An Algebraic Framework for Urgency

Sébastien Bornot and Joseph Sifakis
Sebastien.Bornot@imag.fr Joseph.Sifakis@imag.fr

Verimag, 2 rue Vignate, 38610 Gières, France

1 Introduction

Timed formalisms are extensions of untimed ones by adding *clocks*, real-valued variables that can be tested and modified at transitions. Clocks measure the time elapsed at states when some implicitly or explicitly given *time progress conditions* are satisfied. Timed automata, timed process algebras and timed Petri nets can be considered as timed formalisms.

The semantics of timed formalisms can be defined by means of transition systems that perform time steps or (timeless) transitions. Clearly, such transition systems must satisfy well-timedness requirements related with the possibility for time to progress forever. It is recognized that the compositional description of timed systems that satisfy even weak well-timedness requirements, is a non trivial problem. An inherent difficulty is that usually, the semantics of operators compose separately time steps and transitions by preserving urgency: time can progress in a system by some amount if all its components respect their time progress constraints. This leads to very elegant semantics based on a nice "orthogonality principle" between time progress and discrete state changes. Parallel composition and other operators have been defined according to this principle for timed process algebras and hybrid automata. However, composing independently time steps and transitions may easily introduce timelocks. It is questionable if the application of a strong synchronization rule for time progress is always appropriate. For instance, if two systems are in states from which they will never synchronize, it may be desirable not to further constrain time progress by the strong synchronization rule.

In several papers ([SY96,BS98,BST97]) we have studied compositional description methods that are based on "flexible" composition rules that relax urgency constraints so as to preserve a weak well-timedness property that we call *time reactivity*. The latter means that if no discrete transition can be executed from a state then time can progress. Contrary to other stronger properties, time reactivity is very easy to satisfy by relating directly time progress conditions and enabling conditions of discrete transitions. We have proposed a simple sub-class of timed automata, called *timed automata with deadlines* that are time reactive and we have shown how can be defined choice and parallel composition operators that preserve time reactivity. In this paper, we present a unified algebraic framework that encompasses the already presented results and provides laws for choice and parallel composition on timed systems, modulo strong bisimulation. The algebraic framework is characterized by the following.

- Timed systems are obtained as the composition of *timed actions* by using operators. A timed action is a discrete transition, labeled with an action name, a guard, a deadline and a jump. Guards and deadlines are predicates on clocks characterizing respectively, the states at which the action is enabled and the states at which the action becomes urgent (time progress stops). We require that the deadline implies the corresponding guard which guarantees time reactivity. The jumps are functions that specify clock assignments when the action is executed.
- The operators are timed extensions of untimed operators. They preserve both time reactivity and *activity* of components. The latter is the property meaning that if some action can be executed after waiting by some time in a component, then some action of the composed system can be executed after waiting by some (not necessarily the same) time.
  We propose timed extensions of choice and parallel composition operators that are associative and commutative and are related by an expansion theorem. Choice operators are parameterized by an order relation on actions that is proven to be useful, in particular to define parallel composition with maximal progress.
- In addition to the usual laws of untimed operators, timed operators satisfy specific laws reflecting the structure of timed actions and assumptions about their synchronization. We identify different synchronization modes that take into account the possibility of waiting of the components and study their properties.

The paper is organized as follows. Section 2 presents the basic model, which is essentially automata with clocks, an abstraction of timed automata without the usual restrictions on guards and assignments. Section 3 and section 4 present respectively, basic results on priority choice operators and parallel composition, such as associativity, activity preservation and the expansion theorem. Section 5 presents the algebraic framework.

2 Timed Systems

Definition 1. Timed systems
A Timed System is :

- A discrete labeled transition system $(S, \rightarrow, A)$ where
  - S is a discrete set of states
  - A is a finite vocabulary of actions
  - $\rightarrow \subseteq S \times A \times S$ is a discrete transition relation
- A dense set V of states isomorphic to $\mathbf{R}_+^n$
- A labeling function h mapping *discrete transitions*, elements of $\rightarrow$, into *timed transitions*: $h(s, a, s') = (s, (a, g, d, f), s')$, where
  - g, d are respectively the *guard* and the *deadline* of the transition. Guards and deadlines are unary predicates on V such that $d \Rightarrow g$.
  - f is a *jump* $f : V \rightarrow V$.

According to the above definition, a timed system can be obtained from an untimed one by associating with each action a a *timed action* (a, g, d, f).

Definition 2. Semantics of timed systems
A *state* of a timed system is a pair (s, v), where $s \in S$ is a discrete state and $v \in V$. We associate with a timed system a transition relation $\rightarrow \subseteq (S \times V \times (A \cup \mathbf{R}_+) \times (S \times V)$. Transitions labeled by elements of A correspond to *discrete state changes* while transitions labeled by non-negative reals correspond to *time steps*.

Given $s \in S$, if $\{(s, a_i, s_i)\}_{i \in I}$ is the set of all the discrete transitions issued from s and $h(s, a_i, s_i) = (s, (a_i, g_i, d_i, f_i), s_i)$ then :

- $\forall i \in I \ \forall v \in \mathbf{R}_+ \ . \ (s, v) \xrightarrow{a_i} (s_i, f_i(v_i))$ if $g_i(v)$.
- $(s, v) \xrightarrow{t} (s, v + t)$ if $\forall t' < t \ . \ c_s(v + t')$ where $c_s = \neg \bigvee_{i \in I} d_i$ and $v + t$ is the valuation obtained from v by increasing all the components of v by t.
  We call c_s the *time progress condition* associated with the discrete state s.

We consider timed systems such that for any state s the time progress condition c_s is right-open. The semantics of a timed system is its associated transition relation, modulo strong bisimulation.

Notice that the simplest timed system is a single transition labeled with the timed action (a, g, d, f). The guard g characterizes the set of states from which the timed transition is possible while the deadline d characterizes the subset of these states where the timed transition is enforced by stopping time progress. The relative position of d with respect to g determines the *urgency* of the action. For a given g, the corresponding d may take two extreme values: $d = g$, meaning that the action is *eager*, and $d = false$, meaning that the action is *lazy*. A particularly interesting case is the one of a *delayable* action where $d = g\downarrow$ is the *falling edge* of a right-closed guard g (cannot be disabled without enforcing its execution). The above cases are illustrated in figure 1.

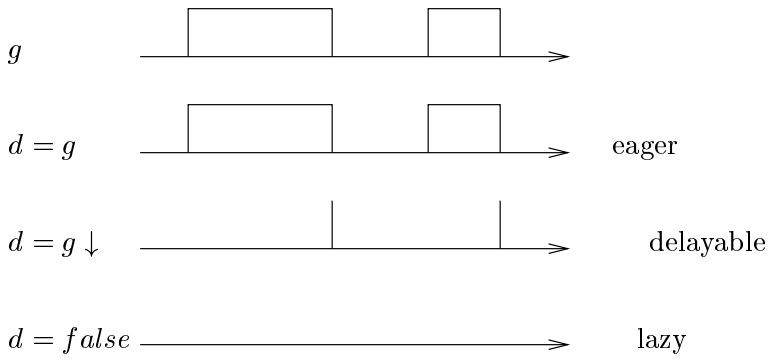


Fig. 1. Using deadlines to specify urgency.

The condition $d \Rightarrow g$ guarantees that if time cannot progress at some state, then some action is enabled from this state. Restriction to right-open time progress conditions guarantees that deadlines can be reached by continuous time trajectories and permits to avoid deadlock situations in the case of eager transitions. For instance, consider the case where $d = g = x > 2$, implying the time progress condition $x \leq 2$, which is not right-open. Then, if x is initially 2, time cannot progress by any delay t, according to definition 2 above. The guard g is not satisfied either. Thus, the system is deadlocked. The assumptions above ensure the property of *time reactivity*, that is, time can progress at any state unless some untimed transition is enabled.

3 Choice Operators

3.1 Non-deterministic Choice

Branching from a state s of a timed system can be considered as a non-deterministic choice operator between all the timed transitions issued from this state. The resulting untimed transition relation is the union of the untimed transition relations of the combined timed transitions. The resulting time step relation is the intersection of the time step relations of the combined timed transitions. We introduce standard process algebra notation to represent timed systems [BK85].

A discrete labeled transition system $(S, \rightarrow, A)$ can be represented as a set of equations of the form $s = \sum_{i \in I} \ a_i.s_i$ where $\{(s, a_i, s_i)\}_{i \in I}$ is the set of all the transitions issued from $s \in S$ and the right-hand sides of the equations are terms p of the form,

$$p ::= \ Nil \ \mid s \in S \ \mid \ a.p \ \mid \ p + p$$

where Nil is a constant and $a \in A$.

The semantics is defined, as usual, by the rules

$$a.p \stackrel{a}{\rightarrow} p$$
$$p_1 \stackrel{a}{\rightarrow} {p_1}' \ \text{ implies } \ p_1 + p_2 \stackrel{a}{\rightarrow} {p_1}' \ \text{ and } \ p_2 + p_1 \stackrel{a}{\rightarrow} {p_1}'$$
$$s = \sum_{i \in I} \ a_i.s_i \ \text{ implies } \ s \stackrel{a_i}{\rightarrow} s_i \forall i \in I$$

As usual, we consider that $+$ is an associative, commutative and idempotent operator with Nil as neutral element. The term $\sum_{i \in I} \ a_i.s_i$ is taken to be Nil, if $I = \emptyset$.

We extend the algebraic notation to timed systems $(S, A, \rightarrow, V, h)$ by replacing untimed actions by the corresponding timed actions via the labeling h. The timed extension of the term $s = \sum_i \ a_i.s_i$ is represented by the equation $s = \sum_i \ b_i.s_i$, if $h(s, a_i, s_i) = (s, b_i, s_i)$ with b_i of the form (a_i, g_i, d_i, f_i). We consider the b_i's as uninterpreted symbols and simplify the timed terms by assuming that $+$ is an associative commutative and idempotent operator with Nil as neutral element. This is obviously compatible with strong bisimulation.

3.2 Priority Choice

Motivation

It is often useful to consider that some priority is applied when from a given state several timed actions are enabled. Intuitively, applying priority implies preventing low priority actions from being executed when higher priority actions are enabled. This amounts to taking the non-deterministic choice between the considered actions by adequately restricting the guards of the actions with lower priority.

Consider, for example, two timed transitions $(s, (a_i, g_i, d_i, f_i), s_i)$, for $i = 1, 2$, with a common source state s. If action a_1 has lower priority than a_2 in the resulting timed system, the transition labeled by a_2 does not change while the transition labeled by a_1 would be of the form $(s, (a_1, g'_1, d'_1, r_1), s_1)$, where $g'_1 \Rightarrow g_1$ and $d'_1 = d_1 \wedge g'_1$.

For untimed systems, g'_1 is usually taken to be $g_1 \wedge \neg g_2$, which means that whenever a_1 and a_2 are simultaneously enabled, a_1 is disabled in the prioritized choice. However, for timed systems other ways to define g'_1 are possible. One may want to prevent action a_1 to be executed if it is established that a_2 will be eventually executed within a given delay. For this reason, we need the following notations.

Definition 3. Modal operators
Given a predicate p on V, we define the modal operators $\Diamond_k\ p$ ("eventually p within k") and $\diamondsuit_k\ p$ ("once p since k"), for $k \in \mathbf{R}_+ \cup \{\infty\}$.

$$\Diamond_k\ p\ (v) \text{ if } \exists t \in \mathbf{R}_+\ 0 \leq t \leq k.\ p(v+t)$$
$$\diamondsuit_k\ p\ (v) \text{ if } \exists t \in \mathbf{R}_+\ 0 \leq t \leq k.\ \exists v' \in V.\ v = v' + t \wedge p(v')$$

We write $\Diamond p$ and $\diamondsuit\ p$ for $\Diamond_\infty\ p$ and $\diamondsuit_\infty\ p$, respectively, and $\Box p$ and $\boxdot\ p$ for $\neg\Diamond\neg p$ and $\neg\diamondsuit\ \neg p$, respectively.

Coming back to the previous example, we can take $g'_1 = g_1 \wedge \neg\Diamond_k g_2$ or even $g'_1 = g_1 \wedge \Box\neg g_2$. In the former case, a_1 gives priority up to a_2 if a_2 is eventually enabled within k time units. In the latter case, a_1 is enabled only if a_2 is disabled forever.

Notice that for classes of timed systems such as timed automata [AD94] modalities can be eliminated to obtain predicates without quantifiers. For example, $\Diamond(1 \leq x \leq 2)$ is equivalent to $x \leq 2$. We shall be using in the sequel guards and deadlines with modalities.

Definition and Results

For timed systems, priorities between actions can be parameterized by the time actions of lower priority leave precedence to actions of higher priority. This motivates the following definition.

Definition 4. Priority order
Consider the relation $\prec \subseteq A \times (\mathbf{N} \cup \{\infty\}) \times A$. We write $a_1 \prec_k a_2$ for $(a_1, k, a_2) \in \prec$ and suppose that

- $\prec_k$ is a partial order relation for all $k \in \mathbf{N} \cup \{\infty\}$
- $a_1 \prec_k a_2$ implies $\forall k' < k.\ a_1 \prec_{k'} a_2$
- $a_1 \prec_k a_2 \wedge a_2 \prec_l a_3$ implies $a_1 \prec_{k+l} a_3$

Property : The relation $a_1 \ll a_2 = \exists k\ a_1 \prec_k a_2$ is an order relation.

Definition 5. Binary priority choice
Let $B_I = \{b_i\}_{i \in I}$ and $B_J = \{b_j\}_{j \in J}$ denote sets of timed actions with $b_i = (a_i, g_i, d_i, f_i)$, for $i \in I \cup J$. The operator $\widehat{+}$ is a binary operator on timed system defined by
$(\sum_{i \in I} b_i.s_i) \widehat{+} (\sum_{j \in J} b_j.s_j) = (\sum_{i \in I} (b_i \backslash B_J).s_i) + (\sum_{j \in J} (b_j \backslash B_I).s_j)$ with

$$
\begin{array}{l}
b_i \backslash B_J = (a_i, g_i \backslash B_J, d_i \backslash B_J, f_i) \\
g_i \backslash B_J = g_i \wedge \bigwedge_{(a_j, g_j, d_j, f_j) \in B_J, a_i \prec_k a_j} \neg \Diamond_k g_j \\
d_i \backslash B_J = d_i \wedge g_i \backslash B_J = d_i \wedge \bigwedge_{(a_j, g_j, d_j, f_j) \in B_J, a_i \prec_k a_j} \neg \Diamond_k g_j
\end{array}
$$

and the $b_j \backslash B_I$'s are defined in a similar manner.

From the above definition it is clear that priority restrictions are applied mutually with respect to actions that are not on the same side of the operator $\widehat{+}$.

Notice that if $a_1 \prec_k a_2$ then in $b_1.s_1 \widehat{+} b_s.s_2 = b_1 \backslash \{b_2\}.s_1 + b_2 \backslash \{b_1\}.s_2 = b_1 \backslash \{b_2\}.s_1 + b_2.s_2$, a_1 is disabled if a_2 will be enabled within k time units.

Consider the guards g_1, g_2 of the actions a_1, a_2. Figure 2 gives the guard $g'_1 = g_1 \backslash \{b_2\}$ obtained when g_1 is restricted by considering the priority orders $a_1 \prec_0 a_2$, $a_1 \prec_1 a_2$, $a_1 \prec_\infty a_2$.

For $b_i = (a_i, g_i, d_i, f_i)$, $i = 1, 2$, two timed actions, we write $b_1 = b_2$ if $a_1 = a_2$, $g_1 = g_2$, $d_1 = d_2$ and $f_1 = f_2$.

Lemma 6. *For a timed action b and sets of timed actions B, B_1, B_2,*

$$
\begin{array}{c}
b \backslash \{b\} \cup B = b \backslash B \\
(b \backslash B_1) \backslash B_2 = b \backslash (B_1 \cup B_2)
\end{array}
$$

Notice that the operator $\widehat{+}$ is commutative and Nil is the neutral element. Commutativity results from the symmetry of the definition and the commutativity of $+$. Moreover, for any term p, $p \widehat{+} Nil = p$ as Nil cannot restrict p and Nil is the neutral element for $+$.

It is important to notice that $\widehat{+}$ is not distributive with respect to $+$:

$$
\begin{array}{l}
(b_1.s_1 + b_2.s_2) \widehat{+} b_3.s_3 \neq (b_1.s_1 \widehat{+} b_3.s_3) + (b_2.s_2 \widehat{+} b_3.s_3) \text{ equivalent to} \\
\quad b_1 \backslash \{b_3\}.s_1 + b_2 \backslash \{b_3\}.s_2 + b_3 \backslash \{b_1, b_2\}.s_3 \neq \\
\qquad b_1 \backslash \{b_3\}.s_1 + b_3 \backslash \{b_1\}.s_3 + b_2 \backslash \{b_3\}.s_2 + b_3 \backslash \{b_2\}.s_3
\end{array}
$$

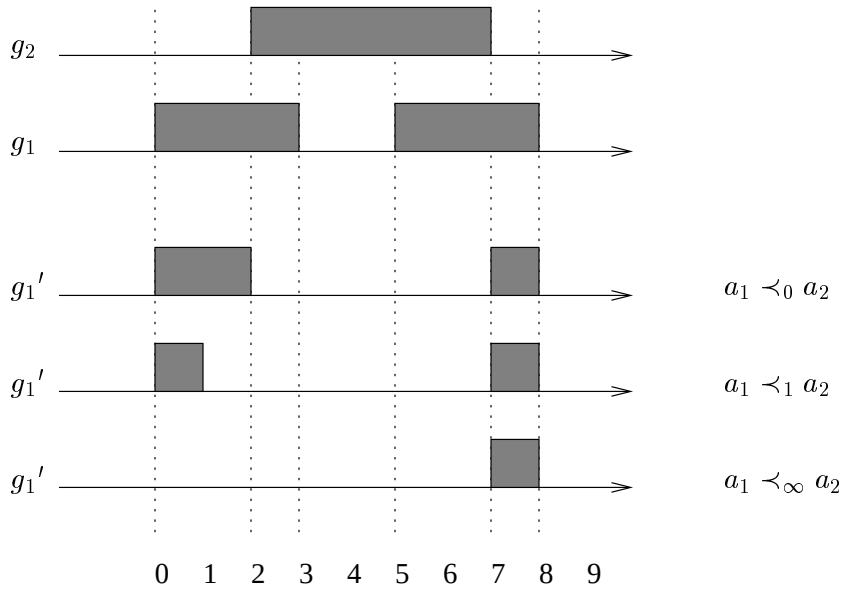


Fig. 2. Different priorities for a_2 over a_1

In fact, if a_3 (the label of b_3) is the action with the lowest priority then in $(b_1.s_1 + b_2.s_2)\widehat{+}b_3.s_3$, b_3 is restricted jointly by both b_1 and b_2 , while in $(b_1.s_1\widehat{+}b_3.s_3) + (b_2.s_2\widehat{+}b_3.s_3)$, the non deterministic choice of b_3 is restricted separately by b_1 and b_2.

Proposition 7. *The binary priority operator is associative i.e., for timed actions* $b_i = (a_i, g_i, d_i, f_i)$,

$$((\textstyle\sum_{i\in I} b_i.s_i)\widehat{+}(\sum_{j\in J} b_j.s_j))\ \widehat{+}(\sum_{k\in K} b_k.s_k) = (\textstyle\sum_{i\in I} b_i.s_i)\widehat{+}((\sum_{j\in J} b_j.s_j)\widehat{+}(\sum_{k\in K} b_k.s_k)$$

The above proposition allows the definition of a n-ary priority choice operator. We denote by $\widehat{\sum}_{i\in I} b_i.s_i$ the term obtained by combining the terms $\{b_i.s_i\}_{i\in I}$ by means of $\widehat{+}$.

Proposition 8. Reduction to non-deterministic choice
Priority choice can be expressed in terms of non-deterministic choice. For any set of terms $\{b_i.s_i\}_{i\in I}$ *with* $b_i = (a_i, g_i, d_i, f_i)$

$$\widehat{\sum_{i\in I}} b_i.s_i = \sum_{i\in I} b'_i.s_i$$

with $b'_i = (a_i, g'_i, d'_i, f_i)$, $g'_i = g_i \wedge \bigwedge_{a_i \prec_k a_j} \neg\Diamond_k g_j$ *and* $d'_i = d_i \wedge g'_i$.

Proposition 9. Activity preservation
The n-ary priority choice operator defined above satisfies the following properties.

1. $\Diamond g_i \Rightarrow \Diamond({g'}_i \vee \bigvee_{a_i \ll a_j} g_j)$
2. $\Diamond \bigvee_{i \in I} g_i = \Diamond \bigvee_{i \in I} {g'}_i$

This proposition has been proved in [BS98].

The first property means that if action a_i can occur in the non-prioritized choice then either a_i can occur in the prioritized choice or some action of higher priority.
The second property follows from the first and simply says that $\widehat{\sum}$ preserves activity : if some action can be executed in the non-prioritized choice then some action can be executed in the prioritized choice and vice versa.

Non-deterministic choice is a special case of priority choice when the priority order is empty. Priority choice is also commutative, associative, idempotent and Nil is the neutral element. For these reasons, we will use priority choice to describe terms, in the sequel.

4 Parallel Composition

In this section, we propose a general method for the definition of parallel composition operators for timed systems as an extension of parallel composition for untimed systems.

4.1 Parallel composition of untimed systems

We consider that for parallel composition of untimed terms the following framework is given.

- The action vocabulary A is provided with an operator $|$ such that $(A, |)$ is a commutative semi-group with absorbing element $\perp$. Words of this monoid represent the action resulting from the synchronization of their elements. The absorbing element $\perp$ means impossibility of synchronization.
- A *parallel composition operator* $\|$ on terms which is supposed to be associative, commutative, has Nil as neutral element and is defined by an expansion rule of the form:

  If $p_1 = \sum_{i \in J} a_i.s_i$ and $p_2 = \sum_{j \in J} a_j.s_j$ then

  $$p_1 \| p_2 = \sum_{i \in I'} a_i.(s_i \| p_2) + \sum_{j \in J'} a_j.(s_j \| p_1) + \sum_{(i,j) \in I \times J} a_i | a_j.(s_i \| s_j) \quad (\alpha)$$

  where I' and J' are subsets of I and J respectively.

  The first two summands correspond to behaviors starting with interleaving of actions. The sets of interleaving actions may be empty, depending on the semantics of $\|$. The third summand contains terms with synchronization transitions where only terms such that $a_i | a_j \neq \perp$ appear.

When such a parallel composition operator is used to compose sequential systems, it is important to combine interleaving and synchronization so as to satisfy two often conflicting requirements:

- *activity preservation*, that is, if in one of the components some action is enabled then in the product some action is enabled too.
- *maximal progress*, that is, when in the product both synchronization and interleaving transitions are enabled, synchronization is taken.

Clearly, it is easy to satisfy each requirement separately.

- If all the actions interleave ($I = I', J = J'$ in the expansion rule) then activity is preserved. However, in this case to achieve maximal progress the description language should provide with mechanisms for eliminating dynamically all the interleaving transitions that are systematically introduced. This is the approach adopted in languages such as CCS [Mil89] where all the actions interleave and a global restriction operator is often applied to prune off interleaving transitions.
- Maximal progress can be easily achieved by not allowing interleaving of actions that may synchronize. However, in this case there is an obvious risk of deadlock when the synchronization actions do not match. This point of view is adopted in languages such as CSP [Hoa85], where actions are partitioned into two classes, synchronizing and interleaving actions.

To our knowledge, there exists no specification methodology for writing untimed specifications satisfying both requirements. We show that such a methodology can be defined for timed systems due to the possibility of controlling waiting times by means of priority choice operators.

4.2 Parallel composition of timed systems

We extend the parallel composition operator $\|$ to timed systems in the following manner:

extension of $|$ We assume that the operator $|$ can be extended componentwise on the set B of timed actions b of the form (a, g, d, f) where $a \in A$, in such a manner that $(B, |)$ is a commutative semi-group with a distinguished absorbing element $\perp$. We take $(\perp, g, d, f) = \perp$ for any g, d, and f.
As ambiguity is resolved by the context, and to simplify notation, we overload the operator $|$.

extension of the priority order If $\prec$ is a priority order on A we suppose that it is preserved by $|$

$$\forall a_1, a_2, a_3 \in A \;.\; a_1 \prec_k a_2 \;\text{ implies }\; a_1|a_3 \prec_k a_2|a_3$$

extension of $\|$ The parallel composition operator $\|$ for timed systems is defined by extension of the expansion rule (α) for untimed terms, where b_i is the timed action associated with a_i.

$$\text{If } p_1 = \widehat{\sum_{i\in J}} b_i.s_i \quad \text{and} \quad p_2 = \widehat{\sum_{j\in J}} b_j.s_j \quad \text{then}$$

$$p_1 \| p_2 = \widehat{\sum_{i\in I'}} b_i.(s_i \| p_2) \;\widehat{+}\; \widehat{\sum_{j\in J'}} b_j.(s_j \| p_1) \;\widehat{+}\; \widehat{\sum_{(i,j)\in I\times J}} b_i|b_j.(s_i \| s_j)$$

Proposition 10. *The timed extension of $\|$ is associative, commutative, idempotent and Nil is the neutral element.*

Proposition 11. *If all the actions interleave then $\|$ preserves activity*

Proof. (hint) If in the expansion rule priority choice is replaced by non-deterministic choice, activity is trivially preserved due to the presence of interleaving actions. Proposition 9 says that replacing non-deterministic choice by priority choice preserves activity.

Proposition 12. *The parallel composition guarantees maximal progress if the priority order gives infinite priority to synchronizations :*

$$\forall a_1, a_2 \in A \;.\; a_1 \prec_\infty a_1|a_2 \quad \textit{and} \quad a_2 \prec_\infty a_1|a_2$$

5 The Algebraic Framework

In this section we develop an algebraic framework for the specification of timed systems by using (priority) choice and parallel composition. We study a simple algebra for the composition of timed actions and deduce laws for terms.

5.1 Composition of Guards and Deadlines

We show how the commutative semi-group $(B, |)$ can be defined. We assume that the composition of timed actions $b_i = (a_i, g_i, d_i, f_i), i = 1, 2$, is a timed action of the form $b_1|b_2 = (a_1|a_2, g_1|g_2, d_1|d_2, f_1|f_2)$.

The definition of $f_1|f_2$ does not pose particular problems. An associative and commutative operator $|$ can be defined on jumps (consider for instance, the easy case where synchronizing actions transform disjoint state spaces).

We suppose that the guard $g_1|g_2$ is defined as a monotonic function of g_1 and g_2 called *synchronization mode*, of the general form

$$g_1|g_2 = (g_1 \wedge m(g_2)) \vee (m(g_1) \wedge g_2)$$

where m is a function such that:

- $\forall g \,.\, g \Rightarrow m(g)$
- $\forall g, g' \,.\, m(g \vee g') = m(g) \vee m(g')$
- $\forall g, g' \,.\, m(g \wedge m(g') \vee m(g) \wedge g') = m(g \wedge g')$

Proposition 13. *For guards (state predicates) g_1, g_2 and $|$ synchronization mode,*

$$(g_1|g_2)|g_3 = g_1|(g_2|g_3)$$
$$g_1|g_2 = g_2|g_1$$
$$g_1 \wedge g_2 \Rightarrow g_1|g_2 \Rightarrow g_1 \vee g_2$$
$$(g_1 \vee g_2)|g_3 = (g_1|g_3) \vee (g_2|g_3)$$

The above properties imply that synchronization may occur only if at least one of the synchronizing actions is enabled. Furthermore, if both synchronizing actions are enabled at a state then synchronization is enabled. Distributivity of the composition of guards with respect to disjunction is an important property for the parallel composition to preserve strong bisimulation. More precisely, if S' is the system S where we replace a transition $s \stackrel{(a,g,d,f)}{\rightarrow} s'$ by the two transitions $s \stackrel{(a,g_1,d_1,f)}{\rightarrow} s'$ and $s \stackrel{(a,g_2,d_2,f)}{\rightarrow} s'$ such that $g = g_1 \vee g_2$ et $d = d_1 \vee d_2$ then S and S' are bisimilar and their parallel composition with a third system should give bisimilar systems.

In previous papers [BST97] we use the following synchronization modes for their practical interest:

- **and-synchronization** when $g_1|g_2 = g_1 \mathit{and}\ g_2 = g_1 \wedge g_2$.
- **max-synchronization** when $g_1|g_2 = g_1 \mathit{max}\ g_2 = (\diamondsuit\, g_1 \wedge g_2) \vee (g_1 \wedge \diamondsuit\, g_2)$. This condition characterizes synchronization with waiting.
- **min-synchronization** when $g_1|g_2 = g_1 \mathit{min}\ g_2 = (\diamond g_1 \wedge g_2) \vee (g_1 \wedge \diamond g_2)$. This condition characterizes synchronization by interrupt, in the sense that synchronization occurs when one of the two actions is enabled provided that the other will be enabled in the future.
- **or-synchronization** when $g_1|g_2 = g_1 \mathit{or}\ g_2 = g_1 \vee g_2$

It is trivial to check that the above functions are indeed synchronization modes.

For a given synchronization guard $g_1|g_2$, the associated deadline $d_1|d_2$ must be such that $d_1|d_2 \Rightarrow g_1|g_2$, to preserve time reactivity. On the other hand, it is desirable to preserve urgency which means $d_1|d_2 \Rightarrow d_1 \vee d_2$. For maximal urgency and time reactivity we take $d_1|d_2 = (g_1|g_2) \wedge (d_1 \vee d_2)$.

5.2 Laws for Extended Guards

We call *extended guard* any pair of predicates $G = (g, d)$ such that $d \Rightarrow g$. We extend the equivalence on predicates to equivalence on extended guards : if g_1 is equivalent to g_2 (noted $g_1 = g_2$) and d_1 is equivalent to d_2 (noted $d_1 = d_2$) then (g_1, d_1) is equivalent to (g_2, d_2) (noted $(g_1, d_1) = (g_2, d_2)$).

If $G_i = (g_i, d_i)$, for $i = 1, 2$, are two extended guards and $|$ is a synchronization mode, we take $G_1|G_2 = (g_1|g_2, g_1|g_2 \wedge (d_1 \vee d_2))$.

Proposition 14. *If* $g_1|g_2 = (g_1 \wedge m(g_2)) \vee (m(g_1) \wedge g_2)$ *and* $G_i = (g_i, d_i)$, *for* $i = 1, 2$, *then* $G_1|G_2 = (g_1|g_2, (d_1 \wedge m(g_2)) \vee (m(g_1) \wedge d_2))$.

This proposition says that the deadline of the synchronization guard has the same form as the synchronization guard. The following are useful laws that follow as a direct application of the proposition.

$G_1 and\ G_2 = (g_1 \wedge g_2, d_1 \wedge g_2 \vee g_1 \wedge d_2)$
$G_1 or\ G_2 = (g_1 \vee g_2, d_1 \vee d_2)$
$G_1 max\ G_2 = (g_1 max\ g_2, (d_1 \wedge \diamondsuit\, g_2) \vee (\diamondsuit\, g_1 \wedge g_2))$
$G_1 min\ G_2 = (g_1 min\ g_2, (d_1 \wedge \diamondsuit g_2) \vee (\diamondsuit g_1 \wedge g_2))$

Proposition 15. *For extended guards* $G_i = (g_i, d_i)$, $i = 1, 2, 3$, *and* $|$ *a synchronization mode, the following laws hold*

$(G_1|G_2)|G_3 = G_1|(G_2|G_3)$
$(G_1|G_2) = (G_2|G_1)$
$(G_1 or\ G_2)|G_3 = (G_1|G_3) or\ (G_2|G_3)$

It is important to notice that any expression involving extended guards and synchronization modes can be reduced to an equivalent extended guard.

5.3 Laws for Timed Actions

We naturally lift the structure of extended guards to timed actions $b = (a, G, f)$. For $b_i = (a_i, G_i, f_i), i = 1, 2$, we take

- $(a_1, G_1, f_1) = (a_2, G_2, f_2)$ if $a_1 = a_2$, $G_1 = G_2$ and $f_1 = f_2$.
- $\perp = (\perp, G, f)$

Proposition 16. *Let* B *be a set of timed actions on a vocabulary* A *as in paragraph 4.1.* $(B, |)$ *is a commutative semi-group with absorbing element* $\perp$ *where* $b_1|b_2 = (a_1|a_2, G_1|G_2, f_1|f_2)$, *for* $b_i = (a_i, G_i, f_i)$, $i = 1, 2$, *and* $|$ *is a given synchronization mode in* $G_1|G_2$.

The above proposition holds for a given synchronization mode. However, it can be easily extended to allow composition of timed actions with different synchronization modes under the following conditions.

Suppose that a partial function μ is given from A into the set of modes. If μ is defined for $a \in A$, $\mu(a)$ denotes the synchronization mode associated with a. We require that actions with different synchronization modes cannot synchronize, that is, $\mu(a_1) \neq \mu(a_2)$ implies $a_1|a_2 = \perp$.

It is trivial to check that $(B, |)$ with $b_1|b_2 = (a_1|a_2, G_1\mu(a_1)G_2, f_1|f_2)$ is a commutative semi-group with $\perp$ as absorbing element. We consider in the sequel, that parallel composition of timed systems is defined in terms of such a general synchronization function.

5.4 Laws for Timed Systems

For timed systems we take $\perp.s = Nil$, as $\perp$ means impossibility of synchronization.

Proposition 17. *The following laws hold for timed systems modulo strong bisimulation.*

- $\widehat{+}$ *is associative, commutative, idempotent, and* Nil *is the neutral element.*
- $\|$ *is associative, commutative, and* Nil *is the neutral element.*
- $$p_1 \| p_2 = \widehat{\sum_{i \in I'} b_i.(s_i \| p_2)} \ \widehat{+} \widehat{\sum_{j \in J'} b_j.(s_j \| p_1)} \ \widehat{+} \widehat{\sum_{(i,j) \in I \times J} b_i | b_j.(s_i \| s_j)}$$
- $b_1.s = b_2.s \quad if \quad b_1 = b_2$
- $(a, G_1 \, or \, G_2, f).s = (a, G_1, f).s \widehat{+} (a, G_2, f).s$

The laws for $\widehat{+}$ and $\|$ follow from propositions in section 3 and 4, respectively. They are extensions of well-known laws characterizing strong bisimulation for untimed systems. The two last laws are specific to timed systems and take into account properties of timed actions.

5.5 Typed Timed Actions

Given an extended guard $G = (g, d)$, it can be decomposed into $G = (g \wedge \neg d, \mathit{false}) \, or \, (d, d)$. That is, any extended guard can be expressed as the disjunction of one lazy and one eager guard. This remark motivates the definition of typed guards. If g is a guard, we write g^λ and g^ϵ to denote respectively, $g^\lambda = (g, \mathit{false})$ and $g^\epsilon = (g, g)$.

Proposition 18. *For* $\alpha \in \{\epsilon, \lambda\}$ *and a synchronization mode* $g_1 | g_2 = g_1 \wedge m(g_2) \vee m(g_1) \wedge g_2$,

- $g_1^\alpha | g_2^\alpha = (g_1 | g_2)^\alpha$
- $g_1^\epsilon \, or \, g_2^\lambda = g_1^\epsilon \, or \, (g_2 \wedge \neg g_1)^\lambda$
- $g_1^\epsilon | g_2^\lambda = (g_1 \wedge m(g_2))^\epsilon \, or \, (m(g_1) \wedge g_2)^\lambda$

A consequence of the above results is that any expression involving typed guards and synchronization modes can be reduced to the disjunction of disjoint eager and lazy guards.

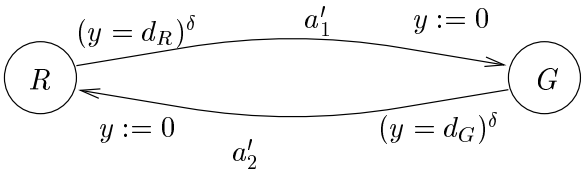
It is often useful to define a type of *delayable guards* denoted by δ. We take $g^\delta = g^\lambda \, or \, g\downarrow^\epsilon$, where $g\downarrow$ is the falling edge of the guard g.

Proposition 19. *Any expression involving delayable guards and the synchronization modes and, max, min, or, can be reduced into the disjunction of delayable guards.*

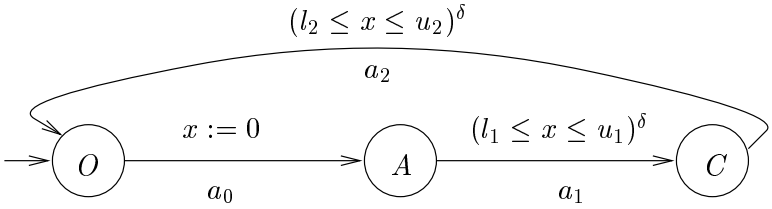
$$g_1^\delta \, and \, g_2^\delta = (g_1 \wedge g_2)^\delta$$
$$g_1^\delta \, max \, g_2^\delta = (g_1 \wedge \diamond\!\!\!\!- \, g_2)^\delta \, or \, (\diamond\!\!\!\!- \, g_1 \wedge g_2)^\delta$$
$$g_1^\delta \, min \, g_2^\delta = (g_1 \wedge \diamond g_2)^\delta \, or \, (\diamond g_1 \wedge g_2)^\delta$$

Using typed timed actions, drastically simplifies the general model. Furthermore, the most commonly used type, in practice, is delayable. The following example illustrates the use of max and min synchronization modes.

Example 20. **Traffic light for tramway crossing**



a. Traffic light



b. A Tramway

Fig. 3. Traffic light and Tramway

The light controlling the car traffic in a crossroads is a cyclic timed process with two states G (Green) and R (Red) and a clock y to enforce sojourn times d_G and d_R, respectively, at G and R (figure 3a).

We want to modify the light so as to control the traffic of tramways. When a tramway approaches the crossing, it sends a signal a_0 after which the light must be green within some interval $[l_1, u_1]$. This guarantees that the tramway crosses without stopping. Then, the light remains green until the tramway exits the crossing. Figure 3b represents a tramway as a process with states O (Out), A (Approach), C (Cross). We assume the tramway exits the cross section within time in the interval $[l_2, u_2]$ since the beginning of the approach phase.

The modified behavior of the light can be obtained as the parallel composition of the traffic light process and the tramway process by taking $\mu(a_1) = \mu(a'_1) = min$ and $\mu(a_2) = \mu(a'_2) = max$. The resulting timed controller handling one tramway (at most) is given in figure 4. It corresponds to the product of the two timed systems under the assumption of maximal progress and that all the actions interleave. The dashed transitions will never be taken due to higher priority of

synchronizations. The typed guards G_1, G'_1, G_{11} and G_{22} are the following:

$$G_{11} = (x \leq u_1 \wedge y = d_R)^\delta \vee (l_1 \leq x \leq u_1 \wedge y \leq d_R)^\delta$$
$$G_{22} = (l_2 \leq x \wedge y = d_G)^\delta \vee (l_2 \leq x \leq u_2 \wedge d_G \leq y)^\delta$$
$$G_1 = (l_1 \leq x \leq u_1 \wedge y > d_R)^\delta$$
$$G'_1 = (y = d_R \wedge x > l_1)^\delta.$$

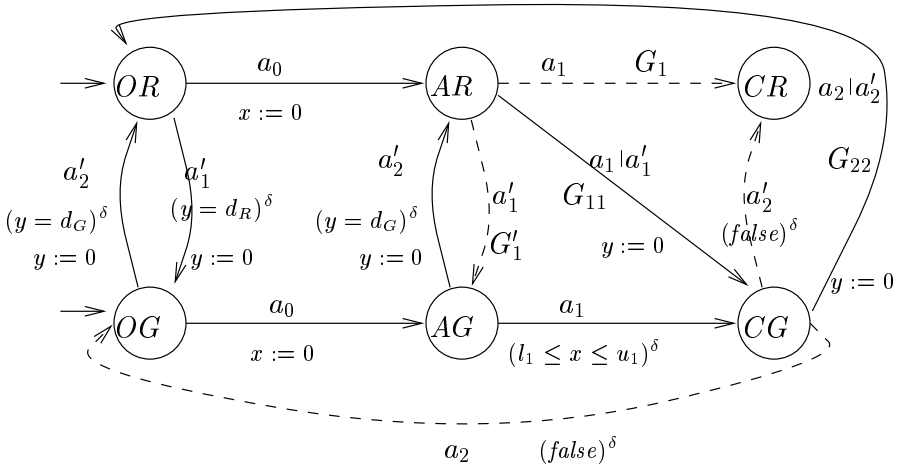


Fig. 4. Controller for a tramway

6 Discussion

The paper presents a framework for extending compositionally the description of untimed systems to timed systems by preserving time reactivity and activity of components. The adopted composition principle contrasts with the most commonly adopted so far which is strong synchronization for time progress and implies preservation of components urgency. Preserving time reactivity requires sometimes to relax urgency constraints, depending on synchronization modes associated with communication actions.

An important outcome of this work is that composition operators for untimed systems admit different timed extensions due to the possibility of controlling waiting times and "predicting" the future. The use of modalities in guards drastically increases concision in modeling and is crucial for compositionality. It does not imply extra expressive power for simple classes of timed systems, such as linear hybrid automata [ACH+95], where quantification over time in guards can be eliminated.

The definition of different synchronization modes has been motivated by the study of high level specification languages for timed systems, such as Timed Petri nets and their various extensions[SDdSS94,SDLdSS96,JLSIR97]. We have shown that the proposed framework is a basis for the study of the underlying semantics and composition techniques; if they are bounded then they can be represented as timed systems with finite control. Another outstanding fact is that using max-synchronization and min-synchronization, in addition to and-synchronization, drastically helps keeping the complexity of the corresponding timed system low [BST97].

The results concerning the algebraic framework itself are very recent. We are currently studying their application to the compositional generation of timed models of real-time applications.

References

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[BK85] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, May 1985. Fundamental studies.

[BS98] S. Bornot and J. Sifakis. *On the composition of hybrid systems.* Spinger-Verlag, Berkeley, March 1998.

[BST97] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *International Symposium: Compositionality - The Significant Difference*, Malente (Holstein, Germany), September 1997. Lecture Notes in Computer Science 1536, Springer Verlag.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[JLSIR97] M. Jourdan, N. Layaida, L. Sabry-Ismail, and C. Roisin. An integrated authoring and presentation environment for interactive multimedia documents. In *4th Conference on Multimedia Modeling*, Singapore, November 1997. World Scientific Publishing.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[SDdSS94] P. Sénac, M. Diaz, and P. de Saqui-Sannes. Toward a formal specification of multimedia scenarios. *Annals of telecomunications*, 49(5-6):297–314, 1994.

[SDLdSS96] P. Sènac, M. Diaz, A. Léger, and P. de Saqui-Sannes. Modeling logical and temporal synchronization in hypermedia systems. In *Journal on Selected Areas in Communications*, volume 14. IEEE, jan. 1996.

[SY96] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *13th Annual Symposium on Theoretical Aspects of Computer Science, STACS'96*, pages 347–359, Grenoble, France, February 1996. Lecture Notes in Computer Science 1046, Spinger-Verlag.