcal{R}$ using the rules of deduction of rewriting logic, are equated modulo a natural notion of *proof equivalence* [29]. However, $\mathcal{T}_\mathcal{R}$ has to be understood in an order-sorted sense, so that for each sort s we have an associated category $(\mathcal{T}_\mathcal{R})_s$, with arrows $\alpha : t \longrightarrow t'$ equivalence classes of proofs with t, t' ground terms of sort s, and with arrow composition corresponding to application of the transitivity rule.

Throughout the text we often use Maude-like notation [12] to present specific rewrite theories. For the most part this notation is self-explanatory. In the case of object-oriented modules, we explain their syntax and basic assumptions in Section 3.5.

2 Time models and real-time rewrite theories

After specifying equationally the general requirements for the models of time that we will consider (Section 2.1) we propose a general notion of *real-time rewrite theory*, consisting of an ordinary rewrite theory, where rewrite rules affecting the whole system have associated time-duration expressions (Section 2.2). We then show that real-time rewrite theories form a category (Section 2.3) and that they can be reduced to *ordinary* rewrite theories by adding an explicit clock to the global state in a way that preserves all the expected properties (Section 2.4). We finish the section with a discussion of several issues and specification techniques for real-time rewrite theories (Section 2.5).

2.1 Time Models

Time is modeled abstractly by a commutative monoid $(Time, +, 0)$ with additional operators $\leq$, $<$, and $\dot{-}$ ("monus") satisfying the following Maude theory. [2]

fth *TIME* **is**
protecting *BOOL*
  sort *Time*
  op $0 : \to Time$
  op $\_+\_ : Time\ Time \to Time \quad [assoc\ \ comm\ \ id : 0]$
  ops $\_<\_, \_\leq\_ : Time\ Time \to Bool$
  op $\_\dot{-}\_ : Time\ Time \to Time$
  vars $x_r, y_r, z_r, w_r : Time$
  ceq $y_r = z_r$ **if** $x_r + y_r == x_r + z_r$
  eq $(x_r < x_r) = false$
  ceq $x_r < z_r = true$ **if** $x_r < y_r$ and $y_r < z_r$
  eq $(x_r \leq y_r) = (x_r < y_r)$ or $(x_r == y_r)$
  eq $0 \leq x_r = true$
  ceq $x_r + y_r \leq z_r + w_r = true$ **if** $x_r \leq z_r$ and $y_r \leq w_r$
  eq $x_r \leq (x_r \dot{-} y_r) + y_r = true$

[2] Note that conditional equations (**ceq**) having as condition a Boolean term b are shorthand notation for the same equations with condition $b = true$. In particular, such Boolean expressions can contain the Boolean-valued equality predicate denoted $==$.

ceq $(x_r \dot{-} y_r) + y_r = x_r$ **if** $y_r \leq x_r$
ceq $x_r \dot{-} z_r \leq y_r \dot{-} z_r = true$ **if** $x_r \leq y_r$
endft

In this theory, it can for example be proved that the relation $\leq$ is a partial order with least element 0, and that for all $x_r, y_r : Time$, $y_r \leq x_r$ if and only if there exists a unique z_r (namely $x_r \dot{-} y_r$) such that $x_r = y_r + z_r$.

For simulation and executable specification purposes we will be interested in *computable* models of the above theory *TIME*. This means that all the operations are computable. By the Bergstra-Tucker Theorem [6], any computable algebra is finitely specifiable as an initial algebra for a set E of Church-Rosser and terminating equations. For example, the nonnegative rational numbers can be so specified as a model of *TIME* by adding a subsort Rat_+ to the specification of rationals in [17], and extending it with order and monus operations in the obvious way. Similarly, the real algebraic numbers with the standard order are also computable [39], and therefore have a finite algebraic specification with Church-Rosser and terminating equations. Note that just taking a constructive version of the real numbers will not yield a computable data type, because the equality and order predicates on the constructive reals are not computable [5].

In some examples in this paper we will need to extend the time domain with a new value ∞ and/or require that the time domain is linear. The following theory gives an abstract specification of the time domain extended with a value ∞.

fth $TIME_\infty$ **is**
including *TIME*
sort $Time_\infty$
subsort $Time \leq Time_\infty$
op $\infty : \to Time_\infty$
op $\_ \leq \_ : Time_\infty\ Time_\infty \to Bool$
op $\_ \dot{-} \_ : Time_\infty\ Time \to Time_\infty$
op $\_ + \_ : Time_\infty\ Time_\infty \to Time_\infty\ [assoc\ \ comm\ \ id : 0]$
var $x_r : Time$
var $y_r : Time_\infty$
eq $y_r \leq \infty = true$
eq $\infty \leq x_r = false$
eq $x_r < \infty = true$
eq $\infty < y_r = false$
eq $\infty \dot{-} x_r = \infty$
eq $\infty + y_r = \infty$
endft

Linear time can be specified by the following theory:

fth *LTIME* **is**
including *TIME*
  op $\min : \mathit{Time}\ \mathit{Time} \to \mathit{Time}\ \ [\mathit{comm}]$
  vars $x_r, y_r : \mathit{Time}$
  ceq $x_r = y_r$ **if** $\mathit{not}(x_r < y_r)$ and $\mathit{not}(y_r < x_r)$
  ceq $\min(x_r, y_r) = y_r$ **if** $y_r \leq x_r$
endft

This theory can also be extended with a time value ∞ as follows:

fth LTIME_∞ **is**
including *LTIME*, TIME_∞
  op $\min : \mathit{Time}_\infty\ \mathit{Time}_\infty \to \mathit{Time}_\infty\ \ [\mathit{comm}]$
  var $x_r : \mathit{Time}_\infty$
  eq $\min(\infty, x_r) = x_r$
endft

Notation: We will use symbols $r,\ r',\ r_1, \ldots$ to denote time values and $x_r,\ y_r, \ldots$ to denote *variables* of the sort of the time domain.

2.2 Real-time rewrite theories

We define in this section real-time rewrite theories; they are used to specify real-time systems in rewriting logic and contain duration information for some rules. Rules are divided into *tick rules*, that model the elapse of time on a system, and *instantaneous rules*, that model instantaneous change and are assumed to take zero time. Having a tick rule $[\mathit{tick}] :\ t \longrightarrow t'$ could lead to rewrites $f(t, u) \longrightarrow f(t', u)$, i.e., rewrites where time only elapses in a part of the system under consideration. To ensure that time advances uniformly in all the parts of a state, we introduce a new sort *System*, with no subsorts, and a free constructor $\{\_\} : \mathit{State} \to \mathit{System}$ with the intended meaning that $\{t\}$ denotes the whole system, which is in state t. Uniform time elapse is then ensured if the global state always has the form $\{t\}$ and every tick rule is of the form $[\mathit{tick}] :\ \{t\} \longrightarrow \{t'\}$.

Before defining real-time rewrite theories, we recall the notion of a theory morphism between equational theories, which will be used to show how the time domain is interpreted in a real-time rewrite theory.

Definition 1 *An* equational theory morphism $H : (\Sigma, E) \to (\Sigma', E')$ *consists of a monotone map* $H : \mathit{sorts}(\Sigma) \to \mathit{sorts}(\Sigma')$*, and a mapping sending each* *function symbol*[3] $f : s_1 \ldots s_n \to s$ *in* Σ *to a* Σ'*-term* $H(f_{s_1 \ldots s_n, s})$ *of sort* $H(s)$, *such that its set of variables is contained in the set* $x_1 : H(s_1), \ldots, x_n : H(s_n)$, *and such that if* $f : s_1 \ldots s_n \to s$, *and* $f : s'_1 \ldots s'_n \to s'$ *are subsort overloaded operators with* $s_i \leq s'_i$, $s \leq s'$, *then the term* $H(f_{s_1 \ldots s_n, s})$ *can be obtained from the term* $H(f_{s'_1 \ldots s'_n, s'})$ *by replacing each variable* $x_i : H(s'_i)$ *by the corresponding variable* $x_i : H(s_i)$. *Furthermore, for each axiom* $(\forall y_1 : s_1, \ldots, y_k : s_k)\ u = v$ **if** C *in* E,

$$E' \models (\forall y_1 : H(s_1), \ldots, y_k : H(s_k))\ \ H^*(u) = H^*(v)\ \textbf{if}\ H^*(C)$$

holds, for H^* *the homomorphic extension of* H *to terms, and to equations in the condition* C.

A real-time rewrite theory can then be defined as a rewrite theory together with an interpretation of the abstract time domain and an assignment of duration to rules involving terms of sort *System*:

Definition 2 *A* real-time rewrite theory $\mathcal{R}_{\phi,\tau}$ *is a tuple* $(\mathcal{R}, \phi, \tau)$, *where* $\mathcal{R} = (\Sigma, E, L, R)$ *is a rewrite theory, such that*[4] *:*

- ϕ *is an equational theory morphism* $\phi : \mathit{TIME} \to (\Sigma, E)$ *where TIME is the theory defined in Section 2.1,*
- *the time domain is functional; that is, whenever* $\alpha : r \longrightarrow r'$ *is a rewrite proof in* $\mathcal{R}$ *and* r *is a term of sort* $\phi(\mathit{Time})$, *then* $r = r'$ *and* α *is equivalent to the identity proof* r,
- (Σ, E) *contains a designated sort that we usually call State and a specific sort System with no subsorts or supersorts and with only one operator*

  $$\{\_\} : \mathit{State} \to \mathit{System}$$

  which satisfies no non-trivial equations; furthermore, for any $f : s_1 \ldots s_n \to s$ *in* Σ, *the sort System does not appear among the* $s_1, \ldots, s_n$*;*
- τ *is an assignment of a term* $\tau_l(x_1, \ldots, x_n)$ *of sort* $\phi(\mathit{Time})$ *to each rewrite rule in* $\mathcal{R}$ *of the form*

  $$(\dagger) \qquad [l] : u(x_1, \ldots, x_n) \longrightarrow u'(x_1, \ldots, x_n)\ \textbf{if}\ C(x_1, \ldots, x_n)$$

  where u *and* u' *are terms of sort System.*

[3] Since the variables $x_1, \ldots, x_n$ are ordered, the assignment $f : s_1 \ldots s_n \to s \mapsto H(f_{s_1 \ldots s_n, s})$ can alternatively be understood as an assignment $f(x_1 : s_1, \ldots, x_n : s_n) \mapsto H(f_{s_1 \ldots s_n, s})$.

[4] We give a definition based on loose semantics of rewrite theories. Real-time rewrite theories can be defined in a similar way in an initial semantics setting.

Notation: We will write

$$[l]: \ u(x_1, \ldots, x_n) \stackrel{\tau_l(x_1,\ldots,x_n)}{\longrightarrow} u'(x_1, \ldots, x_n) \ \textbf{if} \ C(x_1, \ldots, x_n)$$

for a rule l of sort *System* with duration term τ_l. If $\tau_l(x_1, \ldots, x_n)$ equals $\phi(0)$, the rule l will often be written

$$[l]: \ u(x_1, \ldots, x_n) \longrightarrow u'(x_1, \ldots, x_n) \ \textbf{if} \ C(x_1, \ldots, x_n).$$

We will also write $Time_\phi$, 0_ϕ, and $+_\phi$ for, respectively, $\phi(Time)$, $\phi(0)$, and $\phi(+)$. In the rest of the paper we will often further simplify the notation and write *Time*, 0, and $+$ instead of $Time_\phi$, 0_ϕ, and $+_\phi$. Finally, we will often write $Time_\infty$, ∞, $\ldots$ instead of $\phi'(Time_\infty)$, $\phi'(\infty)$, $\ldots$ when $\phi' : TIME_\infty \to (\Sigma, E)$ is an extension of ϕ.

We call rules of the form (†) *global rules.* A global rule l is a *tick rule* if its duration $\tau_l(x_1, \ldots, x_n)$ is different from 0_ϕ for some instance of its variables, and is an *instantaneous rule* otherwise. Rules not of the form (†) are called *local* rules, because they do not act on the system as a whole, but only on some system components. Local rules are always viewed as instantaneous rules that take zero time.

The total time elapse $\tau(\alpha)$ of a rewrite $\alpha : \ \{t\} \longrightarrow \{t'\}$ of sort *System* is defined as the sum of the time elapsed in each tick rule application in α, and can easily be extracted from the proof:

Definition 3 *Let $(\mathcal{R}, \phi, \tau)$ be a real-time rewrite theory with $\mathcal{R} = (\Sigma, E, L, R)$ and let Time denote the time domain $(T_{\Sigma,E})_{Time_\phi}$ viewed as a monoid and therefore as a category with a single object 0, and with the time values as arrows in the usual way. The* time extraction functor

$$\tau : (\mathcal{T}_\mathcal{R})_{System} \to Time$$

which gives the total time elapse $\tau(\alpha)$ of a proof $\alpha : t \longrightarrow t'$, with t, t' ground terms of sort System, is defined as follows:

- $\tau(t) = 0_\phi$ *for every term (seen as an identity arrow) in $(\mathcal{T}_\mathcal{R})_{System}$,*
- $\tau(\{\alpha\}) = 0_\phi$ *for a proof term whose top operator is the constructor $\{\_\}$,*
- $\tau(l(\alpha_1, \ldots, \alpha_n)) = \tau_l(t_1, \ldots, t_n)$ *if l is a (system) rule of the form (†) and $\alpha_1 : t_1 \longrightarrow t_1', \ldots, \alpha_n : t_n \longrightarrow t_n'$ are proofs, and*
- $\tau(\alpha; \beta) = \tau(\alpha) +_\phi \tau(\beta)$ *for proofs α and β.*

This definition does not depend on the choice of representative proof terms. That is, if α and β are two equivalent proofs of terms of sort *System* in a real-time rewrite theory $(\mathcal{R}, \phi, \tau)$, then $\tau(\alpha) = \tau(\beta)$.

Given a real-time rewrite theory $\mathcal{R}$, a *computation* is a non-extensible sequence $t_0 \longrightarrow t_1 \longrightarrow \cdots \longrightarrow t_n$ (that is, one for which t_n cannot be further rewritten) or an infinite sequence $t_0 \longrightarrow t_1 \longrightarrow \cdots$ of one-step $\mathcal{R}$-rewrites $t_i \longrightarrow t_{i+1}$, with t_i and t_{i+1} ground terms, starting with a given initial term t_0 of sort *System*. It should be noted that, since we model time elapse *explicitly* (by rewrite rules), the usual requirement that the total time elapse in a computation is infinite is not needed. Time elapse is totally up to the specifier; we allow both terminating computations and infinite computations with finite total time elapse, that is, "Zeno" computations.

2.3 A category of real-time rewrite theories

The notion of theory morphism—also called theory interpretation—between real-time rewrite theories plays an important role in this work. We give a definition of theory morphism between real-time rewrite theories based on loose semantics and preservation of durations of rewrites. Morphisms based on properties of the initial models of theories, and morphisms having less restrictive requirements on the relationships between the durations in the rewrites could be defined in a similar way. We begin by defining theory morphisms between ordinary rewrite theories.

Definition 4 *A* rewrite theory morphism *from a rewrite theory* $\mathcal{R} = (\Sigma, E, L, R)$ *to another rewrite theory* $\mathcal{R}' = (\Sigma', E', L', R')$ *consists of:*

- *an equational theory morphism* $H : (\Sigma, E) \to (\Sigma', E')$*, and*
- *a map* $H : L \longrightarrow L'$ *of labels such that for each rule* $[l] : t \longrightarrow t'$ **if** C *in* R *the rule*

$$[H(l)] :\ H^*(t) \longrightarrow H^*(t') \ \textbf{if}\ H^*(C)$$

  is in R' *up to a renaming of its variables.*

Rewrite theory morphisms compose in the expected way and define a category *RWTh* of rewrite theories.

Definition 5 *A* real-time rewrite theory morphism *from a real-time rewrite theory* $(\mathcal{R}, \phi, \tau)$ *to a real-time rewrite theory* $(\mathcal{R}', \phi', \tau')$ *is a rewrite theory morphism* $H : \mathcal{R} \to \mathcal{R}'$ *such that:*

- $\phi' = H \circ \phi$,
- H *maps the designated sort of the states in* $\mathcal{R}$ *to the designated state sort in* $\mathcal{R}'$*, maps the sort System to itself, and leaves the constructor* $\{\_\}$ *unchanged, and*

- *H preserves the duration of the tick rules in the sense that for each rule l in $\mathcal{R}$ of sort System,*

$$E' \vdash H^*(\tau_l(x_1 : s_1, \ldots, x_n : s_n)) = \tau'_{H(l)}(x_1 : H(s_1), \ldots, x_n : H(s_n)).$$

It is easy to check that the usual composition of rewrite theory morphisms defines a category *RTRWTh* with real-time rewrite theories as objects and real-time rewrite theory morphisms as arrows.

2.4 Real-time theories internalized in rewriting logic

By adding a clock to the state, a real-time rewrite theory $(\mathcal{R}, \phi, \tau)$ can be transformed into an ordinary rewrite theory without losing timing information. A state in such a clocked system is of the form $\langle t, r\rangle$ with t the global state of sort *System*, and r a value of sort $Time_\phi$, which intuitively is supposed to denote the total time elapsed in a computation if in the initial state the clock had value 0_ϕ.

Definition 6 *The* internalizing functor $(\_)^C$ *from the category RTRWTh of real-time rewrite theories to the category RWTh of rewrite theories takes a real-time rewrite theory $(\mathcal{R}, \phi, \tau)$ to a rewrite theory $\mathcal{R}^C_{\phi,\tau} = (\Sigma^C_{\phi,\tau}, E^C_{\phi,\tau}, L^C_{\phi,\tau}, R^C_{\phi,\tau})$ as follows:*

- *the sorts in $\Sigma^C_{\phi,\tau}$ are those in $\mathcal{R}$, together with a new sort ClockedSystem,*
- *the operations in $\Sigma^C_{\phi,\tau}$ are those in $\mathcal{R}$, together with a new free constructor*

$$\langle \_, \_ \rangle : \mathit{System}\ \mathit{Time}_\phi \to \mathit{ClockedSystem},$$

- *the axioms in $E^C_{\phi,\tau}$ are unchanged from those in $\mathcal{R}$,*
- *$R^C_{\phi,\tau}$ contains the local rules in $\mathcal{R}$ of sorts other than System, together with a rule*[5]

$$[l^C_{\phi,\tau}(x_1, \ldots, x_n, x_r)] : \langle u(x_1, \ldots, x_n), x_r\rangle \longrightarrow \\ \langle u'(x_1, \ldots, x_n), x_r +_\phi \tau_l(x_1, \ldots, x_n)\rangle \ \mathbf{if}\ C(x_1, \ldots, x_n)$$

for each rule

$$[l(x_1, \ldots, x_n)] :\ u(x_1, \ldots, x_n) \longrightarrow u'(x_1, \ldots, x_n)\ \mathbf{if}\ C(x_1, \ldots, x_n)$$

[5] In the unlikely case that any condition C of a rule in $\mathcal{R}$ contains a conjunct $v \longrightarrow v'$ of sort *System*, each such conjunct is replaced by a conjunct $\langle v, 0_\phi\rangle \longrightarrow \langle v', y_r\rangle$ in the condition in $\mathcal{R}^C_{\phi,\tau}$, where y_r is a fresh variable of sort $Time_\phi$.

in $\mathcal{R}$ of sort System, where x_r is a variable of sort $Time_\phi$ which is not in the list $x_1, \ldots, x_n$.

The internalizing functor is defined as expected on arrows in RTRWTh; i.e., an arrow H in RTRWTh is mapped to H^C, which coincides with H on $\mathcal{R}$, leaves the new sort and operator unchanged, and takes a label $l^C_{\phi,\tau}$ of a rule of sort ClockedSystem to the label $(H(l))^C$.

For the sake of a simpler exposition, in the rest of the paper we will assume that no condition of a rewrite rule in a real-time theory contains a rewrite conjunct of sort *System*. We also assume, without loss of generality, that no variable of sort *System* is introduced in the condition of a rule.

Proposition 7 *The mapping $(\_)^C$ above defines a functor from RTRWTh to RWTh.*

Proposition 8 *Let V be the forgetful functor from the category RTRWTh of real-time rewrite theories to the category RWTh of rewrite theories defined by*

$$V((\mathcal{R}, \phi, \tau)) = \mathcal{R} \ \textit{ and } \ V(H) = H.$$

Then the map of rewrite theories $\pi_{(\mathcal{R},\phi,\tau)} : \mathcal{R}^C_{\phi,\tau} \to \mathcal{R}$ defined by:

- *mapping each sort and operator in $\Sigma^C_{\phi,\tau}$ other than ClockedSystem and $\langle \_, \_ \rangle$ identically to themselves,*
- *mapping ClockedSystem to System, mapping the operator $\langle \_, \_ \rangle$ to the term $x_1 : System$, and*
- *mapping each label $l^C_{\phi,\tau}$ to the label l*

defines a natural transformation $\pi : (\_)^C \Rightarrow V$.

Since a rewrite theory morphism $H : \mathcal{R} \to \mathcal{R}'$ induces a forgetful functor $U_H : \mathcal{R}'\text{-}Sys \to \mathcal{R}\text{-}Sys$ in the opposite direction for the corresponding categories of models, our natural rewrite theory morphism $\pi : \mathcal{R}^C_{\phi,\tau} \to \mathcal{R}$ induces a forgetful functor $U_\pi : \mathcal{R}\text{-}Sys \to \mathcal{R}^C_{\phi,\tau}\text{-}Sys$. In particular, the initial model $\mathcal{T}_\mathcal{R}$ is sent to the $\mathcal{R}^C_{\phi,\tau}$-system $U_\pi(\mathcal{T}_\mathcal{R})$ and, by initiality of $\mathcal{T}_{\mathcal{R}^C_{\phi,\tau}}$, we have a unique $\mathcal{R}^C_{\phi,\tau}$-homomorphism $h : \mathcal{T}_{\mathcal{R}^C_{\phi,\tau}} \to U_\pi(\mathcal{T}_\mathcal{R})$ such that:

- h takes objects and arrows of every sort except *ClockedSystem* to themselves,
- $h(\langle t, r \rangle) = t$ for each object $\langle t, r \rangle$ in $(\mathcal{T}_{\mathcal{R}^C_{\phi,\tau}})_{ClockedSystem}$,
- $h(\langle \alpha, \beta \rangle : \langle t, r \rangle \longrightarrow \langle t', r' \rangle) = \alpha : t \longrightarrow t'$,
- $h(l^C_{\phi,\tau}(\alpha_1, \ldots, \alpha_n, \alpha_{n+1}) : \langle t, r \rangle \longrightarrow \langle t', r' \rangle) = l(\alpha_1, \ldots, \alpha_n) : t \longrightarrow t'$, and
- $h(\alpha; \beta) = h(\alpha); h(\beta)$.

The map h expresses the essential semantic equivalence between the initial model of a real-time theory $(\mathcal{R}, \phi, \tau)$ and that of its clocked representation $\mathcal{R}^C_{\phi,\tau}$ in the precise sense that, as we shall see:

(1) if $\alpha : t \longrightarrow t'$ is an arrow in $(\mathcal{T}_\mathcal{R})_{\mathit{System}}$ with $\tau(\alpha) = r$, then, for each value r' of sort Time_ϕ there is a unique arrow $\alpha' : \langle t, r' \rangle \longrightarrow \langle t', r' +_\phi r \rangle$ in $(\mathcal{T}_{\mathcal{R}^C_{\phi,\tau}})_{\mathit{ClockedSystem}}$ such that $h(\alpha' : \langle t, r' \rangle \longrightarrow \langle t', r' +_\phi r \rangle) = \alpha : t \longrightarrow t'$, and

(2) whenever $\alpha : \langle t, r \rangle \longrightarrow \langle t', r' \rangle$ is an arrow in $(\mathcal{T}_{\mathcal{R}^C_{\phi,\tau}})_{\mathit{ClockedSystem}}$ then $r' = r +_\phi \tau(h(\alpha))$.

These two properties are immediate consequences of the following

Theorem 9 *Let $(\mathcal{R}, \phi, \tau)$ be a real-time rewrite theory and let $\alpha : t \longrightarrow t'$ be an arrow in $(\mathcal{T}_\mathcal{R})_{\mathit{System}}$ (therefore, with t and t' ground terms of sort System). Then, for each value r in the time domain, there is a* unique *arrow*

$$\alpha' : \langle t, r \rangle \longrightarrow \langle t', r' \rangle$$

in $(\mathcal{T}_{\mathcal{R}^C_{\phi,\tau}})_{\mathit{ClockedSystem}}$ such that $h(\alpha') = \alpha$, and, in addition, $r' = r +_\phi \tau(\alpha)$.

The theorem can be proved by induction on the structure of the proof terms by first proving the theorem for one-step rewrites, and then proving it for all proofs between terms of sort *System* using the facts that every proof factorizes into a sequence of one-step rewrites and that h distributes over one-step rewrite proofs.

The above theorem implies that, whenever $\alpha : \langle t, r \rangle \longrightarrow \langle t', r' \rangle$ is an arrow in $\mathcal{T}_{\mathcal{R}^C_{\phi,\tau}}$, then the arrow $h(\alpha) : t \longrightarrow t'$ satisfies $r +_\phi \tau(h(\alpha)) = r'$. It also implies that h, viewed as a functor $h : (\mathcal{T}_{\mathcal{R}^C_{\phi,\tau}})_{\mathit{ClockedSystem}} \to (\mathcal{T}_\mathcal{R})_{\mathit{System}}$, is full and faithful, and is an opfibration [4].

2.5 Discussion

We discuss several system specification issues and techniques, including tick rules, eager and lazy rules, and time as an action on the system.

2.5.1 Specifying the tick rules

For simulation of a system having a continuous time domain, the tick rules will often be of the form

$$[\mathit{tick}] : \ \{t\} \xrightarrow{x_r} \{t'(x_r)\} \textbf{ if } x_r \leq \mathit{mte}(t) \text{ and } C(t)$$

or otherwise of the exact same form, but replacing $x_r \leq mte(t)$ by $x_r < mte(t)$, where x_r is a variable not occurring in t and denotes the time advanced by the tick, $mte(t)$ computes the *maximum time elapse* permissible to ensure timeliness of time-critical actions, and the condition $x_r \leq mte(t)$ (resp. $x_r < mte(t)$) ensures that time elapse *may* halt temporarily for the *possible* application of a non-time-critical rule, that is, a rule modeling an action which could occur somewhat "arbitrarily" in time. The introduction of the variable x_r in the righthand side requires additional execution strategies for its instantiations, which is not surprising, since it models behavior which is nondeterministic in time. Allowing for real nondeterminism in timed behavior in this way may lead to Zeno behavior of the system and it is up to the execution strategy to instantiate the righthand side variable so as to avoid that, whenever possible.

2.5.2 Eager and lazy rules

In general, it is not sufficient to ensure that time elapse "stops" whenever necessary. Often, it must also be ensured that time does not advance again before all the necessary instantaneous actions are performed. In particular, an application of a rule often enables a number of instantaneous rules that must be taken immediately, and it must be ensured that all these actions are performed before time elapses again. A rule may, for example, produce a message which must be consumed before time advances again. In many cases it is possible to add conditions on the tick rules such that time will not elapse if some time-critical rule is enabled, but this may considerably complicate the specification. Instead of computing the enabledness condition of every time-critical rule explicitly, it is often more convenient to use the rewriting logic notion of *internal rewrite strategy* [12,10,15], whose execution is well supported by Maude's reflective features [11,14], to deal with these enabledness and priority aspects using a simple strategy.

The idea is to divide the rules in a real-time rewrite theory into *eager* and *lazy* rules and to let the *admissible* rewrites be those rewrites where the application of eager rules takes precedence over the application of lazy rules.

Definition 10 *Let $\mathcal{R}_{\phi,\tau}$ be a real-time theory with its set R of rewrite rules partitioned into disjoint sets R_{eager} and R_{lazy}. Then, the set of* admissible rewrites *in $\mathcal{R}_{\phi,\tau}$ are those rewrites $\alpha : t \longrightarrow t'$ in which lazy rules are applied sequentially and, furthermore, they are only applied when no eager rule is enabled. That is, a rewrite $\alpha : t \longrightarrow t'$ is admissible if and only if α is an identity proof or is equivalent to a factorization $\alpha_1; \ldots; \alpha_n$ of proof terms α_i corresponding to one-step* sequential *rewrites such that if some $\alpha_j : t_{j-1} \longrightarrow t_j$ is a proof of an application of a rule in R_{lazy}, then there is no one-step sequential rewrite $\beta : t_{j-1} \longrightarrow u$ applying a rule in R_{eager}.*

Tick rules and non-time-critical instantaneous rules should be lazy, which implies that time will not advance while some eager rule is enabled. Our treatment of timed Petri nets in Section 3.6 gives an example of the convenience of using this strategy.

Notation: Whenever an eager strategy should be used, the eager and lazy rules will be preceded by the keywords **eager** and **lazy**, respectively.

2.5.3 Time as an action on the whole system

When the state of a system has a rich structure, it may be both natural and necessary to have an explicit function denoting the effect of time elapse on the whole state. The function δ denoting the action of time on a system has the form

$$\delta : State\ Time \to State$$

involving the designated sorts *State* and *Time*. The action δ should be a *monoid action*, that is, it seems natural to require that it satisfies the axioms:

$$\delta(x, 0) = x$$
$$\delta(\delta(x, y_r), z_r) = \delta(x, y_r + z_r).$$

Tick rules should then be of the form

$$(\dagger) \qquad [tick]:\ \{t\} \xrightarrow{r} \{\delta(t, r)\} \textbf{ if } C.$$

Using the action δ to describe the effect of the passage of time on a dynamic evolution of a system is not without possible pitfalls. If done carelessly, it may allow "going back in time" to perform a rewrite. Suppose, for example, that $t = \delta(t', r)$ holds, and that the term t' rewrites to t''. Then, there would also be an "aged" rewrite $\{t\} =_E \{\delta(t', r)\} \longrightarrow \{\delta(t'', r)\}$, as illustrated in the following example.

Example 11 *The following specification is intended to specify a discrete time "clock" (that is, Time is the sort Nat of natural numbers) which can be reset when it has reached the value 24.*

sorts *State System*
op $clock : Nat \to State$
op $\delta : State\ Nat \to State$
op $\{\_\} : State \to System$

vars $x_r, y_r : Nat$
var $z : State$
eq $\delta(clock(x_r), y_r) = clock(x_r + y_r)$
rl $[reset] : clock(24) \longrightarrow clock(0)$
rl $[tick] : \{z\} \xrightarrow{x_r} \{\delta(z, x_r)\}$

However, starting from $\{clock(0)\}$, there is a rewrite sequence

$$\{clock(0)\} \overset{tick(\ldots,35)}{\longrightarrow} \{\delta(clock(0), 35)\} = \{clock(35)\} = \{\delta(clock(24), 11)\}$$
$$\longrightarrow \{\delta(clock(0), 11)\} = \{clock(11)\}$$

where the reset rule is applied at time 35. The problem is that the equation for δ and the reset rule are not coherent as explained below.

For executable specification purposes it is important to require that the set E of equations in a rewrite theory is divided into a set E' of simplifying equations and a set Ax of structural axioms, in such a way that the equations in E' define a Church-Rosser and terminating set of equations modulo the set Ax, and such that the set of rules R is *coherent* [12,43] w.r.t. $E' \uplus Ax$. A rewrite theory is coherent if for every one-step sequential rewrite $t \longrightarrow t_1$ modulo the structural axioms Ax, there is also a rewrite $t!_{E'} \longrightarrow t_1'$ modulo Ax, for $t!_{E'}$ the E'-normal form of t modulo Ax, such that t_1 and t_1' are E-equivalent. A coherent system in which ground terms of the form $\{\delta(t', r)\}$ reduce by the equations to terms of the form $\{t\}$ with no δ's does not allow "going back in time," in the sense that for any rewrite $t' \longrightarrow u'$ inducing a rewrite $\{\delta(t', r)\} \longrightarrow \{\delta(u', r)\}$ there must be an equivalent rewrite $\{t\} \longrightarrow \{w\}$ such that $\{\delta(u', r)\} =_{E' \cup Ax} \{w\}$.

A commonly occurring state structure for which we want the action of time to distribute over the different state components is a *multiset* distributed structure. For example, object-oriented systems and Petri nets have that structure. For multiset distributed systems we can give a general treatment of time actions that avoids lack of coherence problems.

A simple solution to avoid lack of coherence problems is to let each rule rewrite terms of sort *System* only, which would solve the coherence problem w.r.t. the symbol δ, since each rewrite would occur at the top. However, concurrency is lost by this solution. Our solution is to use special tokens of the form '$*$' and to let the extended state be a term in a supersort *ExtendedState* of the designated sort *State*, consisting of the multiset union of the original state and a multiset of tokens. The system operator $\{\_\}$ takes arguments of the sort *ExtendedState*, while δ is left unchanged, i.e., is a function $\delta : State\ \ Time \to State$. If multiset union is denoted by juxtaposition, the tick rules then take the form

$$(\ddagger) \qquad [tick] : \ \{T\ t\} \xrightarrow{r} \{T\ \delta(t, r)\} \textbf{ if } C,$$

for T a variable of a subsort $Tokens \leq ExtendedState$, denoting multisets of tokens, and t a term of sort $State$. Each local rule then has the form[6]

$$[l] : \ * \, t \longrightarrow * \cdots * \ t' \ \textbf{if} \ C.$$

Since one token appears in the lefthand side of each local rule, the global state must contain at least n tokens for n local rewrites to fire concurrently. For object-oriented systems, the number of tokens in a configuration could suitably equal the number of objects in a configuration, since the number of rewrites firing concurrently is bounded by the number of objects present in the global state, under the assumption that at least one object appears in the lefthand side of each rule. Coherence w.r.t. the symbol δ is now trivially unproblematic, since every instance of a lefthand side of a local rule has least sort $ExtendedState$, and therefore cannot be an argument of δ. This general approach to deal with time as an action on multiset distributed states is specialized to object-oriented systems in Section 3.5.2, and is illustrated with a multi-thermostat system example in Section 3.5.3.

To summarize, a monoid action δ denoting the effect of time elapse on the whole state may be useful for specifying real-time systems where the state of the system can have a rich distributed structure, but we must require coherence, since this ensures that δ does not cause counterintuitive rewrites resulting from "going back in time."

3 Specifying models of real-time and hybrid systems in rewriting logic

We show how some well-known models of real-time and hybrid systems can be naturally regarded as specializations of the real-time rewriting logic framework. Since we are interested in executable specifications, we place some computability restrictions on some models. Even though we do not present an exhaustive discussion of real-time models, we think that the models we have chosen are significantly varied and well-known to suggest that rewriting logic is a good semantic framework for real-time and hybrid systems.

[6] To ensure maximal possible concurrency in following rewrite steps, some tokens may have to be added to the righthand side of the rule if the rule increases the size of the state.

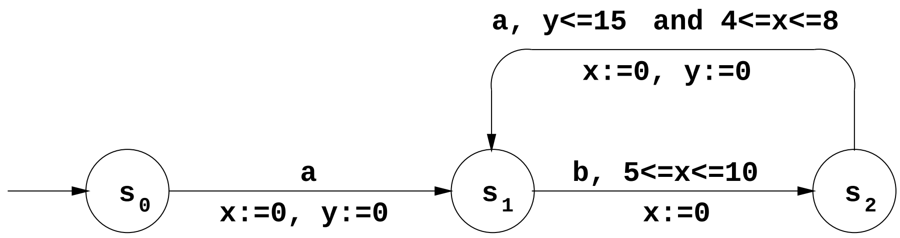

Fig. 1. A timed automaton.

3.1 Timed automata

We show how a *timed automaton* (see, e.g., [3]) can be specified in rewriting logic. A timed automaton reads timed words—infinite words where each symbol in a word has a time stamp—and is equipped with a set of clocks. A transition may reset some clocks to 0, and can only be applied if the values of the clocks satisfy the clock constraint of the transition. For example, the timed automaton in Fig. 1 "accepts" timed words with alternating occurrences of the input symbols a and b, starting with a, such that an occurrence of a is followed by an occurrence of b no earlier than time 5 and no later than time 10 after the occurrence of a, and each occurrence of b is followed by an a in some time in $[4, 8]$. Furthermore, each occurrence of a must be followed by another occurrence of a within time 15.

The time model of timed automata is the set of nonnegative real numbers. However, to get a computable data type, we can replace it by the set of nonnegative elements $\mathbb{R}_+$ of a computable subfield $\mathbb{R}$, such as the rationals or the algebraic real numbers. Omitting details about initial states and acceptance conditions, a timed automaton is given by a tuple (Σ, S, C, E) where:

- Σ is a finite alphabet;
- S is a finite set of states;
- C is a finite set of clocks, defining a set $\Phi(C)$ of clock constraints inductively by

$$\varphi ::= c \leq k \mid k \leq c \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2$$

  where c is a clock in C, and k is a constant in the set of nonnegative rationals, and
- E is a set $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ of transitions. The tuple $(s, s', a, \lambda, \varphi)$ represents a transition from state s to state s' on input symbol a. The set $\lambda \subseteq C$ gives the clocks to be reset with this transition, and φ is a clock constraint over C.

Given a timed word (i.e., an infinite sequence of tuples (a_i, r_i) where a_i is an input symbol and r_i is the time at which it occurs, and where $r_i \leq r_{i+1}$ for all $i \geq 0$), the automaton starts at time 0 with all clocks initialized to 0. As time advances, the values of all clocks change, reflecting the elapsed time; that is, the state of the automaton can change not only by the above transitions, but also by the passage of time, with all the clocks being increased by the elapsed time. At time r_i the automaton changes state from s to s' using some transition of the form $(s, s', a_i, \lambda, \varphi)$ reading input a_i, if the current values of the clocks satisfy φ. With this transition the clocks in λ are reset to 0, and start counting time again.

A *run* ρ of a timed automaton is an *infinite*[7] sequence

$$\rho:\ (s_0, v_0) \xrightarrow[\tau_1]{a_1} (s_1, v_1) \xrightarrow[\tau_2]{a_2} (s_2, v_2) \xrightarrow[\tau_3]{a_3} \cdots$$

with states $s_0, s_1, \ldots$, valuations $v_0, v_1, \ldots \in [C \to \mathbb{R}_+]$ such that $v_0(c) = 0$ for all $c \in C$, and such that for each $i \geq 1$, $\tau_{i-1} \leq \tau_i$ and there is a transition $(s_{i-1}, s_i, a_i, \lambda_i, \varphi_i)$ where the clock valuation $v_{i-1} + (\tau_i - \tau_{i-1})$ (with $\tau_0 \stackrel{\text{def}}{=} 0$) satisfies the clock constraint φ_i, and $v_i(c)$ is 0 if clock c is in λ_i and $v_i(c) = v_{i-1}(c) + (\tau_i - \tau_{i-1})$ otherwise. The set of all runs of a timed automaton A is denoted $\mathit{Runs}(A)$.

We assume in the remainder of this section that the set of clocks is ordered as a sequence $c_1, \ldots, c_n$. A timed automaton $A = (\Sigma, S, C, E)$ can then be naturally represented by a real-time rewrite theory $\Psi_{TA}(A) = ((\Sigma_A, E_A, L_A, R_A), \phi_A, \tau_A)$ as follows:

- (Σ_A, E_A) contains an equational axiomatization of the time domain $\mathbb{R}_+$. Furthermore, the signature Σ_A contains a sort *TAState* with a constant $s :\ \to \mathit{TAState}$ for each $s \in S$, a sort *State* with an $(n+1)$-ary operator $\_,\_,\ldots,\_ : \mathit{TAState}\ \mathit{Time}\ \ldots\ \mathit{Time} \to \mathit{State}$, and an operator $\{\_\} : \mathit{State} \to \mathit{System}$.
- The set L_A of labels is $\Sigma \cup \{\mathit{tick}\}$.
- The set of rules R_A and its associated duration assignment contains an instantaneous rule

$$[a] :\ \{s, x_1, \ldots, x_n\} \longrightarrow \{s', t_1, \ldots, t_n\}\ \textbf{if}\ \varphi(x_1, \ldots, x_n)$$

  for each transition $(s, s', a, \lambda, \varphi) \in E$, where the x_i's are variables of sort *Time*, where the term t_i is 0 if $c_i \in \lambda$ and $t_i = x_i$ otherwise, and where $\varphi(x_1, \ldots, x_n)$ is obtained from φ by substituting each clock c_i with x_i and by substituting $\wedge$ and $\neg$ with the Boolean operators "and" and "not". In addition, a rule

$$[\mathit{tick}] :\ \{z, x_1, \ldots, x_n\} \xrightarrow{y_r} \{z, x_1 + y_r, \ldots, x_n + y_r\}$$

[7] Runs are infinite, since timed words where the automaton cannot proceed upon reading a symbol with a timestamp can never be accepted by the automaton.

(where $z, y_r, x_1, \ldots, x_n$ are all variables) is added to model time elapse.

The timed automaton in Fig. 1 is therefore represented by a real-time rewrite theory containing the following rules:

$$\begin{array}{lrcl}
[a]: & \{s_0, x, y\} & \longrightarrow & \{s_1, 0, 0\} \\
[b]: & \{s_1, x, y\} & \longrightarrow & \{s_2, 0, y\} \textbf{ if } 5 \leq x \leq 10 \\
[a]: & \{s_2, x, y\} & \longrightarrow & \{s_1, 0, 0\} \textbf{ if } y \leq 15 \text{ and } 4 \leq x \leq 8 \\
[tick]: & \{z, x, y\} & \xrightarrow{y_r} & \{z, x + y_r, y + y_r\}
\end{array}$$

for x, y, and y_r variables of sort *Time*, and z a variable of sort *TAState*.

A *TA-step* in $\Psi_{TA}(A)$, which simulates one step in a run ρ of a timed automaton $A = (\Sigma, S, C, E)$, is a ground rewrite $\alpha; b(\overline{\beta}) : t \longrightarrow t'$ in $\Psi_{TA}(A)$, where $b \in \Sigma$ and α is a proof corresponding to a (possibly empty) sequence of tick applications. Since the passage of time r can be modeled by an arbitrary sequence of tick applications with total time elapse r, we consider two TA-steps $\alpha; b(\overline{\beta}) : t \longrightarrow t'$ and $\alpha'; b'(\overline{\beta'}) : u \longrightarrow u'$ to be *equivalent* if and only if $t = u$, $t' = u'$, $\tau(\alpha) = \tau(\alpha')$, and $b = b'$ hold. Together, these conditions imply $\beta = \beta'$. A TA-step $\alpha; b(\overline{\beta}) : t \longrightarrow t'$ is, therefore, uniquely determined by t, t', $\tau(\alpha)$, and b, and will be written $t \xrightarrow[\tau(\alpha)]{b} t'$.

A *TA-computation* of the rewriting logic representation $\Psi_{TA}(A)$ of a timed automaton A is an infinite sequence

$$\tilde{\rho} : \{t_0, 0, \ldots, 0\} \xrightarrow[r_1]{b_1} \{t_1, r_{1_1}, \ldots, r_{1_n}\} \xrightarrow[r_2]{b_2} \{t_2, r_{2_1}, \ldots, r_{2_n}\} \xrightarrow[r_3]{b_3} \cdots$$

of TA-steps in $\Psi_{TA}(A)$. The set of all TA-computations in $\Psi_{TA}(A)$ (modulo TA-step equivalence) is denoted $TA\text{-}\mathcal{C}(\Psi_{TA}(A))$.

Theorem 12 *Let $A = (\Sigma, S, C, E)$ be a timed automaton, and let $\Psi_{TA}(A)$ be its rewriting logic translation. Then, there exists a bijective function $\tilde{(\_)}$: $Runs(A) \rightarrow TA\text{-}\mathcal{C}(\Psi_{TA}(A))$ assigning to each run of A of the form ρ defined above a TA-computation of the form $\tilde{\rho}$ with $s_0 = t_0$, $s_i = t_i$, $\tau_i - \tau_{i-1} = r_i$, $a_i = b_i$, and $r_{i_j} = v_i(c_j)$ for each $i \geq 1$ and $j \in \{1, \ldots, n\}$.*

Proof sketch: The theorem follows from the fact that

$$(s_i, v_i) \xrightarrow[\tau_i + \Delta_i]{a_{i+1}} (s_{i+1}, v_{i+1})$$

is a "transition step" in A for each $i \geq 0, \Delta_i, \tau_i, s_i, s_{i+1}, v_i, v_{i+1}$ if and only if there is a TA-step

$$\{s_i, r_{i_1}, \ldots, r_{i_n}\} \xrightarrow[\Delta_i]{a_{i+1}} \{s_{i+1}, r_{i+1_1}, \ldots, r_{i+1_n}\}$$

in $\Psi_{TA}(A)$ with $r_{j_k} = v_j(c_k)$ for $j = i, i+1$.

Using the equational axioms defining equivalence of rewrite proofs in rewriting logic [29], one can prove

Proposition 13

- *Each non-identity ground rewrite proof $\gamma : t \longrightarrow t'$ in $\Psi_{TA}(A)$ can be decomposed as a sequence $\gamma = \xi_1; \cdots; \xi_n; \delta_1; \cdots; \delta_m$, with $n, m \geq 0$ and $n+m \geq 1$, where $\xi_1, \ldots, \xi_n$ are all TA-steps and the $\delta_1, \ldots, \delta_m$ are all tick applications.*
- *Any infinite computation of $\Psi_{TA}(A)$ (see Section 2.2), starting with a term of the form $\{t_0, 0, \ldots, 0\}$, and involving an infinite number of instantaneous rule applications, can be rearranged in the form of a TA-computation $\tilde{\rho}$ by an appropriate composition of the arrows.*

There are at least two ways of modifying the specification to simulate the behavior of the automaton on only those timed words satisfying a given acceptance condition. It is possible to define a computable predicate *has_computation*, so that *has_computation*$(s, r_1, \ldots, r_n)$ holds if and only if there exists an *accepted* timed word "starting" in state s with (rational-numbered) values $r_1, \ldots, r_n$ of the clocks $c_1, \ldots, c_n$ (such a predicate is computable, and therefore finitely specifiable by Church-Rosser and terminating equations [6], since defining such a predicate reduces to the emptiness problem for timed automata which is decidable [3]). In this way, we obtain a rewrite theory whose computations simulate the behavior of the automaton on *accepted* timed words by adding the condition **if** *has_computation*$(t', t_1, \ldots, t_n)$ to every rule of the form $[l] : \ \{t, x_1, \ldots, x_n\} \longrightarrow \{t', t_1, \ldots, t_n\}$, including the tick rule.

A more modular, alternative way of restricting the rewrites to simulate automata behavior on accepted words only would be to encode the accepting states (or sets of states for Muller-automata) as predicates in the rewrite theory, and then use the internal strategies at the metalevel of rewriting logic to restrict the application of the rules, so that only accepted timed words are executed.

3.2 Hybrid Automata

A *hybrid automaton* [2] (also called a *hybrid system*) is a finite automaton equipped with variables that evolve continuously with time according to dy-

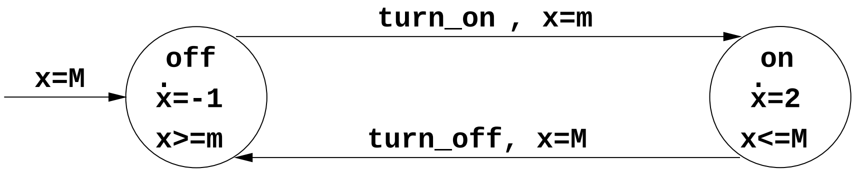

Fig. 2. A hybrid automaton model of a thermostat.

namical laws, and where each location is labeled with an invariant condition that must hold when control resides at that location. The hybrid automaton in Fig. 2 describes a simple thermostat which keeps the temperature in a room between m and M by turning a heater on and off. In our simplified model, the temperature in the room increases by 2 degrees per time unit when the heater is turned on, and decreases by 1 degree per time unit otherwise.

The time model of hybrid automata is the set of nonnegative real numbers, but we replace it by the set $\mathbb{R}_+$ of nonnegative elements in a computable subfield $\mathbb{R}$, as we did for timed automata. A hybrid automaton is given by a tuple $(V_D, Loc, Lab, Act, Inv, Edg)$ where:

- V_D is a finite set of data variables, each ranging over a given data sort, defining the data space Σ_D, that is, Σ_D is the set of sort-consistent valuations v of V_D.
- *Loc* is a finite set of *locations* (corresponding to "states" in untimed automata).
- The state space of a hybrid automaton is $Loc \times \Sigma_D$.
- *Lab* is a set of *synchronization labels*, including the label τ.
- *Act* is a labeling function that assigns to each location $l \in Loc$ a set Act_l of *activities*. An activity is a function from $\mathbb{R}_+$ to Σ_D. For each activity f in Act_l and each time value r there is an activity f^r in Act_l defined by $f^r(r') = f(r + r')$.
- *Inv* is a labeling function that assigns to each location $l \in Loc$ an *invariant* $Inv(l) \subseteq \Sigma_D$.
- *Edg* is a finite set of *transitions*. Each transition $e = (l, \mu, l', a)$ consists of a source location l, a target location l', a transition relation $\mu \subseteq \Sigma_D^2$, and a synchronization label a. For each location l there is a *stutter transition* (l, Id, l, τ) where $Id = \{(v, v) \mid v \in \Sigma_D\}$.

The hybrid automaton in Fig. 2 has two locations, *on* and *off*, one ($\mathbb{R}_+$-valued) data variable x denoting the temperature in the room, and two non-stutter transitions $(\mathit{off}, (\{x \mapsto m\}, \{x \mapsto m\}), \mathit{on}, \mathit{turn\_on})$ and $(\mathit{on}, (\{x \mapsto M\}, \{x \mapsto M\}), \mathit{off}, \mathit{turn\_off})$. For location *on*, $Inv(\mathit{on}) = \{\, \{x \mapsto r\} \mid r \in \mathbb{R}_+ \wedge r \leq M\}$ and Act_{on} is the set of functions $\{f_r : \mathbb{R}_+ \to (\{x\} \to \mathbb{R}) \mid r \in \mathbb{R}\}$ defined by $f_r(r') = \{x \mapsto r + 2r'\}$. The invariant and the activities of location *off* are entirely similar.

The state of a hybrid automaton can change in two ways: (1) by an instantaneous transition that changes the entire state according to the transition relation, or (2) by elapse of time that changes only the values of data variables in a continuous manner, according to the activities of the current location, where the state (l, v) evolves to $(l, f(r))$ in time r whenever f is an activity of location l such that $v = f(0)$. The system may stay at a location only if the invariant at that location remains true. The invariants of a hybrid automaton thus enforce the progress of the underlying discrete transition system. That is, some transition must be taken before the invariant of the location becomes false.

A run ϱ of a hybrid automaton is a finite or infinite sequence

$$\varrho : \ (l_0, v_0) \overset{a_0\ r_0}{\mapsto}_{f_0} (l_1, v_1) \overset{a_1\ r_1}{\mapsto}_{f_1} (l_2, v_2) \overset{a_2\ r_2}{\mapsto}_{f_2} \cdots$$

where $l_0, l_1, \ldots$ denote locations, $v_0, v_1, \ldots$ denote valuations of the variables V_D, $a_0, a_1, \ldots$ denote synchronization labels, $r_0, r_1, \ldots$ denote time values, and where each f_i is either the constant id or an activity in location l_i, such that the following conditions hold: For all i, $v_{i+1} \in Inv(l_{i+1})$; furthermore, whenever f_i is id, we have $r_i = 0$, $v_i \in Inv(l_i)$, and $(v_i, v_{i+1}) \in \mu_i$ for some $(l_i, \mu_i, l_{i+1}, a_i) \in Edg$; and whenever f_i is an activity we have $f_i(0) = v_i$, $f_i(r) \in Inv(l_i)$ for all $0 \le r \le r_i$, and $(f_i(r_i), v_{i+1}) \in \mu_i$ for some $(l_i, \mu_i, l_{i+1}, a_i) \in Edg$. The set of all such runs of a hybrid automaton A is denoted $Runs(A)$, and $Runs_{Inv}(A)$ denotes the subset of these runs where $v_0 \in Inv(l_0)$. This definition of a run is a slight modification of the one appearing in [2], in that it exhibits the transition used at each step, uses the special symbol id to allow a transition to be taken in zero time from a state v_j, satisfying $Inv(l_j)$, even when there is no time-advancing activity acting on the state, and adds the requirement $v_{i+1} \in Inv(l_{i+1})$ to ensure that the final state in a run satisfies the invariant of its location.

In the remainder of this section, we assume that the set V_D of data variables is ordered as a sequence $x_1, \ldots, x_n$. Since the definition of hybrid automata is very general, we restrict our treatment to a subclass of hybrid automata satisfying some natural requirements. First of all, we require that the set of activities Act_l for a location l is generated by a finite set

$$ActGen_l = \{f_i^l : \Sigma_D \times \mathbb{R}_+ \to \Sigma_D \ \mid \ 1 \le i \le n_l\}$$

of computable functions, called activity generators, where each f_i^l satisfies the property

$$f_i^l(f_i^l(v, r), r') = f_i^l(v, r + r') \textbf{ if } f_i^l(v, 0) = v.$$

In the thermostat example, $ActGen_{on} = \{g\}$ and $ActGen_{off} = \{h\}$ for the functions g and h defined by $g(\{x \mapsto r\}, r') = \{x \mapsto r + 2r'\}$ and $h(\{x \mapsto r\}, r') = \{x \mapsto r - r'\}$.

The set Act_l of activities for a location l is generated from $ActGen_l$ as follows,

$$Act_l = \{f : \mathbb{R}_+ \to \Sigma_D \mid (\exists f_i^l \in ActGen_l, v \in \Sigma_D, r \in \mathbb{R}_+) \\ f_i^l(v, 0) = v \wedge f = \lambda x_r . f_i^l(v, r + x_r)\}.$$

Furthermore, we require that for each location l and activity generator $f_i^l \in ActGen_l$, there is a computable function

$$max\_stay_{f_i^l} : \Sigma_D \to \mathbb{R}_+ \cup \{\infty\}$$

where $max\_stay_{f_i^l}(v)$ denotes the amount of time a system in state (l, v) can stay at location l performing the activity-function f_i^l, without violating the invariant of location l. We also require that there is a computable predicate

$$Inv : Loc \times \Sigma_D \to Bool$$

where $Inv(l, v)$ holds if and only if the state v does not violate the invariant of location l. Although not necessary, we also find it natural to assume that *all* τ-transitions are identity transitions, that is, $(l, \mu, l', \tau) \in Edg$ implies $l = l'$ and $\mu = Id$.

We furthermore require that each data type of A is computable. Before stating our last requirement about the transition of the hybrid automaton A, we explain some necessary details about its associated rewrite theory $\Psi_{HA}(A)$.

A hybrid automaton $A = (V_D, Loc, Lab, Act, Inv, Edg)$ with the variables ordered into $x_1, \ldots, x_n$ and satisfying the above requirements can be represented by a real-time rewrite theory $\Psi_{HA}(A) = ((\Sigma_A, E_A, L_A, R_A), \phi_A, \tau_A)$ as follows.

- The signature Σ_A contains a sort *Location* with a constant $l : \to Location$ for each $l \in Loc$, a sort s_i for each data sort s_i of the variable x_i, a sort *Valuation* with an n-ary operator $\langle \_, \ldots, \_ \rangle : s_1 \ldots s_n \to Valuation$, an operator $\_, \_ : Location\ Valuation \to State$, and an operator $\{\_\} : State \to System$. Furthermore, (Σ_A, E_A) contains finitary axiomatizations of the data types of the sorts $s_1, \ldots, s_n$, and of the time domain, so that $\phi_A(Time)$ specifies $\mathbb{R}_+$ and ϕ_A has an extension $\phi'_A : LTIME_\infty \to (\Sigma_A, E_A)$, of the sets of functions $f_i^l : Valuation\ Time \to Valuation$ and $max\_stay_{f_i^l} : Valuation \to Time_\infty$, and of the function $Inv : Location\ Valuation \to Bool$. These functions and data types can be given such finitary equational axiomatizations since they are assumed computable [6].

- The set L_A of labels is $(Lab \setminus \{\tau\}) \cup \{tick_{f_i^l} \mid l \in Loc \wedge f_i^l \in ActGen_l\}$, where $(Lab \setminus \{\tau\}) \cap \{tick_{f_i^l} \mid l \in Loc \wedge f_i^l \in ActGen_l\} = \emptyset$, and $tick_{f_i^l} \neq tick_{f_j^{l'}}$ if $f_i^l \neq f_j^{l'}$.
- Our last assumption about the hybrid automaton A is that its transitions can be specified by rewrite rules. That is, we require that each transition $(l, \mu, l', a) \in Edg$ can be expressed by a finite set of rewrite rules of the form $[a] : \{l, \langle t_1, \ldots, t_n \rangle\} \longrightarrow \{l', \langle t_1', \ldots, t_n' \rangle\}$ **if** C, so that we have $(\{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}, \{x_1 \mapsto u_1', \ldots, x_n \mapsto u_n'\}) \in \mu$ if and only if $a(\ldots) : \{l, \langle u_1, \ldots, u_n \rangle\} \longrightarrow \{l', \langle u_1', \ldots, u_n' \rangle\}$ is a ground one-step rewrite using these rules. Then, the set R_A of rewrite rules in $\Psi_{HA}(A)$ and its associated duration assignment function τ_A contains the corresponding rule(s)

  $$[a] : \ \{l, \langle t_1, \ldots, t_n \rangle\} \longrightarrow \{l', \langle t_1', \ldots, t_n' \rangle\} \ \textbf{if} \ C \wedge Inv(l', \langle t_1', \ldots, t_n' \rangle) = true$$

  for each transition (l, μ, l', a) in Edg with $a \neq \tau$, where the last conjunct in the condition must be added to the translation of the (underlying "untimed") transition to ensure that the resulting state satisfies the invariant of location l'. The tick rules of the system associate to each location l and each activity generator f_i^l a rewrite rule of the form

  $$[tick_{f_i^l}] : \ \{l, V\} \xrightarrow{x_r} \{l, f_i^l(V, x_r)\} \ \textbf{if} \ x_r \leq max\_stay_{f_i^l}(V) \text{ and } f_i^l(V, 0) = V,$$

  where V is a variable of sort *Valuation*.

A rewriting logic translation of the thermostat in Fig. 2 may instantiate this translation schema in the following way. The domains $\mathbb{R}_+$ and $\mathbb{R}$ are interpreted by sorts *Time* and *Temp* with the necessary functions, and the rest of the theory is given by the following declarations:

sorts *Location Valuation State System*
ops $on \ off \ : \ \to Location$
op $\langle \_ \rangle : Temp \longrightarrow Valuation$
op $\_,\_ : Location \ \ Valuation \to State$
op $\{\_\} : State \to System$
ops $g \ h : Valuation \ \ Time \to Valuation$
op $Inv : Location \ \ Valuation \to Bool$
ops $max\_stay_{on} \ \ max\_stay_{off} \ : \ Valuation \to Time_{\infty}$
var $x : Temp$ **var** $y_r : Time$ **var** $v : Valuation$
eq $Inv(on, \langle x \rangle) = x \leq M$
eq $Inv(off, \langle x \rangle) = x \geq m$
eq $max\_stay_{on}(\langle x \rangle) = (M - x)/2$
eq $max\_stay_{off}(\langle x \rangle) = x - m$
eq $g(\langle x \rangle, y_r) = \langle x + 2 * y_r \rangle$
eq $h(\langle x \rangle, y_r) = \langle x - y_r \rangle$
crl $[turn\_on] : \ \{off, \langle m \rangle\} \longrightarrow \{on, \langle m \rangle\}$ **if** $Inv(on, \langle m \rangle)$

$$
\begin{array}{lll}
\textbf{crl} & [\mathit{turn\_off}] : & \{on, \langle M\rangle\} \longrightarrow \{\mathit{off}, \langle M\rangle\} \ \textbf{if}\ \mathit{Inv}(on, \langle M\rangle) \\
\textbf{crl} & [\mathit{tick}_{on}] : & \{on, v\} \xrightarrow{y_r} \{on, g(v, y_r)\} \\
 & & \qquad\qquad \textbf{if}\ y_r \leq \mathit{max\_stay}_{on}(v) \text{ and } g(v, 0) == v \\
\textbf{crl} & [\mathit{tick}_{\mathit{off}}] : & \{\mathit{off}, v\} \xrightarrow{y_r} \{\mathit{off}, h(v, y_r)\} \\
 & & \qquad\qquad \textbf{if}\ y_r \leq \mathit{max\_stay}_{\mathit{off}}(v) \text{ and } h(v, 0) == v
\end{array}
$$

An *HA-step* in $\Psi_{HA}(A)$ is intended to model a step in A and has one of the following forms:

- a ground rewrite of the form $\mathit{tick}_{f_k^l}(v, r); a(\overline{w}) : \{l, v\} \longrightarrow \{l', v'\}$, with $a \in \mathit{Lab} \setminus \tau$, which models a step $(l, v) \overset{a}{\mapsto}{}^{r}_{(\lambda r'.f_k^l(v,r'))} (l', v')$ in A;
- a ground rewrite of the form $a(\ldots) : t \longrightarrow t'$, which models a step $t \overset{a}{\mapsto}{}^{0}_{id}\, t'$ in A;
- a ground rewrite of the form $\mathit{tick}_{f_k^l}(t, r) : \{l, t\} \longrightarrow \{l, t'\}$, which models a step $(l, t) \overset{\tau}{\mapsto}{}^{r}_{(\lambda r'.f_k^l(t,r'))} (l', t')$; or
- an identity rewrite $t : t \longrightarrow t$ for t a ground term of sort *System*, which models a step $t \overset{\tau}{\mapsto}{}^{0}_{id}\, t$ in A.

Since an activity may be generated by different activity generators, we *identify* the HA-steps $\mathit{tick}_{f_k^l}(u, r) : \{l, u\} \longrightarrow \{l, u'\}$ and $\mathit{tick}_{f_j^l}(u, r) : \{l, u\} \longrightarrow \{l, u'\}$ whenever $f_k^l(u, r') = f_j^l(u, r')$ for all r'. Furthermore, a transition a in A could be modeled by more than one rule in $\Psi_{HA}(A)$, although the choice of *which* a-rule to apply should not make two different a-rewrites from some t to t' into different steps. We therefore identify the HA-steps of the form $a(\overline{w}) : t \longrightarrow t'$ and $a(\overline{w'}) : t \longrightarrow t'$. These two identifications of HA-steps extend to composite HA-steps, making the HA-steps $\mathit{tick}_{f_j^l}; a(\overline{w}) : \{l, u\} \longrightarrow \{l', u'\}$ and $\mathit{tick}_{f_k^l}; a(\overline{w'}) : \{l, u\} \longrightarrow \{l', u'\}$ *equivalent* HA-steps if and only if $f_j^l(u, r') = f_k^l(u, r')$ for all r'.

An *HA-computation* in $\Psi_{HA}(A)$ is a finite or infinite sequence

$$\tilde{\varrho} : \ \{l_0, \langle t_{0_1}, \ldots, t_{0_n}\rangle\} \xrightarrow{\alpha_0} \{l_1, \langle t_{1_1}, \ldots, t_{1_n}\rangle\} \xrightarrow{\alpha_1} \{l_2, \langle t_{2_1}, \ldots, t_{2_n}\rangle\} \xrightarrow{\alpha_2} \cdots$$

of HA-steps $\alpha_i : \{l_i, \langle t_{i_1}, \ldots, t_{i_n}\rangle\} \longrightarrow \{l_{i+1}, \langle t_{i+1_1}, \ldots, t_{i+1_n}\rangle\}$ in $\Psi_{HA}(A)$, where $\mathit{Inv}(l_0, \langle t_{0_1}, \ldots, t_{0_n}\rangle) = \mathit{true}$. The set of all HA-computations in $\Psi_{HA}(A)$ (modulo HA-step equivalence) is denoted $\mathit{HA}\text{-}\mathcal{C}(\Psi_{HA}(A))$.

Theorem 14 *Let $A = ((x_1, \ldots, x_n), \mathit{Loc}, \mathit{Lab}, \mathit{Act}, \mathit{Inv}, \mathit{Edg})$ be a hybrid automaton satisfying our additional requirements. Then, there exists a bijective function $\tilde{(\_)} : \mathit{Runs}_{\mathit{Inv}}(A) \to \mathit{HA}\text{-}\mathcal{C}(\Psi_{HA}(A))$ which takes each run of the form ϱ in $\mathit{Runs}_{\mathit{Inv}}(A)$ into an HA-computation of the form $\tilde{\varrho}$ where, for each appropriate i and each $j \in \{1, \ldots, n\}$,*

- $t_{i_j} = v_i(x_j)$,

- $\tau_A(\alpha_i) = r_i$,
- $\alpha_i = tick_{f_k^{l_i}}(\langle t_{i_1}, \ldots, t_{i_n}\rangle, r_i); a_i(\overline{w})$ *if* $a_i \neq \tau$ *and* $f_i \neq id$, *for some* $\overline{w}$ *and some* $f_k^{l_i} \in ActGen_{l_i}$ *satisfying* $f_k^{l_i}(v_i, r') = f_i(r')$ *for all* r',
- $\alpha_i = a_i(\overline{w})$ *for some* $\overline{w}$ *if* $a_i \neq \tau$ *and* $f_i = id$,
- $\alpha_i = tick_{f_k^{l_i}}(\langle t_{i_1}, \ldots, t_{i_n}\rangle, r_i)$ *for some* $f_k^{l_i} \in ActGen_{l_i}$ *found as above, if* $a_i = \tau$ *and* $f_i \neq id$,
- α_i *is the identity step if* $a_i = \tau$ *and* $f_i = id$.

Furthermore, each computation in $\Psi_{HA}(A)$ *starting in a state satisfying the invariant of its location, and each prefix of such a computation, corresponds to (at least) one HA-computation in* $HA\text{-}\mathcal{C}(\Psi_{HA}(A))$.

Proof sketch: It is easy to see that $\tilde{(\_)}$ induces a bijection between the set of initial states in $Runs_{Inv}(A)$ and the set of initial states of HA-computations. The theorem and definition of HA-steps indicate how each step in the run ϱ is in one-to-one-correspondence with an HA-step in $\tilde{\varrho}$. That, say, $(l, v) \overset{a}{\mapsto}{}^{r}_{f} (l', v')$ is mapped into a *unique* HA-step $tick_{f_k^l}(v, r); a(\overline{w}) : \{l, v\} \longrightarrow \{l', v'\}$ modulo the equivalence on HA-steps follows from the key property that $f \in Act_l$ implies that there is an activity generator f_k^l such that $f(r) = f_k^l(f(0), r)$ for all r. Conversely, an application $tick^l_{f_k^l}(v, r)$ is the image under $\tilde{(\_)}$ of a step $(l, v) \overset{\tau}{\mapsto}{}^{r}_{f} (l, f(r))$ in A. The key property here is that the condition of the tick rule implies $f_k^l(v, 0) = v$ for the state $\{l, v\}$ being rewritten, and by the definition of the relationships between activities and activity generators, there is an activity $f \in Act_l$ defined by $f(r') = f_k^l(v, r')$ for all r'. Uniqueness follows directly from the definitions.

Each computation (see Section 2.2), or prefix thereof, in $\Psi_{HA}(A)$, which starts with an initial state of sort *System* and where the initial valuation satisfies the invariant of the state's location, is trivially an HA-computation, since each one-step rewrite is also an HA-step.

3.3 Timed transition systems

A *timed transition system* (TTS) [25] is a transition system where each transition is equipped with a lower bound, denoting the shortest time the transition must be continuously enabled before being taken, and an upper bound, denoting the longest time the transition can be continuously enabled without being taken. Formally, a TTS is a tuple (X, T, l, u) where X is a finite set $\{x_1, \ldots, x_n\}$ of data variables (with each x_i ranging over a data sort s_i) defining the state space Σ_D of all sort-consistent valuations v of the variables, T a finite set of transitions $a : \Sigma_D \to 2^{\Sigma_D}$, and l and u are functions $l : T \to \mathbb{R}_+$

and $u : T \to \mathbb{R}_+ \cup \{\infty\}$, which assign to each transition a a "lower bound" l_a and an "upper bound" u_a where $0 \leq l_a \leq u_a$.[8] All transitions are assumed to be *self-disabling*, i.e., $v' \in a(v)$ implies $a(v') = \emptyset$ for all $a \in T$ and $v, v' \in \Sigma_D$.

For example, a system where a variable x ranges over $\{a, b\}$, and changes from a to b at some time $r \in [5, 10]$ after changing from b to a, and then changes back to a at some time $r' \in [4, 8]$ thereafter, can be modeled by a TTS $\mathcal{T}_{ab}$ with transitions $\{\{x \mapsto a\} \mapsto \{\{x \mapsto b\}\}, \{x \mapsto b\} \mapsto \emptyset\}$, having lower bound 5 and upper bound 10, and $\{\{x \mapsto b\} \mapsto \{\{x \mapsto a\}\}, \{x \mapsto a\} \mapsto \emptyset\}$, having lower bound 4 and upper bound 8.

A *computation* of a TTS $\mathcal{T} = (X, T, l, u)$ is an infinite sequence

$$\sigma : \ (v_0, r_0) \to (v_1, r_1) \to (v_2, r_2) \to \cdots$$

of pairs of valuations v_i and time values r_i (with $r_0 = 0$) such that for each $i \geq 0$, either $v_i = v_{i+1} \wedge r_i < r_{i+1}$, or $r_i = r_{i+1} \wedge v_{i+1} \in a(v_i)$ holds for some transition a in T which has been enabled uninterruptedly for at least time l_a, that is, there is a $j \leq i$ such that $r_i - r_j \geq l_a$ and a is enabled on all the states $v_j, v_{j+1}, \ldots, v_i$. Furthermore, if a transition b is enabled on a state v_i, it must be disabled on some state v_{i+k} with $r_{i+k} \leq r_i + u_b$. Finally, it is required that r_j grows beyond any bound as j increases. The set of all computations of a TTS $\mathcal{T}$ is denoted by $\mathit{Comps}(\mathcal{T})$.

Again, we let the time domain be a computable subfield of the nonnegative reals and assume that the underlying untimed transition system can be finitely specified in rewriting logic in such a way that all data types are computable, and each transition can be specified by a finite set of rewrite rules in a way that will be made more precise after giving some necessary information about the translation. We also assume that, for each transition a, it is possible to define a computable predicate $\mathit{enabled}_a$ such that $\mathit{enabled}_a(v)$ is true if and only if transition a is enabled on state v, i.e., $\mathit{enabled}_a(v) \Leftrightarrow \exists v' \, (v' \in a(v))$.

We model a TTS in rewriting logic by representing the system state by a term $\{\langle t_1, \ldots, t_n \rangle, c_1, \ldots, c_m\}$, where the term $\langle t_1, \ldots, t_n \rangle$ represents a valuation $v = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ of the transition system variables, and where each c_i is a "clock" value which is nil if transition a_i is not enabled on the state, and is r_i if the transition a_i has been enabled continuously for time r_i. More precisely, let $\mathcal{T}$ be a timed transition system $(\{x_1, \ldots, x_n\}, T, l, u)$ satisfying the above requirements and where the set T of timed transitions is ordered as a sequence $a_1, \ldots, a_m$. Then, its rewriting logic translation $\Psi_{TTS}(\mathcal{T}) =$

[8] We ignore that a TTS is also equipped with a set of *initial* states.

$((\Sigma_\mathcal{T}, E_\mathcal{T}, L_\mathcal{T}, R_\mathcal{T}), \phi_\mathcal{T}, \tau_\mathcal{T})$ is the following real-time rewrite theory:

- $(\Sigma_\mathcal{T}, E_\mathcal{T})$ contains the specification of the sorts $s_1, \ldots, s_n$ of the variables in $\mathcal{T}$ and of the time domain such that the equational theory morphism $\phi_\mathcal{T} : \mathit{TIME} \to (\Sigma_\mathcal{T}, E_\mathcal{T})$ can be extended to an equational theory morphism $\phi'_\mathcal{T} : \mathit{TIME}_\infty \to (\Sigma_\mathcal{T}, E_\mathcal{T})$. In addition, $\Sigma_\mathcal{T}$ contains the sorts *Valuation*, *State*, *System*, and *Time*?, a subsort declaration $\mathit{Time} \leq \mathit{Time?}$, a constant $\mathit{nil} : \to \mathit{Time?}$, an n-ary operator $\langle \_, \ldots, \_ \rangle : s_1 \ldots s_n \to \mathit{Valuation}$, an $(m+1)$-ary operator $\_, \ldots, \_ : \mathit{Valuation}\ \mathit{Time?} \ldots \mathit{Time?} \to \mathit{State}$, an operator $\{\_\} : \mathit{State} \to \mathit{System}$, and declarations $\_ \leq \_ : \mathit{Time?}\ \mathit{Time}_\infty \to \mathit{Bool}$ and $\_+\_ : \mathit{Time?}\ \mathit{Time} \to \mathit{Time?}$. We also add to $E_\mathcal{T}$ the axioms $\mathit{nil}+x_r = \mathit{nil}$ for $x_r : \mathit{Time}$, and $\mathit{nil} \leq y_r = \mathit{true}$ for $y_r : \mathit{Time}_\infty$. Finally, $(\Sigma_\mathcal{T}, E_\mathcal{T})$ axiomatizes the predicate $\mathit{enabled}_a : \mathit{Valuation} \to \mathit{Bool}$ for each transition a in T.
- For the set $L_\mathcal{T}$ of labels, it is enough to choose the set $\{\mathit{trans}, \mathit{tick}\}$, since computations in a TTS are independent of the choice of transitions taken.
- Before giving the rules $R_\mathcal{T}$, we make more precise our assumption about being able to represent the transitions in T as rewrite rules by assuming that each (untimed) transition $a_i \in T$ can be modeled by a finite set of rewrite rules of the form $\langle t_1, \ldots, t_n \rangle \longrightarrow \langle t'_1, \ldots, t'_n \rangle$ **if** C, such that $\{x_1 \mapsto w'_1, \ldots, x_n \mapsto w'_n\} \in a_i(\{x_1 \mapsto w_1, \ldots, x_n \mapsto w_n\})$ holds if and only if $\langle w_1, \ldots, w_n \rangle \longrightarrow \langle w'_1, \ldots, w'_n \rangle$ is a one-step ground rewrite using one of these rules. The set $R_\mathcal{T}$ of rules together with its duration assignment $\tau_\mathcal{T}$ can then be given as follows. Each transition a_i is modeled in the *timed* system $\Psi_{TTS}(\mathcal{T})$ by the corresponding instantaneous rewrite rule(s)

$$[\mathit{trans}] : \ \{\langle t_1, \ldots, t_n \rangle, c_1, \ldots, c_m\} \longrightarrow \{\langle t'_1, \ldots, t'_n \rangle, c'_1, \ldots, c'_m\} \quad \textbf{if } C \wedge (l_{a_i} \leq c_i)$$

  where $\{c_1, \ldots, c_m\}$ is a set of m distinct variables of sort *Time*? not occurring in any t_i or t'_i, and each c'_j is an abbreviation for the expression

  if not$(\mathit{enabled}_{a_j}(\langle t'_1, \ldots, t'_n \rangle))$ **then** nil **else if** $c_j == \mathit{nil}$ **then** 0 **else** c_j **fi fi**.

  In addition, $R_\mathcal{T}$ contains the following tick rule, which ensures, for each transition a_i, that time will not elapse past the moment when a_i would have been enabled for time u_{a_i}:

$$[\mathit{tick}] : \{V, c_1, \ldots, c_m\} \xrightarrow{x_r} \{V, c_1 + x_r, \ldots, c_m + x_r\} \quad \textbf{if } 0 < x_r \wedge \textstyle\bigwedge_i (c_i + x_r \leq u_{a_i}),$$

  for V a variable of sort *Valuation* and x_r a variable of sort *Time*.

The real-time rewrite theory $\Psi_{TTS}(\mathcal{T}_{ab})$ of the example system $\mathcal{T}_{ab}$ consists of the following rules, which are simplified in that we do not use the *enabled*-

predicate explicitly, and where the transition from a to b is the first transition in the ordering used in the construction of $\Psi_{TTS}(\mathcal{T}_{ab})$:

$$
\begin{array}{lrcl}
[\mathit{trans}]: & \{\langle a\rangle, x, y\} & \longrightarrow & \{\langle b\rangle, \mathit{nil}, 0\} \ \textbf{if}\ 5 \leq x \\
[\mathit{trans}]: & \{\langle b\rangle, x, y\} & \longrightarrow & \{\langle a\rangle, 0, \mathit{nil}\} \ \textbf{if}\ 4 \leq y \\
[\mathit{tick}]: & \{v, x, y\} & \xrightarrow{x_r} & \{v, x + x_r, y + x_r\} \\
 & & & \quad \textbf{if}\ 0 < x_r \wedge x + x_r \leq 10 \wedge y + x_r \leq 8
\end{array}
$$

for variables x, y of sort *Time?*, v of sort *Valuation*, and x_r of sort *Time*.

A *TTS-computation* in $\Psi_{TTS}(\mathcal{T})$ is an infinite sequence

$$\tilde{\sigma}:\ \{\langle t_{0_1}, \ldots, t_{0_n}\rangle, c_{0_1}, \ldots, c_{0_m}\} \xrightarrow{\alpha_1} \{\langle t_{1_1}, \ldots, t_{1_n}\rangle, c_{1_1}, \ldots, c_{1_m}\} \xrightarrow{\alpha_1} \cdots$$

of one-step ground rewrites in $\Psi_{TTS}(\mathcal{T})$ where the sum $\Sigma_{i=1}^{j}\tau_{\mathcal{T}}(\alpha_i)$ grows beyond any bound as j increases, and where c_{0_i} is 0 if transition a_i is enabled on the valuation $\{x_1 \mapsto t_{0_1}, \ldots, x_n \mapsto t_{0_n}\}$ and *nil* otherwise. The set of TTS-computations in $\Psi_{TTS}(\mathcal{T})$ is denoted by $\mathit{TTS}\text{-}\mathcal{C}(\Psi_{TTS}(\mathcal{T}))$.

Theorem 15 *Let $\mathcal{T} = (\{x_1, \ldots, x_n\}, \{a_1, \ldots, a_m\}, l, u)$ be a timed transition system satisfying the additional requirements in this section. Then, there is a bijective function $\tilde{(\_)} : \mathit{Comps}(\mathcal{T}) \to \mathit{TTS}\text{-}\mathcal{C}(\Psi_{TTS}(\mathcal{T}))$ taking a computation of $\mathcal{T}$ of the form σ defined above into a TTS-computation of the form $\tilde{\sigma}$ with $\tau_{\mathcal{T}}(\alpha_i) = r_i - r_{i-1}$ for each proof term α_i, and $t_{i_j} = v_i(x_j)$ for each $i \geq 0$ and $j \in \{1, \ldots, n\}$.*

Proof sketch: Clearly, $\tilde{(\_)}$ induces a bijection between the initial states of $\mathit{Comps}(\mathcal{T})$ and the initial states of $\mathit{TTS}\text{-}\mathcal{C}(\Psi_{TTS}(\mathcal{T}))$. To prove that $\tilde{(\_)}$ gives a one-to-one correspondence between prefixes of computations and TTS-computations having $k+1$ steps, given such a correspondence for the first k steps, we observe that, if $c_{i_j} = \mathit{nil}$ in $\tilde{\sigma}$, then a_j is not enabled on v_i in the computation σ, and that if $c_{i_j} \neq \mathit{nil}$, then there is an $l \leq i$ with $r_l + c_{i_j} = r_i$ such that a_j is enabled on $v_l, v_{l+1}, \ldots, v_i$, and either $l = 0$ or a_j is not enabled on v_{l-1}.

3.4 Phase transition systems

Phase transition systems (PTSs) [25] generalize timed transition systems to a model suitable for specifying hybrid systems[9]. We give here only a brief

[9] Note that the expression *phase transition system* is sometimes used for the hy-

overview of a representation of PTSs in rewriting logic. The reference [37] gives more details about the translation. Intuitively, the PTS model extends the TTS model by letting time act on each valuation according to a function

$$\delta : \Sigma_D \times \mathbb{R}_+ \to \Sigma_D$$

where $\delta(v, r)$ denotes the state of the PTS after time has acted on a system in state v for time r. Furthermore, time cannot elapse past a moment when the enabling condition of a transition changes. The special case of TTS can then be characterized as those PTSs such that δ is the projection function π_{Σ_D}, that is, when time acts trivially on the states. Since the action of time can change the enabling of transitions, we assume that there is a computable function

$$\mathit{enabling\_change} : \Sigma_D \to \mathbb{R}_+ \cup \{\infty\}$$

which takes a state as argument and gives the maximum time the system can proceed without changing the enabling of a transition.

The global state and the (instantaneous) transitions of a PTS are modeled in rewriting logic as for the TTS case. The functions δ and *enabling_change* are defined on terms of the sort *Valuation*. The following tick rule ensures, in addition to the TTS requirement, that time cannot elapse beyond the latest moment when a transition must be taken, that all state components are updated according to their continuous behavior, and that the corresponding clocks are updated when an enabling condition changes:

$$\begin{aligned} [\mathit{tick}] : \ & \{V, c_1, \ldots, c_m\} \xrightarrow{x_r} \{\delta(V, x_r), c'_1, \ldots, c'_m\} \\ & \quad \mathbf{if} \ \textstyle\bigwedge_i (c_i + x_r \leq u_{a_i}) \wedge (0 < x_r \leq \mathit{enabling\_change}(V)) \end{aligned}$$

for V, $c_1, \ldots, c_n$, and x_r distinct variables of appropriate sorts, and where each c'_j abbreviates

$$\begin{aligned} & \mathbf{if} \ \mathit{not}(\mathit{enabled}_{a_j}(\delta(V, x_r)) \ \mathbf{then} \ \mathit{nil} \\ & \mathbf{else \ if} \ c_j == \mathit{nil} \ \mathbf{then} \ 0 \ \mathbf{else} \ c_j + x_r \ \mathbf{fi} \ \mathbf{fi}. \end{aligned}$$

brid systems extension of the *clocked transition system* [26] model. Due to space limitations, we do not discuss clocked transition systems in this paper.

In a concurrent object-oriented system, the concurrent state, which is usually called a *configuration*, has typically the structure of a *multiset* made up of objects and messages. Therefore, we can view configurations as built up by a binary multiset union operator, which we can represent with empty syntax as

$$\_\_ : \mathit{Configuration}\ \mathit{Configuration} \to \mathit{Configuration}\ \ [\mathit{assoc}\ \ \mathit{comm}\ \ \mathit{id}\colon\ \mathit{none}]$$

where the multiset union operator $\_\_$ is declared to satisfy the structural laws of associativity and commutativity, and to have identity *none*. The subsort declaration

$$\mathit{Object}\ \ \mathit{Msg} \le \mathit{Configuration}$$

states that objects and messages are singleton multiset configurations, so that more complex configurations are generated from them by multiset union. A sort *ObjConfiguration* denoting configurations without messages can be obtained by adding the subsort declaration

$$\mathit{Object} \le \mathit{ObjConfiguration} \le \mathit{Configuration}$$

and the operator declaration

$$\begin{aligned} \_\_ : \mathit{ObjConfiguration}\ \mathit{ObjConfiguration}\ &\to\ \mathit{ObjConfiguration} \\ &\qquad [\mathit{assoc}\ \ \mathit{comm}\ \ \mathit{id}\colon\ \mathit{none}]. \end{aligned}$$

Objects are terms (of sort *Object*) of the form

$$\langle O : C \mid \mathit{att}_1 : \mathit{val}_1, \ldots, \mathit{att}_n : \mathit{val}_n \rangle$$

denoting an object named O, where O belongs to a sort *Oid* of *object identifiers*, of class C in a state having values $\mathit{val}_1, \ldots, \mathit{val}_n$ for the attributes $\mathit{att}_1, \ldots, \mathit{att}_n$, respectively.

The dynamic behavior of concurrent object systems is axiomatized by specifying each of its concurrent transition patterns by a rewrite rule. For example, the rule

$$\begin{aligned} [l] :\ m(O, w)\,\langle O : C \mid \mathit{att}_1 : x,\ \mathit{att}_2 : y,\ \mathit{att}_3 : z \rangle &\longrightarrow \\ \langle O : C \mid \mathit{att}_1 : x + w,\ \mathit{att}_2 : y,\ \mathit{att}_3 : z \rangle\, m'(y, x + w)& \end{aligned}$$

defines a (family of) transition(s) in which a message m having arguments O and w is consumed by an object O of class C, with the effect of altering the attribute att_1 of the object and of generating a new message $m'(y, x+w)$. By convention, attributes, such as att_3 in our example, whose values do not change and do not affect the next state of other attributes need not be mentioned in a rule. Attributes like att_2 whose values influence the next state of other attributes or the values in messages, but are themselves unchanged, may be omitted from righthand sides of the rules. Thus the above rule could also be written

$$[l] : \ m(O, w)\, \langle O : C \mid att_1 : x,\ att_2 : y \rangle \longrightarrow \langle O : C \mid att_1 : x + w \rangle\ m'(y, x + w).$$

Real-time object-oriented systems can be specified by means of real-time rewrite theories by extending this setting with a sort *System* and an operator

$$\{\_\} : \mathit{Configuration} \to \mathit{System}.$$

Therefore, tick rules are *global* transitions that simultaneously advance the time for all objects in the configuration. However, the system may still exhibit concurrency in its *local* transitions, which may occur between tick applications. We illustrate this style of specification of real-time object-oriented systems with a simple example.

3.5.1 Example: A single-thermostat system

A *single-thermostat system* consists of a thermostat object and zero or more "user" objects. A user may request a new desired temperature at any time by sending a message to the thermostat, which has to provide a temperature which is within 5 degrees of the user's desire, whenever this is possible. The thermostat otherwise works as the one modeled in Fig. 2, in that the temperature increases by 2 degrees per time unit when the heater is turned on, and decreases by 1 degree per time unit when the heater is turned off.

We assume that the specification includes a specification of *Time*, which satisfies the theory *TIME*, and a sort *Temp* denoting possible temperature values together with all the necessary operations. A sort *OnOff* contains the constants *on* and *off*, describing the state of the heater associated with the thermostat. A thermostat object has attributes *curr_temp* and *desired_temp* of sort *Temp*, denoting the current and desired temperatures, as well as an attribute *heater*, denoting the state of its heater. A user object is an object with an empty set of attributes.

In the following, let U and TS be variables of the sort *Oid* of object names, let x_r be a variable of sort *Time*, let y and z be variables of sort *Temp*, and let OC be a variable of the sort *ObjConfiguration* of message-less configurations.

At any time, a user may request a new desired temperature:

$$[\mathit{new\_temp}]: \ \langle U : \mathit{User}\rangle \longrightarrow \langle U : \mathit{User}\rangle(\mathit{set\_temp}(y)).$$

The thermostat object should treat such a message by recording the new desired temperature (followed by the changing of the heater state if necessary):

$$\begin{aligned}[\mathit{treat\_request}]: \ & (\mathit{set\_temp}(y))\langle TS : \mathit{Thermostat} \mid \mathit{desired\_temp}\colon z\rangle \longrightarrow \\ & \langle TS : \mathit{Thermostat} \mid \mathit{desired\_temp}\colon y\rangle.\end{aligned}$$

The thermostat must turn off/on the heater, either when time has acted on a system such that the current temperature is exactly the desired temperature plus/minus 5 degrees, or when the system must change due to the adjustment of the desired temperature, in which case the current temperature may be more than 5 degrees off the desired temperature:

$$\begin{aligned}[\mathit{on}]: \ & \langle TS : \mathit{Thermostat} \mid \mathit{curr\_temp}\colon y, \mathit{desired\_temp}\colon z, \mathit{heater}\colon \mathit{off}\rangle \longrightarrow \\ & \langle TS : \mathit{Thermostat} \mid \mathit{heater}\colon \mathit{on}\rangle \ \textbf{if} \ y \leq z - 5 \\ [\mathit{off}]: \ & \langle TS : \mathit{Thermostat} \mid \mathit{curr\_temp}\colon y, \mathit{desired\_temp}\colon z, \mathit{heater}\colon \mathit{on}\rangle \longrightarrow \\ & \langle TS : \mathit{Thermostat} \mid \mathit{heater}\colon \mathit{off}\rangle \ \textbf{if} \ y \geq z + 5.\end{aligned}$$

The following tick rules model the *effect* of time elapse on a system and ensure that:

(1) time elapse can "stop" at any moment, reflecting the fact that the rule *new_temp* could be applied at any time,
(2) time does not elapse past the moments when the heater state should be changed, and
(3) time does not elapse while there are any messages in the system (i.e., the requested temperature should be recorded at the time when it is requested by a user); this is accomplished by the use of the variable OC of sort *ObjConfiguration*.

$[tick_{on}]:$

$$\{\langle TS : Thermostat \mid curr\_temp\colon y,\ desired\_temp\colon z, heater\colon on\rangle\ OC\} \xrightarrow{x_r}$$
$$\{\langle TS : Thermostat \mid curr\_temp\colon y + x_r + x_r\rangle\ OC\}\ \textbf{if}\ y + x_r + x_r \leq z + 5$$

$[tick_{off}]:$

$$\{\langle TS : Thermostat \mid curr\_temp\colon y,\ desired\_temp\colon z, heater\colon off\rangle\ OC\} \xrightarrow{x_r}$$
$$\{\langle TS : Thermostat \mid curr\_temp\colon y - x_r\rangle\ OC\}\ \textbf{if}\ y - x_r \geq z - 5.$$

The specification will work as expected, provided that the initial state contains exactly one thermostat object. A specification of a many-thermostat system is given in Section 3.5.3.

3.5.2 Object-oriented δ-systems

The elapse of time affects one (functional) attribute in the single-thermostat system described above. The tick rules could therefore be given by specifying directly the effect of time on that attribute. However, in more general object-oriented systems there can be an unbounded number of objects in a configuration which are affected by the elapse of time, in which case a finite number of tick rules cannot specify the elapse of time directly on the functional attributes. A simple solution is to use a function δ denoting the action of time on a configuration. The important class of real-time object-oriented systems where the objects have only functional attributes provides an instance of multiset distributed systems. Therefore, we can use the techniques described in Section 2.5.3 to circumvent lack of coherence problems w.r.t. δ without sacrificing concurrency. The following declarations should be added to the general framework for specifying object-oriented real-time systems given above, with $\{\_\}$ redefined as stated below.

sorts $\quad Tokens\ \ *Configuration$
subsorts $Tokens\ Configuration \leq *Configuration$
op $\quad * : \ \to Tokens$
op $\quad \_\_ : Tokens\ Tokens \to Tokens\ \ [assoc\ comm\ id\colon none]$
op $\quad \_\_ : *Configuration\ \ *Configuration \to\ *Configuration$
$\qquad [assoc\ comm\ id\colon none]$
op $\quad \{\_\} : *Configuration \to System$
op $\quad \delta : Configuration\ Time \to Configuration.$

The tick rules are of the form (‡) in Section 2.5.3 with the sort *State* specialized to *Configuration*. Each instantaneous rule should have the form [10]

$$[l] : * \, t \longrightarrow * \cdots * \; t',$$

for t, t' terms of sort *Configuration*, and where the number of tokens $*$ in the righthand side should equal one plus the number of objects created by the rule, minus the number of objects deleted by the rule. The initial state of a system should be of the form $\{* \cdots * \; u\}$, where the number of tokens equals the number of objects in the term u of sort *Configuration*.

Distribution over configurations. An operator δ acting on configurations provides, as we have just seen, a natural way of expressing the action of time on object systems where the number of objects in a configuration upon which time has an effect is unbounded. In these cases, δ should typically distribute over the elements in the configuration as modeled by the following axioms:

$$\begin{aligned} &\delta(none, x_r) = none \\ &\delta(C\ C', x_r) = \delta(C, x_r)\ \delta(C', x_r) \ \textbf{if}\ C \neq none \text{ and } C' \neq none \end{aligned}$$

(for C, C' variables of sort *Configuration*), to which the definition of the specific effect of time on single objects and on messages must be added to completely specify δ. The condition that C and C' be different from `none` ensures that the two equations above define a terminating rewrite system modulo associativity, commutativity, and identity ($none$) of the configuration union operator, when oriented from left to right.

In systems parameterized by $LTIME_\infty$ theories, a function mte giving the maximum time elapse for objects and messages can be extended to configurations by the axioms

$$\begin{aligned} &mte(none) = \infty \\ &mte(C\ C') = \min(mte(C), mte(C')) \ \textbf{if}\ C \neq none \text{ and } andC' \neq none. \end{aligned}$$

[10] In systems where the number of objects created by a rule application depends on the state, the condition on the form of the rules could be relaxed so that $* \cdots *$ can be given by a term of sort *Tokens*, computing the number of tokens as a function of the state.

3.5.3 Example: A multi-thermostat system

A multi-thermostat system can have an arbitrary number of rooms, each equipped with a thermostat that works as in the single-thermostat system. Each user object is extended to contain a list of the thermostats to which it has access.

Let the specification be parameterized by the theory LTIME_∞. Furthermore, let TS and U be variables of sort Oid, let S be a variable of a sort of sets of Oids (where set union is denoted by juxtaposition), let C be a variable of sort $\mathit{Configuration}$, let T be a variable of sort Tokens, let x_r be a variable of sort Time, and let y and z be variables of sort Temp. Then, the function $\delta : \mathit{Configuration}\ \mathit{Time} \to \mathit{Configuration}$ denoting the action of time, and the function $\mathit{mte} : \mathit{Configuration} \to \mathit{Time}_\infty$ computing the maximum time elapse in a tick both distribute over configurations according to the equations described above, and are defined for singleton configurations as follows:

The rules of the system can then be given as follows:

$$
\begin{array}{ll}
[\mathit{new\_temp}] : & * \langle U : \mathit{User} \mid \mathit{thermostats} \colon \mathit{TS}\ S \rangle \longrightarrow \\
 & * \langle U : \mathit{User} \rangle\ (\mathit{set\_temp}(\mathit{TS}, y)) \\
[\mathit{set\_temp}] : & * (\mathit{set\_temp}(\mathit{TS}, y))\ \langle \mathit{TS} : \mathit{Thermostat} \mid \mathit{desired\_temp} \colon z \rangle \longrightarrow \\
 & * \langle \mathit{TS} : \mathit{Thermostat} \mid \mathit{desired\_temp} \colon y \rangle \\
[\mathit{turn\_on}] : & * \langle \mathit{TS} : \mathit{Thermostat} \mid \mathit{curr\_temp} \colon y,\ \mathit{desired\_temp} \colon z,\ \mathit{heater} \colon \mathit{off} \rangle \\
 & \longrightarrow\ * \ \langle \mathit{TS} : \mathit{Thermostat} \mid \mathit{heater} \colon \mathit{on} \rangle \ \mathbf{if}\ y \leq z - 5 \\
[\mathit{turn\_off}] : & * \langle \mathit{TS} : \mathit{Thermostat} \mid \mathit{curr\_temp} \colon y,\ \mathit{desired\_temp} \colon z,\ \mathit{heater} \colon \mathit{on} \rangle \\
 & \longrightarrow\ * \ \langle \mathit{TS} : \mathit{Thermostat} \mid \mathit{heater} \colon \mathit{off} \rangle \ \mathbf{if}\ y \geq z + 5 \\
[\mathit{tick}] : & \{T\ C\} \xrightarrow{x_r} \{T\ \delta(C, x_r)\}\ \mathbf{if}\ x_r \leq \mathit{mte}(C).
\end{array}
$$

3.6 Timed Petri Nets

A Petri net [40] is usually presented as a set of *places* (each place representing a certain kind of *resource*), a disjoint set of *transitions*, and a relation of causality between them that associates to each transition the set of resources consumed and produced by its firing. Meseguer and Montanari recast this idea in an algebraic framework in [33], viewing the distributed states of the net, called *markings*, as multisets of places, and viewing the transitions as the arrows of an ordinary graph whose nodes are markings. In [29,34] it has been shown how Petri net computations can be expressed by rewriting of markings.

Petri nets have been extended to model real-time systems in different ways (see e.g. [1,35,19]). Three of the most frequently used ways of adding time to Petri nets are the following [35], from which other timed versions of Petri nets can be obtained either as special cases or by combining the extensions:

(1) Each *transition* t has an associated time interval $[l_t, u_t]$. A transition fires as soon as it can, but the resulting tokens are delayed, that is, when a transition t fires, the resulting tokens are not visible in the system until after some time $r \in [l_t, u_t]$.
(2) Each *place* p has a duration d_p. A token at place p cannot participate in a transition until it has been at p for at least time d_p.
(3) Each transition t is associated with a time interval $[l_t, u_t]$, and the transition t cannot fire before it has been continuously enabled for at least time l_t. Also, the transition t cannot have been enabled continuously for more than time u_t without being taken.

We will cover the first two cases as special cases of the *interval timed colored Petri net* (ITCPN) model proposed by van der Aalst [1]. The third case can be given a treatment similar to that of timed and phase transition systems in Sections 3.3 and 3.4.

ITCPNs appear in the context of *colored* Petri nets, where instead of having *atomic* places one has structured data. In this exposition, we abstract away from the colors of the tokens to concentrate on real-time features (see e.g. [34] for a treatment of colored nets in rewriting logic).

3.6.1 Interval Timed Petri Nets

We give a definition of *interval timed Petri nets* (ITPNs) similar to that of interval timed colored Petri nets given in [1]. The only differences between these two models are that ITPNs ignore the coloring of the tokens, and have a notion of concurrent firing of multisets of transitions.

In the ITPN model, as in the ITCPN model, we attach a *timestamp* to each token. This timestamp indicates the time when a token becomes available. The *enabling time* of a transition is the maximum timestamp of the tokens to be consumed. Transitions are *eager* to fire (i.e., they fire as soon as possible). Therefore the transition with the smallest enabling time will fire first. Firing is an atomic action, producing tokens with a timestamp equal to the firing time plus some *firing delay*, which is specified by a lower and upper bound of its possible values for each token produced by the transition.

The ITPN in Fig. 3 models a setting where each process performs transition `a`, followed by performing transition `b` within time 5 to 10, again followed by performing transition `a` time 4 to 8 thereafter, and so on. Furthermore, each

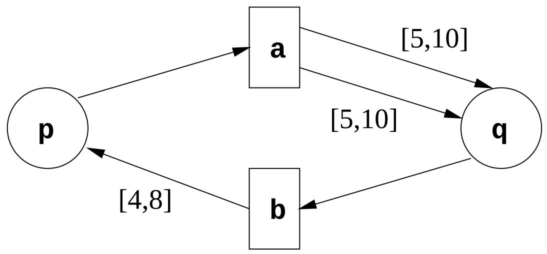

Fig. 3. An interval timed Petri net.

process *forks* when performing transition a (this is modeled by having two arcs from a to q), and the created process has the same behavior as its parent process.

A *finite multiset* over a set S is a function m from S to the set $\mathbb{N}$ of natural numbers such that its support $\mathcal{S}(m) = \{s \in S \mid m(s) > 0\}$ is finite. We denote by $S^{\oplus}$ the set of finite multisets over S, and by $\emptyset_S$ the empty multiset over S, and define membership by $s \in m \Leftrightarrow m(s) > 0$, inclusion by $m \sqsubseteq m' \Leftrightarrow \forall s \in S\,(m(s) \leq m'(s))$, multiset union by $m \oplus m' = \lambda s \in S.m(s) + m'(s)$, and subtraction by $(m - s)(s) = \max(0, m(s) - 1)$ and $(m - s)(s') = m(s')$ for $s' \neq s$.

The time domain of ITCPNs is the set of nonnegative real numbers, but for executability purposes, we assume that the time domain of an ITPN is the nonnegative part $\mathbb{R}_+$ of a computable subfield of the reals. The set TI of all *time intervals* is the set $TI = \{\, [r_1, r_2] \mid r_1, r_2 \in \mathbb{R}_+ \wedge r_1 \leq r_2\}$.

Definition 16 *An* interval timed Petri net *(ITPN) is a tuple* $N = (\mathsf{P}, \mathsf{T}, \partial_0, \partial_1)$, *where* P *and* T *are finite sets of* places *and* transitions, *and* $\partial_0 : \mathsf{T} \to \mathsf{P}^{\oplus}$ *and* $\partial_1 : \mathsf{T} \to (\mathsf{P} \times TI)^{\oplus}$ *are functions (denoting, respectively, the tokens consumed and produced by a transition, together with the delay intervals of the produced tokens) such that* $\partial_0(t) \neq \emptyset_{\mathsf{P}}$ *for each* $t \in \mathsf{T}$.

In the following, let $N = (\mathsf{P}, \mathsf{T}, \partial_0, \partial_1)$ be an ITPN. The set of *markings* $\mathsf{MRK} = (\mathsf{P} \times \mathbb{R}_+)^{\oplus}$ is the set of all finite multisets of pairs (p, r) representing the presence of a token at place p with timestamp r. The function *places* : $\mathsf{MRK} \to \mathsf{P}^{\oplus}$ which removes the timestamps from a marking is defined by $places(m)(p) = \Sigma_{(p,r) \in \mathcal{S}(m)}\, m(p, r)$. The function $max : \mathsf{MRK} \backslash \{\emptyset_{(\mathsf{P} \times \mathbb{R}_+)}\} \to \mathbb{R}_+$ which finds the maximal timestamp in a non-empty marking is given by $max(m) = \max\{r \in \mathbb{R}_+ \mid \exists p\,((p, r) \in m)\}$. The earliest enabling time of any transition in a marking is given by a function $EET : \mathsf{MRK} \to \mathbb{R}_+ \cup \{\infty\}$ defined by

$$EET(m) = \min\{max(m') \mid \exists t \in \mathsf{T}, m' \in \mathsf{MRK}\,(m' \sqsubseteq m \wedge places(m') = \partial_0(t))\}$$

with $\min(\emptyset) = \infty$. The function $addToInt : (\mathsf{P} \times TI)^{\oplus} \times \mathbb{R}_+ \to (\mathsf{P} \times TI)^{\oplus}$

adds the second argument to all the intervals of the tokens, and is defined by $addToInt(m,r)(p,[r_1+r,r_2+r]) = \Sigma_{(p,[r_1,r_2])\in\mathcal{S}(m)}\, m(p,[r_1,r_2])$, and by $addToInt(m,r)(p,[r',r'']) = 0$ when $r' < r$. Finally, to relate multisets of tokens with timestamps with multisets of tokens with time intervals, we define the specialization relation $\lhd \subseteq \mathsf{MRK} \times (\mathsf{P} \times \mathit{TI})^{\oplus}$, where $m \lhd m'$ holds if and only if each token in m corresponds to one token in m', such that they are in the same place and the timestamp of the token in m is in the interval of the corresponding token in m'. That is, $m \lhd m'$ if and only if either $(m = \emptyset_{\mathsf{P}\times\mathbb{R}_+} \wedge m' = \emptyset_{\mathsf{P}\times \mathit{TI}})$ or $\exists (p,r) \in m\ \ (\exists (p,[r_1,r_2]) \in m'\ \ (r_1 \leq r \leq r_2 \wedge (m - (p,r)) \lhd (m' - (p,[r_1,r_2]))))$.

An ITPN makes computational progress by applying transitions, thereby consuming and producing multisets of timestamped tokens. A finite number of transition firings occurring at the same time can be composed in parallel to form a *step*, which, therefore, is a nonempty finite multiset of transitions. Given an ITPN N, we denote by $m \stackrel{e}{\to} m'$ a step from marking m to marking m' by the concurrent firing of the transitions e, and define the step relation $\to\ \subseteq \mathsf{MRK} \times (\mathsf{T}^{\oplus} \setminus \{\emptyset_{\mathsf{T}}\}) \times \mathsf{MRK}$ to be all triples $m \stackrel{e}{\to} m'$ which can be generated by the following rules:

$$\frac{places(m) = \partial_0(t) \quad max(m) = EET(m) \quad m' \lhd addToInt(\partial_1(t), EET(m))}{m \stackrel{t}{\to} m'}$$

$$\frac{m \stackrel{e}{\to} m' \qquad m'' \in \mathsf{MRK} \qquad EET(m \oplus m'') = EET(m)}{(m \oplus m'') \stackrel{e}{\to} (m' \oplus m'')}$$

$$\frac{m_1 \stackrel{e_1}{\to} m_1' \qquad m_2 \stackrel{e_2}{\to} m_2' \qquad EET(m_1 \oplus m_2) = EET(m_1) = EET(m_2)}{(m_1 \oplus m_2) \stackrel{e_1 \oplus e_2}{\to} (m_1' \oplus m_2')}.$$

A *step sequence* in N is a (finite or infinite) sequence

$$\varsigma :\quad m_0 \stackrel{e_1}{\to} m_1 \stackrel{e_2}{\to} m_2 \stackrel{e_3}{\to} \cdots$$

of steps $m_{i-1} \stackrel{e_i}{\to} m_i$, where each such step represents the simultaneous firing of the transitions e_i at time $EET(m_{i-1})$. The set of all step sequences of an ITPN N is denoted $\mathcal{C}_N^{\infty}$.

Timed Petri nets of type (1), where transitions have durations, that is, where all tokens produced by a transition have the same firing delay, can be seen as a special case of ITPNs as follows. A transition t with time interval $[l_t, u_t]$ which consumes the tokens m and produces the tokens m' can be simulated in an ITPN by adding a new place p_t, and having a transition t_1 which consumes the tokens m and produces one token at place p_t in some time in the interval $[l_t, u_t]$, and another transition t_2 which consumes one token from p_t and produces the tokens m' in zero time. Timed Petri nets of type (2), where each place has a duration, can be seen as a special case of ITPNs where each token produced at place p has firing delay d_p.

3.6.2 Specifying ITPNs as real-time rewrite theories.

The translation into rewriting logic of timed Petri nets is based on the rewriting logic representation of untimed Petri nets given in [29], where the state of a Petri net is represented by a multiset of places called a *marking*—where if place p has multiplicity n we interpret this as the presence of n tokens in that place—and where the transitions correspond to rewrite rules on the corresponding multisets of pre- and post-places.

For the sake of simplicity of the rewriting logic representation of ITPNs, we choose not to carry the timestamps in the tokens at all times. Instead, a term $[p]$ of sort *VisibleMarking* represents a "visible" occurrence of a token at place p. A token that will become visible at place p in time r is represented by the term $dly(p, r)$ which has the sort *DelayedMarking* whenever $r \neq 0$.[11] A token with delay 0 is visible, i.e., $dly(p, 0) = [p]$. A *marking* is a multiset of these two forms of tokens, where multiset union is represented by juxtaposition. The sort *Marking* denotes the set of all markings. The function *mte* takes a term of sort *DelayedMarking* and returns the time that can pass until the next delayed token becomes available. The function δ models the effect of the passage of time on delayed tokens by decreasing their delays according to the time elapsed.

The rewriting logic translation of an ITPN $N = (\mathsf{P}, \mathsf{T}, \partial_0, \partial_1)$, with $\mathsf{P} = \{p_1, \ldots, p_n\}$, is a real-time rewrite theory $\Psi_{ITPN}(N) = ((\Sigma_N, E_N, L_N, R_N), \phi_N, \tau_N)$ where $\phi_N(TIME)$ is an axiomatization of the time domain $\mathbb{R}_+$ which can be extended to $\phi'_N : LTIME_\infty \to (\Sigma_N, E_N)$, together with the following declarations and axioms:

sorts	*Place EmptyMarking VisibleMarking DelayedMarking Marking System*
subsorts	*EmptyMarking* $\leq$ *VisibleMarking DelayedMarking* $\leq$ *Marking*
ops	$p_1 \ \ldots p_n \to Place$
op	$dly : Place\ Time \to Marking$
op	$empty : \ \to EmptyMarking$
op	$\_\_ : Marking\ Marking \to Marking\ \ [assoc\ comm\ id : empty]$
op	$\_\_ : DelayedMarking\ DelayedMarking \to DelayedMarking$ $[assoc\ comm\ id : empty]$

[11] We will see later that no interesting information about time is lost by this simplification, since the time when a firing of a transition occurs can always be extracted from the proof term.

op $\_\_ : \mathit{VisibleMarking}\ \mathit{VisibleMarking} \to \mathit{VisibleMarking}$ $[\mathit{assoc}\ \mathit{comm}\ \mathit{id} : \mathit{empty}]$
op $\_\_ : \mathit{EmptyMarking}\ \mathit{EmptyMarking} \to \mathit{EmptyMarking}$ $[\mathit{assoc}\ \mathit{comm}\ \mathit{id} : \mathit{empty}]$
op $\{\_\} : \mathit{Marking} \to \mathit{System}$
op $\delta : \mathit{DelayedMarking}\ \mathit{Time} \to \mathit{Marking}$
op $\mathit{mte} : \mathit{DelayedMarking} \to \mathit{Time}_\infty$
vars $\mathit{DM}\ \mathit{DM}' : \mathit{DelayedMarking}$ **var** $\mathit{VM} : \mathit{VisibleMarking}$
var $P : \mathit{Place}$ **vars** $x_r\ y_r : \mathit{Time}$
cmb $\mathit{dly}(P, x_r) : \mathit{DelayedMarking}$ **if** $x_r \neq 0$
eq $\mathit{dly}(P, 0) = [P]$
eq $\delta(\mathit{empty}, x_r) = \mathit{empty}$
eq $\delta(\mathit{dly}(P, x_r), y_r) = \mathit{dly}(P, x_r \dot{-} y_r)$
ceq $\delta(\mathit{DM}\ \mathit{DM}', x_r) = \delta(\mathit{DM}, x_r)\ \ \delta(\mathit{DM}', x_r)$
  if $\mathit{DM} \neq \mathit{empty}$ and $\mathit{DM}' \neq \mathit{empty}$
eq $\mathit{mte}(\mathit{empty}) = \infty$
ceq $\mathit{mte}(\mathit{dly}(P, x_r)) = x_r$ **if** $x_r \neq 0$
ceq $\mathit{mte}(\mathit{DM}\ \mathit{DM}') = \min(\mathit{mte}(\mathit{DM}), \mathit{mte}(\mathit{DM}'))$
  if $\mathit{DM} \neq \mathit{empty}$ and $\mathit{DM}' \neq \mathit{empty}$

The set R_N of rules in $\Psi_{ITPN}(N)$ consists of a *lazy* tick rule modeling time elapse and, for each transition in N, a corresponding *eager* rule. A transition t is modeled by an instantaneous rule

eager $[t] :\ m \longrightarrow \mathit{dly}(p_1, x_1) \ldots \mathit{dly}(p_n, x_n)$ **if** $l_1 \leq x_1 \leq u_1 \wedge \ldots \wedge l_n \leq x_n \leq u_n$

where m is a marking consisting of, for each p in P, $\partial_0(t)(p)$ occurrences of the term $[p]$, and where $\mathit{dly}(p_1, x_1) \ldots \mathit{dly}(p_n, x_n)$, with $\{x_1, \ldots, x_n\}$ a set of n distinct variables of the sort of the time domain, is such that for each occurrence of $(p, [l, u])$ in $\partial_1(t)$, there is a distinct $i \in \{1, \ldots, n\}$ with $p_i = p$, $l_i = l$, and $u_i = u$. The tick rule advances time until the first delayed token becomes visible [12] :

lazy $[\mathit{tick}] :\ \{\mathit{VM}\ \ \mathit{DM}\} \stackrel{\mathit{mte}(\mathit{DM})}{\longrightarrow} \{\mathit{VM}\ \ \delta(\mathit{DM}, \mathit{mte}(\mathit{DM}))\}$
  if $\mathit{mte}(\mathit{DM}) \neq \infty$.

For example, the translation of the ITPN in Fig. 3 contains the above tick

[12] To have the possibility of stopping at any moment in the time domain, a nondeterministic time advance $x_r \leq \mathit{mte}(\mathit{DM})$ could be used instead if desired.

rule and the following two instantaneous rules:

$$\textbf{eager}\ [a]:\ [p] \longrightarrow dly(q, x_r)\ dly(q, y_r)\ \textbf{if}\ 5 \leq x_r \leq 10 \wedge 5 \leq y_r \leq 10$$
$$\textbf{eager}\ [b]:\ [q] \longrightarrow dly(p, x_r)\ \textbf{if}\ 4 \leq x_r \leq 8$$

Transitions are supposed to fire as soon as possible in timed Petri nets. This is accomplished by the strategy described in Section 2.5.2 that triggers all eager transition rules until none of these can be applied, followed by one application of the tick rule. The use of eager and lazy rules allows a simple condition in the tick rule, which would otherwise have to take into account the enabledness of every transition in the system, together with the *mte* part of the tick rule. Here, the tick rule only needs to compute the time until the next delayed token(s) become visible and advance time by that amount. After such a tick, the tick rule is again enabled but, due to its being lazy, it will not be applied if the introduction of the new token(s) enables an (eager) transition (which in turn could trigger more transitions in zero-time).

Although ITPN markings are multisets, it is not necessary to apply the techniques described in Section 2.5.3 to specify ITPNs in rewriting logic. This is because terms of sort *DelayedMarking* cannot be rewritten, which implies that there are no "aged" rewrites, and that the introduction of the function $mte : \mathit{DelayedMarking} \to \mathit{Time}_\infty$ does not lead to undesired nontrivial rewrites in the time domain.

Since a step $m \overset{e}{\to} m'$ of an ITPN does not depend on the firing delays of the individual transitions taken in the step, two *one-step* rewrites $\{\alpha\} : u \longrightarrow v$ and $\{\beta\} : u \longrightarrow v$ should be considered equal—in the sense that we add this equivalence as a further equality identifying rewrite proofs—if the multisets of rule labels in α and β are the same. That is, α and β are equivalent iff $\alpha = t(r_1, \ldots, r_n)\ \alpha'$ and $\beta = t(r'_1, \ldots, r'_n)\ \beta'$ for $t \in \mathsf{T}$ and α' and β' are (recursively) equivalent. An *ITPN-computation* in $\Psi_{ITPN}(N)$ is a finite or infinite sequence

$$\tilde{\varsigma}:\quad u_0 \overset{\alpha_1;\beta_1}{\longrightarrow} u_1 \overset{\alpha_2;\beta_2}{\longrightarrow} u_2 \overset{\alpha_3;\beta_3}{\longrightarrow} \cdots$$

of admissible rewrite proofs $\alpha_i; \beta_i : u_{i-1} \longrightarrow u_i$ in $\Psi_{ITPN}(N)$ where $u_{i-1}, u_i \in \mathcal{T}_{\Psi_{ITPN}(N)}$, modulo the equivalence on proof terms given above, such that each α_i corresponds either to the identity proof or to a sequence of tick applications, each β_i corresponds to a one-step concurrent rewrite using instantaneous rules, and u_0 is a term $\{m_0\}$ with m_0 a term of sort *Marking*. The set of ITPN-computations in $\Psi_{ITPN}(N)$ is denoted $\mathit{ITPN}\text{-}\mathcal{C}(\Psi_{ITPN}(N))$. The following proposition follows from the factorization property of proofs in rewriting logic [29].

Proposition 17

- *Each non-identity admissible ground rewrite* $\epsilon : \{m\} \longrightarrow \{m'\}$ *in* $\Psi_{ITPN}(N)$ *is equivalent to a rewrite* $\alpha; \gamma$*, such that* α *can be rearranged as a finite ITPN computation, and* γ *corresponds to the identity proof or to a sequence of tick applications.*
- *Each (infinite) computation of* $\Psi_{ITPN}(N)$*, consisting of admissible rewrites involving ground terms of sort System, which contains an infinite number of applications of instantaneous rules, can be rearranged as an ITPN-computation by an appropriate composition of the arrows.*

The fact that ITPNs are faithfully represented in their rewriting logic translation is made precise in the following theorem.

Theorem 18 *Let* N *be an ITPN. Then, there is a bijective function* $\tilde{(\_)} : \mathcal{C}_N^{\infty} \to$ *ITPN-*$\mathcal{C}(\Psi_{ITPN}(N))$ *taking a step sequence of the form* ς *to an ITPN-computation of the form* $\tilde{\varsigma}$ *such that:*

- *Each* u_i *is a term equivalent to a term of the form* $\{w_i\}$*, which consists of* $m_i(p, r)$ *occurrences of the term* $dly(p, r \dot{-} \tau(\alpha_1; \beta_1; \ldots; \alpha_i; \beta_i))$*, for all* p *and* r *(recall that* $dly(p, 0)$ *is equivalent to* $[p]$*).*
- *The transitions fire at the same time in* ς *and* $\tilde{\varsigma}$*, that is,* $\tau(\alpha_1; \beta_1; \ldots; \alpha_{i+1}) = EET(m_i)$*.*
- *The transitions taken (concurrently) in each step are the same. That is, each* β_i *is equivalent to a proof term of the form* $\{w_i\, \varepsilon_i\}$*, where* w_i *is a term, and where* ε_i *is a multiset consisting of, for each* $t \in \mathsf{T}$*, exactly* $e_i(t)$ *occurrences of proof terms of the form* $t(r_1, \ldots, r_n)$*.*

Proof sketch Given a step sequence ς, we can inductively define a unique ITPN-computation $\tilde{\varsigma}$, having the desired properties, as follows. The term $u_0 = \{w_0\} = \tilde{m}_0$ in $\tilde{\varsigma}$ is obtained from m_0 by letting w_0 consist of $m_0(p, r)$ terms of the form $\mathtt{dly}(p, r)$ for each $(p, r) \in m_0$. Let $m_i \overset{e_{i+1}}{\to} m_{i+1}$ be a step in ς. Then, by induction we already have a unique prefix of $\tilde{\varsigma}$ of i steps. In particular, we can assume that $u_i = \tilde{m}_i$ is a term of the form $\{w_i\}$, where w_i consists of $m_i(p, r)$ occurrences of the term $\mathtt{dly}(p, r \dot{-} r_i)$ for each $(p, r) \in m_i$, where $r_i = \Sigma_{j=1}^{i}(\tau_N(\alpha_j)) = EET(m_{i-1})$ for $i > 0$, and $r_0 = 0$. Furthermore, for each state $m \in \mathsf{MRK}$ and $r \in \mathbb{R}_+$, let $m^{\leq r}$ denote the submarking of m containing only the pairs (p, r') where $r' \leq r$, that is, those elements of m which could be consumed by firings taking place at time r. Then, the multiset of undelayed places in w_i corresponds to $m_i^{\leq r_i}$. Since no transition is enabled on the state m_i before time $EET(m_i)$, no eager rule will be enabled on u_i until time has elapsed $EET(m_i) - r^i$. Therefore, since the tick rule exhibits deterministic behavior in time, a unique sequence α_{i+1} of tick applications with $\tau_N(\alpha_{i+1}) = EET(m_i) - r_i$ can be applied to u_i, giving a term u_i'. The term u_i' contains $m_i(p,r)$ occurrences of the term $\texttt{dly}(p, r \dot{-} EET(m_i))$ for each $(p,r) \in m_i$. That is, the visible marking in u_i' is $places(m_i^{\leq EET(m_i)})$. Therefore, we can associate to e_{i+1} a unique (modulo the equivalence on proofs described above) rewrite $\beta_{i+1} : u_i' \longrightarrow u_{i+1}$ where β_{i+1} is a proof term of the form $\{v_{i+1}\, \gamma_{i+1}\}$ for some term v_{i+1}, or of the form $\{\gamma_{i+1}\}$, with γ_{i+1} a nonempty multiset of proofs consisting of an occurrence of t for each transition t in e_{i+1}. The term u_{i+1} is a term $\{w_{i+1}\}$ where w_{i+1} contains $m_{i+1}(p,r)$ occurrences of the term $dly(p, r \dot{-} EET(m_i))$ for each $(p,r) \in m_{i+1}$.

Conversely, given an ITPN-computation of the form $\tilde{\varsigma}$, we can inductively define its uniquely associated step sequence ς, satisfying the desired properties, as follows. For each $p \in \mathsf{P}$ and $r \in \mathbb{R}_+$, $m_0(p,r)$ equals the number of occurrences of the term $dly(p,r)$ in u_0 (again, with $[p] = dly(p,0)$). Let now $u_i \stackrel{\alpha_{i+1};\beta_{i+1}}{\longrightarrow} u_{i+1}$ be the ith step in $\tilde{\varsigma}$ with $u_i \stackrel{\alpha_{i+1}}{\longrightarrow} u_i'$. By induction, the state m_i corresponding to u_i consists of an occurrence of (p, r'), for some $r' \leq r_i = \Sigma_{j=1}^{i} \tau_N(\alpha_j)$, for each occurrence of the term $[p]$ in u_i, and of an occurrence of $(p, r_i + r'')$ for each occurrence of the term $dly(p, r'')$ in u_i. It can be shown that $r_{i+1} = \Sigma_{j=1}^{i+1} \tau_N(\alpha_j) = EET(m_i)$, since no instantaneous rule is enabled on u_i before time reaches $EET(m_i)$, and that, due to eagerness of instantaneous transitions, the tick rule cannot be applied, since an instantaneous rule would be enabled on u_i' when $r_{i+1} = EET(m_i)$. The multiset of "visible" tokens p in u_i' is then equivalent to $places(m_i^{\leq EET(m_i)})$, and, therefore, β_{i+1} maps to a unique e_{i+1} which contains an occurrence of the transition t for each occurrence of the proof term $t(\ldots)$ (for $t \in \mathsf{T}$) in β_{i+1}. A new token $dly(p,r)$ created by β_{i+1} corresponds to the presence of a new pair $(p, EET(m_i) + r)$ in m_{i+1}, ensuring that the resulting states satisfy the desired relationship between u_{i+1} and m_{i+1}.

4 Relationship to timed rewriting logic

In this section we investigate the relationship between Kosiuczenko and Wirsing's *timed rewriting logic* (TRL) [22] and the framework we have presented for specifying real-time systems directly in rewriting logic. After briefly introducing TRL in Section 4.1, we propose in Section 4.2 a translation from TRL into rewriting logic such that the translation of any TRL-sequent derivable in a TRL theory is also derivable in the corresponding rewriting logic theory. The converse is in general not true, due to some conceptual differences between TRL and our method of specifying real-time systems in rewriting logic, as explained in Section 4.3.

Timed transitivity:

$$\frac{t_1 \xrightarrow{r_1} t_2 \quad t_2 \xrightarrow{r_2} t_3}{t_1 \xrightarrow{r_1+r_2} t_3}$$

Synchronous replacement:

$$\frac{t_0 \xrightarrow{r} t_0', \quad t_{i_1} \xrightarrow{r} t_{i_1}', \ldots, t_{i_k} \xrightarrow{r} t_{i_k}'}{t_0(t_1/x_1, \ldots, t_n/x_n) \xrightarrow{r} t_0'(t_1'/x_1, \ldots, t_n'/x_n)}$$

where $\{x_{i_1}, \ldots, x_{i_k}\} = \mathcal{V}(t_0) \cap \mathcal{V}(t_0')$.

Compatibility with equality:

$$\frac{t_1 = u_1, \ r_1 = r_2, \ t_2 = u_2, \ t_1 \xrightarrow{r_1} t_2}{u_1 \xrightarrow{r_2} u_2}$$

Renaming of variables:

$$\frac{t_1 \xrightarrow{r} t_2}{\rho(t_1) \xrightarrow{r} \rho(t_2)}$$

for ρ a renaming of the variables $\mathcal{V}(t_1) \cup \mathcal{V}(t_2)$.

Fig. 4. Deduction rules in timed rewriting logic.

4.1 Timed rewriting logic

Rewriting logic has been extended by Kosiuczenko and Wirsing to handle real-time systems in their timed rewriting logic (TRL) [22]. TRL has been shown well-suited for giving object-oriented specifications of complex hybrid systems such as the steam-boiler [36] and has been illustrated by a number of specifications of simpler real-time systems. A translation into ordinary rewriting logic can illuminate the conceptual relationships between both formalisms.

A TRL theory (Σ, E, L, TR) consists of an equational specification (Σ, E) satisfying the theory *TIME* [13], a set L of labels, and a set TR of *timed* rewrite rules of the form $[l] : t \xrightarrow{r} t'$, where r is a ground term of sort *Time* denoting the duration of the rewrite. A TRL sequent has the form $t \xrightarrow{r} t'$ and its intuitive meaning is that t evolves to t' in time r. More specifically, the set of sequents derivable from a TRL theory consists of all rules in the theory, and all sequents which can be derived by equational reasoning and by using the deduction rules in Figure 4, where $\mathcal{V}(t)$ denotes the set of variables in t. This deduction system extends and modifies the rules of deduction in rewriting

[13] They impose in some cases further requirements, such as *TIME* being an Archimedean monoid. This could of course be easily accommodated.

logic with time stamps as follows:

- Reflexivity is dropped as a general axiom, since it would allow (parts of) the system to stay idle for arbitrarily long periods of time.
- Transitivity yields the addition of the time stamps. If t_1 evolves to t_2 in time r_1 and t_2 evolves to t_3 in time r_2, then t_1 evolves to t_3 in time $r_1 + r_2$.
- The synchronous replacement rule enforces uniform time elapse in all components of a system: a system rewrites in time r iff all its components do so.

4.2 Timed rewriting logic in rewriting logic

In this section we define a mapping $\mathcal{M}$ which takes any timed rewriting logic theory $\mathcal{T}$ to a real-time rewrite theory $\mathcal{M}(\mathcal{T})$ such that $\mathcal{T} \vdash t \xrightarrow{r} t'$ implies that $\mathcal{M}(\mathcal{T}) \vdash \alpha : \{t\} \longrightarrow \{t'\}$, for some α with $\tau(\alpha) = r$, for all ground ($\mathcal{T}$-) terms t, t' of the designated sort *State*.

The idea is to introduce for each sort s an operator $\delta : s\ \mathit{Time} \to s$ corresponding to the effect of time elapse. Then, a TRL sequent $t \xrightarrow{r} t'$ ("t evolves in time r to t'") can be mapped to a rewriting logic sequent $\delta(t, r) \longrightarrow t'$ ("if time has acted on t for time r, then it rewrites to t'") for ground terms t, t'. Rewrite rules must be used to define δ, since the effect of time on a TRL state is not necessarily functional.

Sort information is used to separate terms containing the symbol δ from terms of the original signature, and a tick rule is added to the rules defining δ such that for ground $\mathcal{T}$-terms t, t' of sort *State*, $\mathcal{M}(\mathcal{T}) \vdash \alpha : \{t\} \longrightarrow \{t'\}$ holds for some α with $\tau(\alpha) = r$ if and only if $\mathcal{M}(\mathcal{T}) \vdash \delta(t, r) \longrightarrow t'$, which in turn holds whenever $\mathcal{T} \vdash t \xrightarrow{r} t'$ holds. The resulting real-time rewrite theory $\mathcal{M}(\mathcal{T})$ is not easily executable, since the tick rule introduces two variables in its righthand side. This reflects the fact that in TRL it is in general undecidable whether a term rewrites in time r $(r > 0)$, and, even if it is known that t rewrites in time r, it is also in general undecidable whether t rewrites to a given term t' in time r.

We assume that the time domain is functional, that is, that no rewrites of the form $t \xrightarrow{r} t'$, with $t \neq t'$ terms of sort *Time*, can be inferred from the TRL theory $\mathcal{T}$, and restrict our treatment to TRL theories where no extra variables are introduced in the righthand side of a rule. The reason for the latter restriction is that if $[l_1] : f(x) \xrightarrow{2} g(x, y)$ and $[l_2] : g(x, y) \xrightarrow{2} h(y)$ are two rules, any system t' that appears in $h(t)$ as a result of the second rule, must have evolved for 2 time units from a system t in $g(u, t)$. However, by transitivity of the rules, the sequent $f(x) \xrightarrow{4} h(y)$ is derivable, which means that *any* system t could replace y in $h(y)$, including the systems which have not evolved for 2 time units.

4.2.1 The mapping from TRL to real-time rewrite theories

The mapping $\mathcal{M}$ sends an order-sorted TRL theory $\mathcal{T} = (\Sigma, E, L, TR)$ to a real-time rewrite theory $\mathcal{M}(\mathcal{T}) = ((\mathcal{M}(\Sigma), \mathcal{M}(E), \mathcal{M}(L), \mathcal{M}(TR)), \phi(\mathcal{T}), \tau(\mathcal{T}))$ and sends a $\mathcal{T}$-sequent $t \xrightarrow{r} t'$ to an $\mathcal{M}(\mathcal{T})$-sequent $\mathcal{M}(t \xrightarrow{r} t')$. It is defined as follows:

- The signature morphism $\phi(\mathcal{T})$ in $\mathcal{M}(\mathcal{T})$ takes *Time* to the sort *Time* in $\mathcal{T}$ denoting the time domain, and takes the functions in *TIME* to the corresponding functions in $\mathcal{T}$.
- The set of sorts in $\mathcal{M}(\Sigma)$ consists of all the sorts in Σ, plus a new sort s^δ for each sort s in Σ, as well as a new sort *System*. For each sort s in Σ, $s \leq s^\delta$, and if $s \leq s'$ in Σ, then $s^\delta \leq s'^\delta$ in $\mathcal{M}(\Sigma)$.
- $\mathcal{M}(\Sigma)$ contains function declarations $f : s_1 \ldots s_n \to s$ and $f : s_1^\delta \ldots s_n^\delta \to s^\delta$ for each function $f : s_1 \ldots s_n \to s$ in Σ, a constructor $\{\_\} : State \to System$ for the designated sort *State* in Σ, and a function $\delta : s^\delta \; Time \to s^\delta$ for each sort s in Σ.
- $\mathcal{M}(E)$ contains an axiom e^δ for each axion e in E, where e^δ is the axiom e where each variable $x : s$ is replaced by $x : s^\delta$. The set $\mathcal{M}(E)$ must also define δ to be a monoid action, that is, it contains the axioms

$$\delta(x_{s^\delta}, 0_\phi) = x_{s^\delta}$$

$$\delta(\delta(x_{s^\delta}, y_r), z_r) = \delta(x_{s^\delta}, y_r +_\phi z_r)$$

  for each sort s^δ in $\mathcal{M}(\Sigma)$, variable x_{s^δ} of sort s^δ, and variables y_r, z_r of sort *Time*.
- The mapping $\mathcal{M}$ from TRL-sequents to rewriting logic sequents is given by

$$\mathcal{M}(t(x_1 : s_1, \ldots, x_n : s_n) \xrightarrow{r} t'(x_1 : s_1, \ldots, x_n : s_n)) =$$

$$\delta(t(x_1 : s_1^\delta, \ldots, x_n : s_n^\delta), r) \longrightarrow t'(\delta(x_1 : s_1^\delta, r)/x_1, \ldots, \delta(x_n : s_n^\delta, r)/x_n)$$

  where the free variables in t are $x_1, \ldots, x_n$ and contain those of t'.

  The set of rules $\mathcal{M}(TR)$ consists of a rule $[l] : \mathcal{M}(t \xrightarrow{r} t')$ **if** C^δ for each timed rule $[l] : t \xrightarrow{r} t$ **if** C in TR, and a tick rule

$$[tick] : \;\; \{x\} \xrightarrow{y_r} \{x'\} \textbf{ if } \delta(x, y_r) \longrightarrow x'$$

  for variables x, x' of sort *State* and y_r of sort *Time*.

The theorem below shows that $\mathcal{M}$ can be naturally understood as a map of logics. Specifically, as a map $\mathcal{M} : TRL \longrightarrow RWL$ from the entailment system

[28] of TRL to that of rewriting logic.

Theorem 19 *Let $\mathcal{T}$ be a TRL specification and let $\mathcal{M}$ be defined as above. Then, for all terms $t, t', r \in \mathcal{T}_\Sigma(X)$,*

$$\mathcal{T} \vdash t \xrightarrow{r} t' \ \textit{ implies } \ \mathcal{M}(\mathcal{T}) \vdash \mathcal{M}(t \xrightarrow{r} t').$$

As a corollary to this theorem, which can be easily proved by induction on the size of the proof $t \xrightarrow{r} t'$, we obtain that $\mathcal{T} \vdash t \xrightarrow{r} t'$ implies $\mathcal{M}(\mathcal{T}) \vdash \delta(t, r) \longrightarrow t'$ for all ground terms t, t', and r, which in turn gives a rewrite $\mathcal{M}(\mathcal{T}) \vdash \alpha : \{t\} \longrightarrow \{t'\}$ with $\tau(\alpha) = r$ when t and t' are of sort *State* by applying the tick rule. It is also easy to see that $\mathcal{M}(\mathcal{T}) \vdash \alpha : \{t\} \longrightarrow \{t'\}$ implies $\mathcal{M}(\mathcal{T}) \vdash \delta(t, \tau(\alpha)) \longrightarrow t'$ for ground $\mathcal{T}$-terms t, t' of sort *State*.

4.3 Differences between TRL and its rewriting logic translation

Even though $t \xrightarrow{r} t'$ implies $\delta(t, r) \longrightarrow t'$ for ground terms, the converse is not necessarily true. In this section we discuss the differences between deduction in TRL and in its translation into rewriting logic.

4.3.1 Zero-time idling

In the rewriting logic translation, a TRL sequent $t \xrightarrow{0} t$ translates to $\delta(t, 0) \longrightarrow t(\delta(x_1, 0)/x_1, \ldots, \delta(x_n, 0)/x_n)$, which, due to the axiom $\delta(x, 0) = x$, is equal to $t \longrightarrow t$, which is always deducible in rewriting logic. However, in TRL, $t \xrightarrow{0} t$ is not necessarily valid. This obviously indicates a difference between both systems, since the notion of "zero-time idling" is always available in our approach but not in TRL.

4.3.2 Non-right-linear rules

Given the TRL theory $\{\, [l_1] : f(x) \xrightarrow{2} g(x, x),\ [l_2] : a \xrightarrow{2} b,\ [l_3] : a \xrightarrow{2} c \,\}$, the term $f(a)$ rewrites to either $g(b, b)$ or $g(c, c)$ in time two, but will *not* rewrite to $g(b, c)$. In the rewriting logic translation

$$\begin{aligned} \{\, [l_1] : \ \delta(f(x), 2) \longrightarrow g(\delta(x, 2), \delta(x, 2)),\ \ [l_2] : \ \delta(a, 2) \longrightarrow b, \\ [l_3] : \ \delta(a, 2) \longrightarrow c\,,\ \ [tick] : \ \{y\} \xrightarrow{x_r} \{y'\} \ \mathbf{if}\ \delta(y, x_r) \longrightarrow y' \,\}, \end{aligned}$$

where y and y' are variables of the designated state sort and range over δ-free terms, there is a rewrite $\delta(f(a), 2) \longrightarrow g(\delta(a, 2), \delta(a, 2)) \longrightarrow g(b, c)$, and therefore also a rewrite $\alpha : \{f(a)\} \longrightarrow \{g(b,c)\}$ with $\tau(\alpha) = 2$.

The difference depends on how the fork of a process is modeled. The rule $[l] : f(x) \xrightarrow{r} g(x,x)$ can be understood as a fork of the (sub)process t in the system $f(t)$. In the TRL setting, the actual "fork" (the point in time when the two instances of the process x can behave independently of each other) is taking place at the end of the time period of length r in the rule. In the rewriting logic setting, the "forking" took place at the beginning of the time period of duration r [14].

4.3.3 Problems related to synchrony in TRL

Another aspect in which TRL and our rewriting logic translation are different is illustrated by the following TRL specification:

$$\{\, [l_1] : f(a,y) \xrightarrow{2} g(a,y),\ [l_2] :\ g(x,y) \xrightarrow{2} h(x,y),$$
$$[l_3] :\ h(x,c) \xrightarrow{2} k(x,c),\ [l_4] :\ a \xrightarrow{4} d,\ [l_5] :\ b \xrightarrow{4} c \,\}.$$

Due to the strong synchrony requirements in TRL, $f(a,b)$ cannot be rewritten, even though the b (in the place of y), and a (for x), could be rewritten in time 4. In many cases, it would however be natural to assume that the system represented by $f(a,b)$ rewrites to $k(d,c)$ in time 6. In the rewriting logic translation, $\delta(f(a,b),6)$ rewrites indeed to $k(d,c)$.

4.4 Aging in TRL

To overcome the strong requirements of synchrony in TRL, which caused the differences in Sections 4.3.2 and 4.3.3, the special symbol *age* is introduced in [22]. It aims at making a term t, which rewrites in time r', "accessible" to synchronous rewrites in time r with $r' \geq r$, by making it visible as $age(t,r)$.

Formally, with aging the following two deduction rules are added to the TRL deduction rules given in Figure 4. In both deduction rules, $[l] :\ t \xrightarrow{r+r'} t'$ is assumed to be a timed rewrite *rule* in the specification.

$$age_1 :\ \frac{}{t \xrightarrow{r} age(t,r)} \qquad age_2 :\ \frac{}{age(t,r) \xrightarrow{r'} t'}$$

[14] Note that in the rewriting logic setting, adding a rule $[l_4] : \delta(k(x),2) \longrightarrow f(\delta(x,2))$ to the system above gives $\delta(k(x),4) \longrightarrow g(\delta(x,4),\delta(x,4))$, hence a "fork" which took place too early. Such behavior can be avoided by requiring that the variable x in the rule $[l_1] :\ \delta(f(x),2) \longrightarrow g(\delta(x,2),\delta(x,2))$ has a "non-δ-sort".

The *age* operator also satisfies the axiom $age(age(t, r), r') = age(t, r + r')$ for all terms t and time values r, r'.

With aging, the "fork" differences disappear, since (assuming $g(x, y) \xrightarrow{0} g(x, y)$) we have $f(a) \xrightarrow{2} g(age(a, 2), age(a, 2)) \xrightarrow{0} g(b, c)$ for the system in the example of Section 4.3.2, and the strong synchrony is loosened, as illustrated by the fact that in the example of Section 4.3.3, $f(a, b) \xrightarrow{6} k(d, c)$ is derivable, since $f(a, b) \xrightarrow{2} g(a, age(b, 2))$, $g(a, age(b, 2)) \xrightarrow{2} h(age(a, 2), c)$, and $h(age(a, 2), c) \xrightarrow{2} k(d, c)$ are derivable.

Unfortunately, the deduction rules for aging lead to counterintuitive results, as illustrated by the following example:

Example 20 *In the TRL theory*

$$\{\, [l_1] : f(x) \xrightarrow{2} g(x),\ [l_2] : f(b) \xrightarrow{2} g(c),\ [l_3] : a \xrightarrow{2} b \,\},$$

one would expect $f(a) \xrightarrow{2} g(c)$ not *to be derivable. However, the sequents* $f(x) \xrightarrow{2} age(f(x), 2)$ *and* $age(f(x), 2) \xrightarrow{0} g(x)$ *are both derivable, and so are* $f(b) \xrightarrow{2} age(f(b), 2)$ *and* $age(f(b), 2) \xrightarrow{0} g(c)$.

The sequents $f(x) \xrightarrow{2} age(f(x), 2)$ *and* $a \xrightarrow{2} b$ *give* $f(a) \xrightarrow{2} age(f(b), 2)$ *by synchronous replacement, which in turn rewrites to* $g(c)$ *using* $age(f(b), 2) \xrightarrow{0} g(c)$. *Transitivity gives the undesired sequent* $f(a) \xrightarrow{2} g(c)$.

We can summarize the situation as follows. We have seen that the rewriting translation of a TRL theory $\mathcal{T}$ is looser than $\mathcal{T}$ itself, in some cases with some pleasant consequences. If we attempt to tighten the correspondence between both systems by adding aging rules to TRL, we get indeed closer, but we unfortunately encounter paradoxical examples in the reformulation of TRL.

5 Concluding Remarks

We have presented a general method for specifying real-time and hybrid systems in rewriting logic in an executable way, have shown how a wide range of real-time and hybrid system models can be naturally expressed in rewriting logic, and have illustrated the ideas with several examples. This work should be further extended in several directions.

The systems that we have considered can be distributed and can exhibit concurrent computations, in which several components of the state can change simultaneously and independently. However, time is still in some sense *global*, since time acts on the global state, even though its effects can be local and distributed—for example, by advancing the local clocks of different distributed objects. The situation is entirely similar to that in some real-time models for distributed systems such as Lynch's *general timed automata* [24], where time also acts uniformly on all the distributed components. In fact, although we have not discussed general timed automata in this paper, they can also be specified within our general framework. Although the current framework can already be used for specifying and reasoning about a range of distributed time-based systems, it would be worth investigating how the assumption of global time action could be relaxed to local or distributed time actions.

We have explored what we think is a representative range of real-time and hybrid system models. However, the general timed (I/O) automata model mentioned above, real-time dataflow models such as Lustre's [18], and a variety of other models, including timed process algebra models, should also be specified in detail in rewriting logic. The interest is not merely conceptual: by using a formal meta-tool such as Maude [13], one can turn the rewriting logic specification of a model into a tool for executing and analyzing formal specifications in that model, as in fact we have done in the Real-Time Maude language and tool [38]. Since at present some formalisms lack execution and analysis environments, this offers a way of developing new formal tools with considerably less effort than what would be required for conventional implementations.

As mentioned above, the ideas presented in this paper have been used as the basis of the Real-Time Maude specification language and tool [38]. Real-Time Maude specifications extend the usual Maude specifications by allowing specifications of real-time rewrite theories. Such theories can be specified in *timed system modules* and *timed object-oriented modules*. Using the general reduction from real-time rewrite theories to ordinary rewrite theories described in Section 2.4, the Real-Time Maude tool then transforms timed system modules and timed object-oriented modules into ordinary Maude modules.

The Real-Time Maude tool provides the user with a family of execution strategies specially designed for simulating and formally analyzing real-time and hybrid system specifications. Strategies of this kind include several default execution strategies, analysis strategies for exploring future states, finding deadlocks and checking specific properties, and model checking strategies for a class of timed linear-time temporal logic formulas that can express invariants and liveness properties. Furthermore, the library of formal analysis strategies can be extended by the user with new application-specific strategies. We have used the Real-Time Maude tool to specify, simulate, and formally analyze a number of applications, including the AER/NCA suite of active network communication protocols [21], scheduling problems, the *timed automaton with deadlines* model of Bornot, Sifakis, and Tripakis [8], and several other benchmarks.

Much work remain ahead. On the one hand, the Real-Time Maude tool should be further developed, and the model checking methods it supports should be optimized and should be more closely compared with those of other model-checking tools. On the other hand, the formal analysis methods should be complemented with abstraction and theorem proving techniques, and the precise details of a timed linear-time temporal logic well suited to be used in conjunction with real-time rewrite theories should be developed.

Acknowledgements

We cordially thank Christiano Braga, Manuel Clavel, Francisco Durán, Alexander Knapp, Piotr Kosiuczenko, Narciso Martí-Oliet, Sigurd Meldal, Joseph Sifakis, Mark-Oliver Stehr, Carolyn Talcott, Tomás Uribe, Martin Wirsing, and two anonymous referees for their comments and suggestions, that have helped us in the development of these ideas and in improving their presentation.

References

[1] W. M. P. van der Aalst. Interval timed coloured Petri nets and their analysis. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 453–472. Springer, 1993.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] M. Barr and C. Wells. *Category Theory for Computing Science.* Centre de Recherches Mathématiques, third edition, 1999.

[5] M. J. Beeson. *Foundations of Constructive Mathematics.* Springer, 1985.

[6] J. A. Bergstra and J. V. Tucker. Algebraic specification of computable and semicomputable data types. *Theoretical Computer Science*, 50:137–181, 1987.

[7] N. Bjørner, Z. Manna, H. B. Sipma, and T. E. Uribe. Deductive verification of real-time systems using STeP. In M. Bertran and T. Rus, editors, *Proc. of ARTS'97*, volume 1231 of *Lecture Notes in Computer Science*, pages 22–43. Springer, 1997.

[8] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In W.-P. de Roever, H. Langmaack, and A. Pnueli, editors, *Proc.*

Compositionality'97, volume 1536 of *Lecture Notes in Computer Science*, pages 103–129. Springer, 1998.

[9] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996. `http://www.elsevier.nl/locate/entcs/volume4.html`.

[10] M. Clavel. *Reflection in general logics and in rewriting logic, with applications to the Maude language*. PhD thesis, University of Navarre, 1998.

[11] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Martí-Oliet, and J. Meseguer. Metalevel computation in Maude. In C. Kirchner and H. Kirchner, editors, *Proc. 2nd Intl. Workshop on Rewriting Logic and its Applications*, volume 15 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998. `http://www.elsevier.nl/locate/entcs/volume15.html`.

[12] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. *Maude: Specification and Programming in Rewriting Logic*. Computer Science Laboratory, SRI International, Menlo Park, 1999. `http://maude.csl.sri.com`.

[13] M. Clavel, F. Durán, S. Eker, J. Meseguer, and M.-O. Stehr. Maude as a formal meta-tool. In J. M. Wing, J. Woodcock, and J. Davies, editors, *Proc. FM'99*, volume 1709 of *Lecture Notes in Computer Science*, pages 1684–1703. Springer, 1999.

[14] M. Clavel and J. Meseguer. Axiomatizing reflective logics and languages. In G. Kiczales, editor, *Proc. Reflection'96*, pages 263–288, 1996. `http://jerry.cs.uiuc.edu/reflection/`.

[15] M. Clavel and J. Meseguer. Reflection and strategies in rewriting logic. In J. Meseguer, editor, *Proc. 1st Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996. `http://www.elsevier.nl/locate/entcs/volume4.html`.

[16] K. Futatsugi and R. Diaconescu. CafeOBJ report. AMAST Series, World Scientific, 1998.

[17] J. A. Goguen and J. Meseguer. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.

[18] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[19] H. M. Hanisch. Analysis of place/transition nets with timed arcs and its application to batch process control. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 282–299. Springer, 1993.

[20] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997. See also HyTech home-page at `http://www-cad.eecs.berkeley.edu/~tah/HyTech/`.

[21] S. Kasera, S. Bhattacharyya, M. Keaton, D. Kiwior, J. Kurose, D. Towsley, and S. Zabele. Scalable fair reliable multicast using active services. Technical Report TR 99-44, University of Massachusetts, Amherst, CMPSCI, 1999.

[22] P. Kosiuczenko and M. Wirsing. Timed rewriting logic with an application to object-based specification. *Science of Computer Programming*, 28(2-3):225–246, 1997.

[23] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997. See also UPPAAL home-page at `http://www.uppaal.com/`.

[24] Nancy Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[25] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30:609–678, 1993.

[26] Z. Manna and A. Pnueli. Clocked transition systems. *Logic and Software Engineering*, pages 3–42, 1996. Also available as Stanford University CSD technical report STAN-CS-TR-96-1566.

[27] Z. Manna and H. Sipma. Deductive verification of hybrid systems using STeP. In T. A. Henzinger and S. Sastry, editors, *Hybrid Systems: Computation and Control*, volume 1386 of *Lecture Notes in Computer Science*, pages 305–318. Springer, 1998.

[28] J. Meseguer. General logics. In H.-D. Ebbinghaus et al., editor, *Logic Colloquium'87*, pages 275–329. North-Holland, 1989.

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

[30] J. Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In U. Montanari and V. Sassone, editors, *Proc. Concur'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 331–372. Springer, 1996.

[31] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proc. WADT'97*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1998.

[32] J. Meseguer. Research directions in rewriting logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, NATO Advanced Study Institute, Marktoberdorf, Germany, July 29 – August 6, 1997*, NATO ASI Series F: Computer and Systems Sciences 165, pages 347–398. Springer, 1998.

[33] J. Meseguer and U. Montanari. Petri nets are monoids. *Information and Computation*, 88:105–155, 1990.

[34] J. Meseguer, P. C. Ölveczky, and M.-O. Stehr. Rewriting logic as a unifying framework for Petri nets. Submitted for publication, 2000.

[35] S. Morasca, M. Pezzè, and M. Trubian. Timed high-level nets. *The Journal of Real-Time Systems*, 3:165–189, 1991.

[36] P. C. Ölveczky, P. Kosiuczenko, and M. Wirsing. An object-oriented algebraic steam-boiler control specification. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Application: Specifying and Programming the Steam-Boiler Control*, volume 1165 of *Lecture Notes in Computer Science*, pages 379–402. Springer, 1996.

[37] P. C. Ölveczky and J. Meseguer. Specifying real-time systems in rewriting logic. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996. `http://www.elsevier.nl/locate/entcs/volume4.html`.

[38] P. C. Ölveczky and J. Meseguer. Real-Time Maude: A tool for simulating and analyzing real-time and hybrid systems. In *Third International Workshop on Rewriting Logic and its Applications*, 2000. To appear.

[39] M. Rabin. Computable algebra: General theory and theory of computable fields. *Transactions of the American Mathematical Society*, 95:341–360, 1960.

[40] W. Reisig. *Petri Nets*, volume 4 of *EATCS monographs on Theoretical Computer Science*. Springer, 1985.

[41] The Stanford Temporal Prover. `http://www-step.stanford.edu/`.

[42] L. J. Steggles and P. Kosiuczenko. A timed rewriting logic semantics for SDL: A case study of the alternating bit protocol. In C. Kirchner and H. Kirchner, editors, *Proc. 2nd International Workshop on Rewriting Logic and its Applications*, volume 15 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998. `http://www.elsevier.nl/locate/entcs/volume15.html`.

[43] P. Viry. Rewriting: An effective model of concurrency. In C. Halatsis, D. Maritsas, G. Philokyprou, and S. Theodoridis, editors, *Proc. PARLE'94*, volume 817 of *Lecture Notes in Computer Science*, pages 648–660. Springer, 1994.

[44] S. Yovine. Kronos: A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1(1/2), 1997. See also Kronos home-page at `http://www-verimag.imag.fr/TEMPORISE/kronos/`.