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 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 describes the algebraic framework. Two examples illustrating its use are given in section 6. We conclude by discussing future work directions and relations to existing work.

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* $f : V \to V$.

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 $\to \subseteq (S \times V \times (A \cup \mathbf{R}_+) \times (S \times V)$. 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 :

- $\forall i \in I \; \forall v \in \mathbf{R}_+ \; . \; (s, v) \xrightarrow{a_i} (s_i, f_i(v))$ 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 [Par81,Mil83,Mil89].

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), defined by

$$g\downarrow (v) = g(v) \wedge \exists \epsilon > 0 \; . \; \forall \epsilon' \in (0, \epsilon] \; . \; \neg g(v + \epsilon').$$

The above cases are illustrated in figure 1.

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.

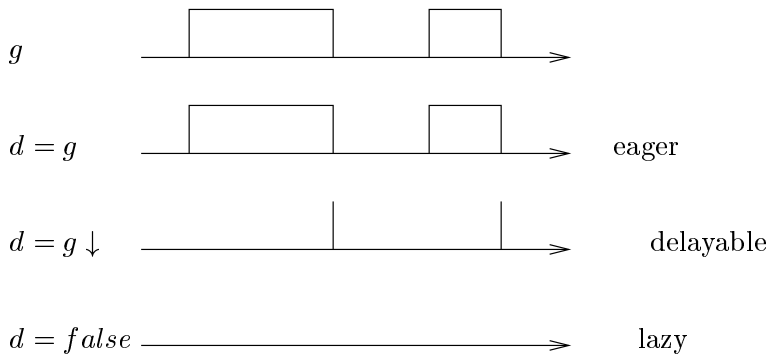


Fig. 1. Using deadlines to specify urgency.

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,Mil83].

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

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

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 $s = \sum_i \ b_i.s_i$, where $h(s, a_i, s_i) = (s, b_i, s_i)$ with b_i of the form (a_i, g_i, d_i, f_i).

Definition 3. *Observational equivalence*
For two terms s_1, s_2 we say that they are observationally equivalent, if for any valuation $v \in V$, the states (s_1, v) and (s_2, v) are bisimilar.

As observational equivalence is defined in terms of bisimulation of transition systems and admits no syntactical characterization, we prefer working with a stronger equivalence which is behavioral congruence.

Definition 4. *Behavioral congruence*
Behavioral congruence is the least congruence induced by the following rules

$s_1 + s_2 = s_2 + s_1$	*commutativity*
$(s_1 + s_2) + s_3 = s_1 + (s_2 + s_3)$	*associativity*
$s + s = s$	*idempotence*
$s + Nil = s$	*neutralelement*
$b_1 = b_2$ implies $b_1.s = b_2.s$	

Clearly if two terms are behaviorally congruent, then they are observationally equivalent.

Throughout this section, equality of timed terms is behavioral congruence.

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 5. 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 $k \in \mathbf{R}_+ \cup \{\infty\}$.

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

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

Coming back to the previous example, we can take $g_1' = g_1 \wedge \neg\diamondsuit_k g_2$ 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, $\diamondsuit(1 \leq x \leq 2)$ 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 6. Priority order
Consider the relation $\prec \subseteq A \times (\mathbf{R}_+ \cup \{\infty\}) \times A$. 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 $k \in \mathbf{R}_+ \cup \{\infty\}$
- $a_1 \prec_k a_2$ 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.

Proof. $\ll$ is antireflexive and transitive by definition. It is antisymmetric : if $a_1 \prec_k a_2$ then for every $k' \leq k$, $a_1 \prec_{k'} a_2$ and since $\prec_0$ is antisymmetric, $a_2 \prec_0 a_1$ does not hold; this implies that for any $k' \in \mathbf{R}_+ \cup \{\infty\}$, $a_2 \prec_{k'} a_1$ does not hold. □

Definition 7. 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

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

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

Notice that $\widehat{+}$ preserves behavioral congruence in the sense that if $s_1 = s_1'$ then $s_1 \widehat{+} s_2 = s_1' \widehat{+} s_2$. This definition introduces $\widehat{+}$ as a macronotation : any term with priority choice can be expanded into a term with non-deterministic choice (its meaning). The equality of terms with priority choice operators is the behavioral congruence of their meanings.

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.

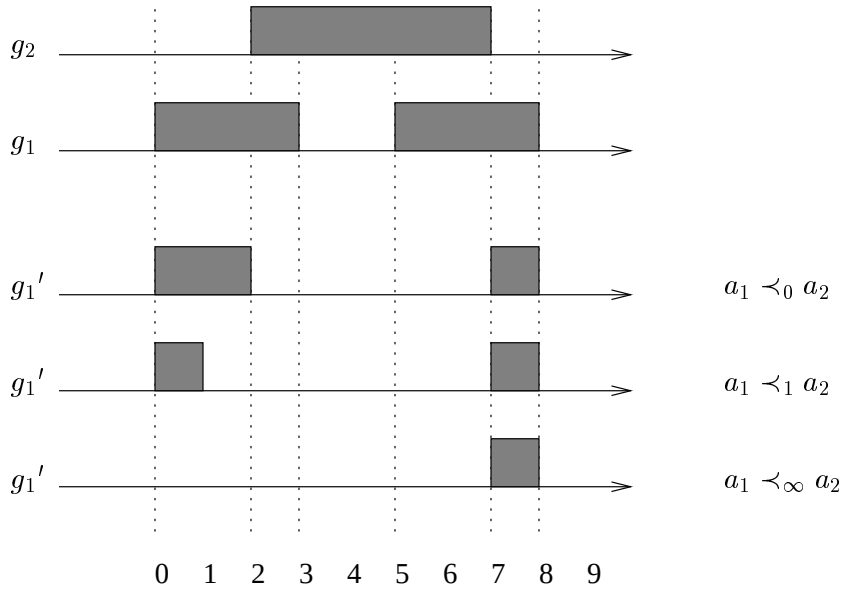


Fig. 2. The restricted guard g_1' for different degrees of priority

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

$$b\backslash\{b\} \cup B = b\backslash B$$
$$(b\backslash B_1)\backslash B_2 = b\backslash(B_1 \cup B_2)$$

Proof. Let $b = (a, g, d, f)$.
The first property results from the fact that priority orders are antireflexive.
$b\backslash\{b\} \cup B = (a, g\backslash\{b\} \cup B, d\backslash\{b\} \cup B, f)$
with

$$\begin{aligned} g\backslash\{b\} \cup B &= g \wedge \textstyle\bigwedge_{(a_i,g_i,d_i,f_i)\in\{b\}\cup B,\ a\prec_k a_i} \neg\Diamond_k g_i \\ &= g \wedge \textstyle\bigwedge_{(a_i,g_i,d_i,f_i)\in B,\ a\prec_k a_i} \neg\Diamond_k g_i \\ &= g\backslash B \end{aligned}$$

and

$$d\backslash\{b\} \cup B = d \wedge g\backslash\{b\} \cup B = d \wedge g\backslash B = d\backslash B$$

That is, $b\backslash\{b\} \cup B = b\backslash B$.

For the second property , we have by direct application of definition 5 :

$$(b\backslash B_1)\backslash B_2 = (a, g\backslash B_1, d\backslash B_1, f)\backslash B_2 = (a, (g\backslash B_1)\backslash B_2, (d\backslash B_1)\backslash B_2, f)$$

Let us compute $(g\backslash B_1)\backslash B_2$:

$$\begin{aligned} (g\backslash B_1)\backslash B_2 &= (g \wedge \textstyle\bigwedge_{(a_i,g_i,d_i,f_i)\in B_1,\ a\prec_k a_i} \neg\Diamond_k g_i)\backslash B_2 \\ &= (g \wedge \textstyle\bigwedge_{(a_i,g_i,d_i,f_i)\in B_1,\ a\prec_k a_i} \neg\Diamond_k g_i) \\ &\quad \wedge \textstyle\bigwedge_{(a_i,g_i,d_i,f_i)\in B_2,\ a\prec_k a_i} \neg\Diamond_k g_i \\ &= g \wedge \textstyle\bigwedge_{(a_i,g_i,d_i,f_i)\in B_1\cup B_2,\ a\prec_k a_i} \neg\Diamond_k g_i \\ &= g\backslash(B_1 \cup B_2) \end{aligned}$$

This implies

$$\begin{aligned} (d\backslash B_1)\backslash B_2 &= (d \wedge g\backslash B_1)\backslash B_2 = (d \wedge g\backslash B_1) \wedge (g\backslash B_1)\backslash B_2 \\ &= d \wedge g\backslash(B_1 \cup B_2) = d\backslash(B_1 \cup B_2) \end{aligned}$$

□

It will be shown that the operator $\widehat{+}$ is commutative and Nil is the neutral element. However, it is important to notice that $\widehat{+}$ is not distributive with respect to $+$:

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

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 $(b_1.s_1\widehat{+}b_3.s_3) + (b_2.s_2\widehat{+}b_3.s_3)$, b_3 is restricted separately by b_1 and b_2.

However , $\widehat{+}$ is associative as it will be shown in proposition 10. Associativity is an important property which is satisfied due to the adequate definition of priority orders. In particular, the transitivity property is crucial for achieving associativity, as it is shown by the following example.

Example 9. Consider the timed terms $p = (b_1.p_1 \widehat{+} b_2.p_2) \widehat{+} b_3.p_3$ and $q = b_1.p_1 \widehat{+} (b_2.p_2 \widehat{+} b_3.p_3)$ with $b_i = (a_i, g_i, d_i, f_i)$, $i = 1, 2, 3$. Suppose that $a_1 \prec_{10} a_2$ and $a_2 \prec_{10} a_3$ and that $a_1 \prec_d a_3$ for some $d \in \mathbf{R}_+$.

Then p and q are respectively equivalent to

$$p = (b_1 \backslash \{b_2\}) \backslash \{b_3\}.s_1 + b_2 \backslash \{b_3\}.s_2 + b_3.s_3$$
$$q = b_1 \backslash \{b_2 \backslash \{b_3\}, b_3\}.s_1 + b_2 \backslash \{b_3\}.s_2 + b_3.s_3$$

For $\widehat{+}$ to be associative, the guard g_1' of $(b_1 \backslash \{b_2\}) \backslash \{b_3\}$, $g_1' = g_1 \wedge \neg \Diamond_{10} g_2 \wedge \neg \Diamond_d g_3$, and the guard g_1'' of $b_1 \backslash \{b_2 \backslash \{b_3\}, b_3\}$, $g_1'' = g_1 \wedge \neg \Diamond_{10} (g_2 \wedge \neg \Diamond_{10} g_3) \wedge \neg \Diamond_d g_3$ must be equivalent.

Clearly $g_1' \Rightarrow g_1''$. Suppose that g_1'' is true at some valuation v and that $d < 20$. In that case, it is possible that $\neg \Diamond_{10} (g_2 \wedge \neg \Diamond_{10} g_3)(v)$ while $\Diamond_{10} g_2(v)$, as it is shown in figure 3. On the contrary, if $d \geq 20$ (the transitivity axiom is satisfied) then $\neg \Diamond_d g_3$ implies that $\neg \Diamond_{10} (g_2 \wedge \neg \Diamond_{10} g_3)$ is equivalent to $\neg \Diamond_{10} g_2$.

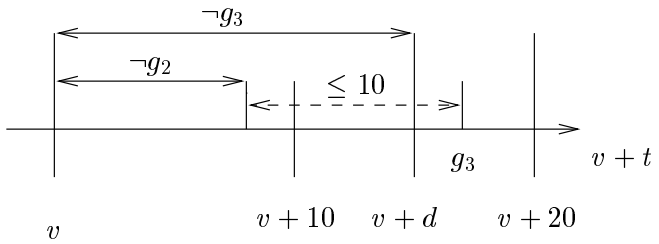


Fig. 3. Case $d < 20$

Proposition 10. *The binary priority operator is associative i.e., for timed actions* $b_i = (a_i, g_i, d_i, f_i)$,

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

Proof. We denote by B_I, B_J and B_K respectively the three sets $\{b_i\}_{i \in I}$, $\{b_j\}_{j \in J}$ and $\{b_k\}_{k \in K}$. We have to show the three following equalities :

$$\forall i \in I.\ (b_i \backslash B_J) \backslash B_K = b_i \backslash (\{b_j \backslash B_K\}_{j \in J} \cup \{b_k \backslash B_J\}_{k \in K})$$
$$\forall j \in J.\ (b_j \backslash B_I) \backslash B_K = (b_j \backslash B_K) \backslash B_I$$
$$\forall k \in K.\ b_k \backslash (\{b_i \backslash B_J\}_{i \in I} \cup \{b_j \backslash B_I\}_{j \in J}) = (b_k \backslash B_J) \backslash B_I$$

Due to the lemma this is equivalent to

$$\forall i \in I.\ b_i \backslash (B_J \cup B_K) = b_i \backslash (\{b_j \backslash B_K\}_{j \in J} \cup \{b_k \backslash B_J\}_{k \in K})$$
$$\forall k \in K.\ b_k \backslash (B_J \cup B_I) = b_k \backslash (\{b_j \backslash B_I\}_{j \in J} \cup \{b_i \backslash B_J\}_{i \in I})$$

It is then sufficient to show that :

$$\forall i \in I.\ g_i \backslash (B_J \cup B_K) = g_i \backslash (\{b_j \backslash B_K\}_{j \in J} \cup \{b_k \backslash B_J\}_{k \in K})$$

By definition of $g \backslash B$, this equality can be reduced to

$$\begin{aligned} &\textstyle\bigwedge_{j \in J,\ a_i \prec_{l_{ij}} a_j} \neg \Diamond_{l_{ij}} (g_j \backslash B_K) \wedge \bigwedge_{k \in K,\ a_i \prec_{l_{ik}} a_k} \neg \Diamond_{l_{ik}} (g_k \backslash B_J) \\ &= \textstyle\bigwedge_{j \in J,\ a_i \prec_{l_{ij}} a_j} \neg \Diamond_{l_{ij}} g_j \wedge \bigwedge_{k \in K,\ a_i \prec_{l_{ik}} a_k} \neg \Diamond_{l_{ik}} g_k \end{aligned}$$

for every i in I.

For a given i, we will now prove this by induction on the cardinality of $J \cup K$.

- The case $\mathrm{card}(J \cup K) = 1$ is trivial and left to the reader.
- Let us suppose that the property holds for all J' and K' such that $\mathrm{card}(J' \cup K') = n$,

$$\begin{aligned} &\textstyle\bigwedge_{j \in J',\ a_i \prec_{l_{ij}} a_j} \neg \Diamond_{l_{ij}} (g_j \backslash B_{K'}) \wedge \bigwedge_{k \in K',\ a_i \prec_{l_{ik}} a_k} \neg \Diamond_{l_{ik}} (g_k \backslash B_{J'}) \\ &= \textstyle\bigwedge_{j \in J',\ a_i \prec_{l_{ij}} a_j} \neg \Diamond_{l_{ij}} g_j \wedge \bigwedge_{k \in K',\ a_i \prec_{l_{ik}} a_k} \neg \Diamond_{l_{ik}} g_k. \end{aligned}$$

We will now show that this holds for all J and K such that $\mathrm{card}(J \cup K) = n + 1$.
Let a be an action of least priority in $J \cup K$:
$\forall j \in J \cup K,\ \neg(a_j \ll a)$
If a has no priority over a_i, then the property to prove is identical to the assumption. Let us suppose that a has priority over a_i, and (without loss of generality) that it appears in J : $a = a_{j_0}$. The property to be shown is then

$$\begin{aligned} &(\neg \Diamond_{l_{ij_0}} g_{j_0} \backslash B_K) \wedge \textstyle\bigwedge_{j \in (J \backslash \{j_0\}),\ a_i \prec_{l_{ij}} a_j} \neg \Diamond_{l_{ij}} (g_j \backslash B_K) \\ &\qquad\qquad \wedge \textstyle\bigwedge_{k \in K,\ a_i \prec_{l_{ik}} a_k} \neg \Diamond_{l_{ik}} (g_k \backslash B_J) \\ &= \neg \Diamond_{l_{ij_0}} g_{j_0} \wedge \textstyle\bigwedge_{j \in (J \backslash \{j_0\}),\ a_i \prec_{l_{ij}} a_j} \neg \Diamond_{l_{ij}} g_j \wedge \bigwedge_{k \in K,\ a_i \prec_{l_{ik}} a_k} \neg \Diamond_{l_{ik}} g_k. \end{aligned}$$

Since a_{j_0} has the least priority in $J \cup K$, we know that :

$$\forall k \in K.\ g_k \backslash B_J = g_k \backslash (B_J \backslash \{b_{j_0}\})$$

We can use the induction hypothesis on $(J \backslash \{j_0\}) \cup K$:

$$\begin{aligned} &(\neg \Diamond_{l_{ij_0}} g_{j_0} \backslash B_K) \wedge \textstyle\bigwedge_{j \in (J \backslash \{j_0\}),\ a_i \prec_{l_{ij}} a_j} \neg \Diamond_{l_{ij}} (g_j \backslash B_K) \\ &\qquad\qquad \wedge \textstyle\bigwedge_{k \in K,\ a_i \prec_{l_{ik}} a_k} \neg \Diamond_{l_{ik}} (g_k \backslash (B_J \backslash \{b_{j_0}\})) \\ &= (\neg \Diamond_{l_{ij_0}} g_{j_0} \backslash B_K) \wedge \textstyle\bigwedge_{j \in (J \backslash \{j_0\}),\ a_i \prec_{l_{ij}} a_j} \neg \Diamond_{l_{ij}} g_j \wedge \bigwedge_{k \in K,\ a_i \prec_{l_{ik}} a_k} \neg \Diamond_{l_{ik}} g_k. \end{aligned}$$

Since $\Diamond_k$ is distributive with respect to disjunction and since $\Diamond_l \Diamond_k g = \Diamond_{l+k} g$, we have :

$$\begin{aligned} \textstyle\bigwedge_{k \in K,\ a_i \prec_{l_{ik}} a_k} \neg \Diamond_{l_{ik}} g_k &= \textstyle\bigwedge_{k \in K,\ a_i \prec_{l_{ik}} a_k} \neg \Diamond_{l_{ij_0}} \Diamond_{l_{j_0} k} g_k \\ &= \neg \Diamond_{l_{ij_0}} \textstyle\bigvee_{k \in K,\ a_i \prec_{l_{ik}} a_k} \Diamond_{l_{j_0} k} g_k \\ &= \neg \Diamond_{l_{ij_0}} \textstyle\bigvee_{k \in K,\ a_{j_0} \prec_{l_{ij_0}} a_k} \Diamond_{l_{j_0} k} g_k \end{aligned}$$

Let us take $G = \bigvee_{k \in K,\ a_{j_0} \prec_{l_{ij_0}} a_k} \Diamond_{l_{j_0 k}} g_k$. Then, the following holds :

$$
\begin{aligned}
&(\neg\Diamond_{l_{ij_0}} g_{j_0} \backslash B_K) \wedge \textstyle\bigwedge_{j \in (J \backslash \{j_0\}),\ a_i \prec_{l_{ij}} a_j} \neg\Diamond_{l_{ij}} (g_j \backslash B_K) \\
&\qquad\qquad \wedge \textstyle\bigwedge_{k \in K,\ a_i \prec_{l_{ik}} a_k} \neg\Diamond_{l_{ik}} (g_k \backslash (B_J \backslash \{b_{j_0}\})) \\
&= (\neg\Diamond_{l_{ij_0}} (g_{j_0} \wedge \neg G)) \wedge \textstyle\bigwedge_{j \in (J \backslash \{j_0\}),\ a_i \prec_{l_{ij}} a_j} \neg\Diamond_{l_{ij}} g_j \wedge \neg\Diamond_{l_{ij_0}} G \\
&= (\neg\Diamond_{l_{ij_0}} ((g_{j_0} \wedge \neg G) \vee G)) \wedge \textstyle\bigwedge_{j \in (J \backslash \{j_0\}),\ a_i \prec_{l_{ij}} a_j} \neg\Diamond_{l_{ij}} g_j \\
&= (\neg\Diamond_{l_{ij_0}} (g_{j_0} \vee G)) \wedge \textstyle\bigwedge_{j \in (J \backslash \{j_0\}),\ a_i \prec_{l_{ij}} a_j} \neg\Diamond_{l_{ij}} g_j \\
&= (\neg\Diamond_{l_{ij_0}} g_{j_0}) \wedge \neg\Diamond_{l_{ij_0}} G \wedge \textstyle\bigwedge_{j \in (J \backslash \{j_0\}),\ a_i \prec_{l_{ij}} a_j} \neg\Diamond_{l_{ij}} g_j \\
&= \neg\Diamond_{l_{ij_0}} g_{j_0} \wedge \textstyle\bigwedge_{j \in (J \backslash \{j_0\}),\ a_i \prec_{l_{ij}} a_j} \neg\Diamond_{l_{ij}} g_j \wedge \textstyle\bigwedge_{k \in K,\ a_i \prec_{l_{ik}} a_k} \neg\Diamond_{l_{ik}} g_k
\end{aligned}
$$

QED.

□

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 11. *The priority choice operator $\widehat{+}$ is commutative, idempotent and Nil is the neutral element.*

Proof. Directly from the definition, $\widehat{+}$ is commutative and Nil is the neutral element. It is trivial that $p \widehat{+} p = p$ for all term of the form $p = b.s$, for some timed action b and some term s. By associativity of $\widehat{+}$, this equality can be generalized to all terms p, that is, $\widehat{+}$ is idempotent. □

Proposition 12. 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 $d'_i = d_i \wedge g'_i$. That is $b'_i = b_i \backslash \{b_j\}_{j \in I}$.

Proof. Immediate by induction on I, with the help of the two previous propositions.□

This proposition shows the global effect of prioritization on the initial actions of the terms of a priority choice operator by taking into account associativity and commutativity. It implies that behavioral congruence of terms with priority is preserved by associativity and commutativity of $\widehat{+}$. Thus, it is a congruence with respect to $\widehat{+}$.

This result allows to consider $\widehat{+}$ not only as a macronotation but also as a basic operator.

Definition 13. *Priority congruence*
Consider the language of terms obtained by replacing non-deterministic choice by priority choice in the definition of paragraph 3.1. Priority congruence is the least congruence induced by the following rules

$s_1 \widehat{+} s_2 = s_2 \widehat{+} s_1$	commutativity
$(s_1 \widehat{+} s_2) \widehat{+} s_3 = s_1 \widehat{+} (s_2 \widehat{+} s_3)$	associativity
$s \widehat{+} s = s$	idempotence
$s \widehat{+} Nil = s$	neutral element
$b_1 = b_2$ implies $b_1.s = b_2.s$	

Proposition 14. *If two terms are priority congruent then they are behaviorally congruent.*

Proof. Trivially follows from the fact that behavioral congruence is a congruence with respect to $\widehat{+}$. □

Proposition 15. Activity preservation
If $\widehat{\sum}_{i \in I} b_i.s_i = \widehat{\sum}_{i \in I} b'_i.s_i$ *as in proposition 10, then the followings properties hold between the guards* g_i *of* b_i *and the restricted guards* g'_i *of* b'_i.

1. $\Diamond g_i \Rightarrow \Diamond({g'}_i \vee \bigvee_{a_i \ll_{a_j}} g'_j)$, *for any* $i \in I$
2. $\Diamond \bigvee_{i \in I} g_i = \Diamond \bigvee_{i \in I} {g'}_i$

Proof. The proof of this property is a direct application of associativity of $\widehat{+}$. Let us consider a timed action $b = (a, g, d, f)$ with infinitely less priority than all actions in I ($\forall i \in I.\ a \prec_\infty a_i$) and a maximal guard ($g = true$). The reduced guard g' of b in

$$b.s \widehat{+} \widehat{\sum_{i \in I}} b_i.s_i$$

is $g' = true \backslash \{b'_i\}_{i \in I} = true \backslash \{b_i\}_{i \in I}$, which can be written $\bigwedge_{i \in I} \neg \Diamond g'_i = \bigwedge_{i \in I} \neg \Diamond g_i$ and gives the equality.

The first property is obtained by considering only the actions a_j having priority over a_i : $\Diamond(g_i \vee \bigvee_{a_i \ll_{a_j}} g_j) = \Diamond({g'}_i \vee \bigvee_{a_i \ll_{a_j}} g'_j)$. □

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 simply says that $\widehat{\sum}$ preserves activity of components : if some action can be executed in the non-prioritized choice then some action can be executed in the prioritized choice and vice versa.

The results of this section show that non-deterministic choice is a special case of priority choice when the priority order is empty. In this case, priority congruence and behavioral congruence coincide. Priority choice is actually a generalization of non-deterministic choice and for this reason we consider it as the choice operator, in the sequel. This allows to describe behaviors parameterized by a priority order.

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 a distinguished absorbing element $\perp \in A$. 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.

We show that for timed systems a parallel composition operation can be defined preserving process activity and maximal progress 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 $\mid$ We assume that the operator $\mid$ 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 $(B, \mid)$ 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 notation for $\mid$ and $\perp$.

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

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

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

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

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

Proposition 16. *For priority congruence, the parallel composition operator $\|$ defined above is associative, commutative, distributive with respect to $\widehat{+}$ and has Nil as neutral element.*

Proof. The proof is standard and similar to the one given in [Mil83] as priority congruence satisfies the same axioms as the strong congruence. It is based on the uniqueness of solution of well-guarded equations and on properties of $\widehat{+}$. □

Proposition 17. *If all the actions interleave then $\|$ preserves activity. That is, if g_i are the guards of b_i, $i \in I \cup J$, in the expansion rule, g_i' are the restricted*

guards of the interleaving actions, $i \in I \cup J$ and g_{ij} are the guards of $b_i|b_j$, $(i,j) \in I \times J$, then

$$\Diamond g_i \Rightarrow \Diamond(g'_i \vee \bigvee_{j\in J} g_i, j)$$
$$\Diamond(\bigvee_{i\in I} g_i \vee \bigvee_{j\in J} g_j) = \Diamond(\bigvee_{i\in I} g'_i \vee \bigvee_{j\in J} g'_j \vee \bigvee_{i,j\in I\times J} g_{ij})$$

Proof. 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 15 says that replacing non-deterministic choice by priority choice preserves activity. □

This proposition is a local deadlockfreeness preservation. If some action is possible in a component, then in the product, either this action can interleave or it can participate to a synchronization.

To achieve maximal progress in the expansion rule, it is sufficient to consider the priority order which gives infinite priority to synchronizations :

$$\forall a_1, a_2 \in A \;.\; a_1|a_2 \neq \bot \text{ implies } a_1 \prec_\infty a_1|a_2 \text{ and } a_2 \prec_\infty a_1|a_2$$

Example 18. Consider $b_1.s_1 \parallel b_2.s_2$ with $b_i = (a_i, g_i, d_i, id)$ such that $b_1|b_2 = (a_1|a_2, g_1 \wedge g_2, (d_1 \vee d_2) \wedge g_1 \wedge g_2, id)$ with $a_1|a_2 = \bot$, $g_1 = d_1 = (x = k_1)$, $g_2 = (y \leq k_2)$ and $d_2 = (y = k_2)$.

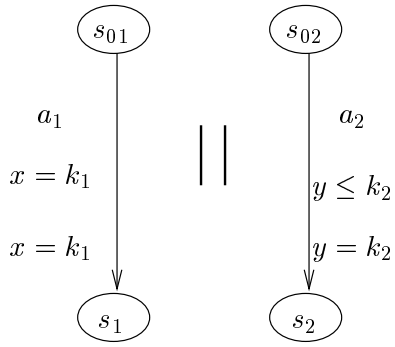
If a_1 and a_2 do not interleave, then $b_1.s_1 \parallel b_2.s_2 = (b_1|b_2)(s_1 \parallel s_2)$. We have maximal progress but if we start from states such that $\neg\Diamond((x = k_1) \wedge (y \leq k_2))$, we have a deadlock (figure 4b).

If actions a_1 and a_2 interleave and there is no priority between $a_1|a_2$ and these actions, then activity is preserved but either of the interleaving actions can be taken when synchronization is possible (figure 4c).

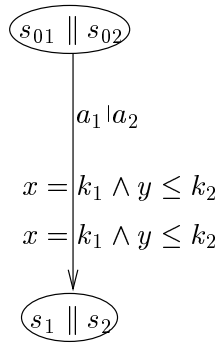
Finally, if actions a_1 and a_2 interleave and $a_1 \prec_\infty a_1|a_2, a_2 \prec_\infty a_1|a_2$ then activity is preserved due to proposition 17. Furthermore, we have maximal progress because the guards of the interleaving actions are respectively $g_1 \wedge \neg\Diamond(g_1 \wedge g_2)$ and $g_2 \wedge \neg\Diamond(g_1 \wedge g_2)$, which means that they can be taken only if the synchronization is disabled forever.

5 The Algebraic Framework

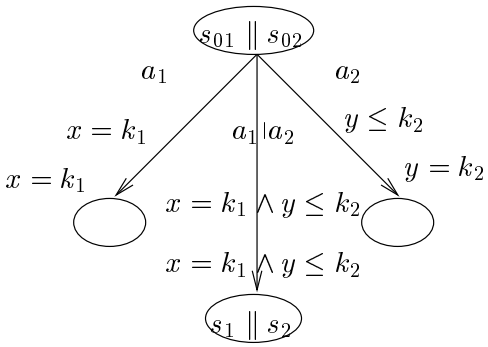
In this section we develop an algebraic framework for the specification of timed systems which takes into account the structure of timed actions. We study a simple algebra for the composition of timed actions and deduce two classes of laws for terms. The first class contains laws modulo priority congruence, resulting from the properties of priority choice and the definition of parallel composition operators. The second class contains laws reflecting properties of timed actions and preserving observational equivalence.



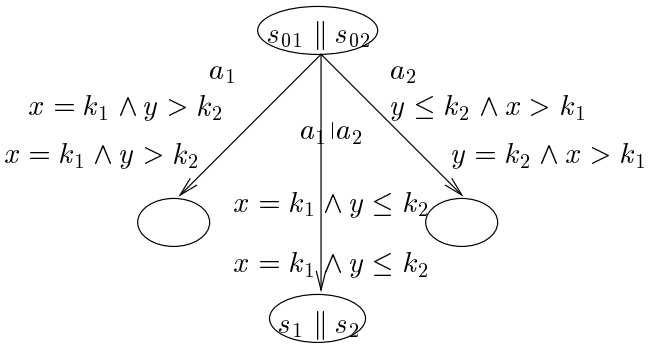
a. The components



b. No interleaving



c. Interleaving without priorities



d. Interleaving and priorities

Fig. 4. Parallel composition

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:

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

Proposition 19. *For guards (state predicates) g_1, g_2 and $|$ synchronization mode,*

$$\begin{gathered} g_1|g_2 = g_2|g_1 \\ (g_1|g_2)|g_3 = g_1|(g_2|g_3) \\ (g_1 \vee g_2)|g_3 = (g_1|g_3) \vee (g_2|g_3) \\ g_1 \wedge g_2 \Rightarrow g_1|g_2 \Rightarrow g_1 \vee g_2 \end{gathered}$$

Proof. Commutativity of $|$ follows directly from its definition.
Associativity is a simple application of the definition and the properties of m :

$$\begin{aligned} (g_1|g_2)|g_3 &= (m(g_1|g_2) \wedge g_3 \vee (g_1|g_2) \wedge m(g_3) \\ &= m(g_1) \wedge m(g_2) \wedge g_3 \vee m(g_1) \wedge g_2 \wedge m(g_3) \vee g_1 \wedge m(g_2) \wedge m(g_3) \end{aligned}$$

Due to commutativity of $|$, this is equal to $g_1|(g_2|g_3)$, too.
Distributivity with respect to disjunction is rather simple too :

$$\begin{aligned} (g_1 \vee g_2)|g_3 &= m(g_1 \vee g_2) \wedge g_3 \vee (g_1 \vee g_2) \wedge m(g_3) \\ &= (m(g_1) \vee m(g_2)) \wedge g_3 \vee g_1 \wedge m(g_3) \vee g_2 \wedge m(g_3) \\ &= (m(g_1) \wedge g_3 \vee g_1 \wedge m(g_3)) \vee (m(g_2) \wedge g_3 \vee g_2 \wedge m(g_3)) \\ &= (g_1|g_3) \vee (g_2|g_3) \end{aligned}$$

The last property is derived from $g_1 \Rightarrow m(g_1)$ and $g_2 \Rightarrow m(g_2)$, knowing that $g_1|g_2 = m(g_1) \wedge g_2 \vee g_1 \wedge m(g_2)$:

$$g_1 \wedge g_2 \vee g_1 \wedge g_2 \Rightarrow g_1|g_2$$

Moreover $g_1 \wedge m(g_2) \Rightarrow g_1$ and $g_2 \wedge m(g_1) \Rightarrow g_2$ imply :

$$g_1|g_2 \Rightarrow g_1 \vee g_2$$

□

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 as parallel composition distributes over choice operator. 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 $d = d_1 \vee d_2$ we would like that the parallel composition of S and S' with another system yields observationally equivalent systems.

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

- **and-synchronization** when $g_1 | g_2 = g_1 \, and \; g_2 = g_1 \wedge g_2$.
- **max-synchronization** when $g_1 | g_2 = g_1 \, max \; g_2 = (\diamondsuit \, g_1 \wedge g_2) \vee (g_1 \wedge \diamondsuit \, g_2)$. This condition characterizes synchronization with waiting.
- **min-synchronization** when $g_1 | g_2 = g_1 min \; g_2 = (\Diamond g_1 \wedge g_2) \vee (g_1 \wedge \Diamond g_2)$. 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 $g_1 | g_2 = g_1 \, or \; g_2 = g_1 \vee g_2$

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 20. *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* $G_1 | G_2 = (g_1 | g_2, (d_1 \wedge m(g_2)) \vee (m(g_1) \wedge d_2))$.

Proof. By definition, $G_1 | G_2 = (g_1 | g_2, (g_1 | g_2) \wedge (d_1 \vee d_2)$. Let us compute the deadline :

$$(g_1 | g_2) \wedge (d_1 \vee d_2) = (m(g_1) \wedge g_2 \vee g_1 \wedge m(g_2)) \wedge (d_1 \vee d_2)$$

Since $d_1 \Rightarrow g_1 \Rightarrow m(g_1)$, this can be reduced to :

$$\begin{aligned} (g_1 | g_2) \wedge (d_1 \vee d_2) &= d_1 \wedge g_2 \vee d_1 \wedge m(g_2) \vee m(g_1) \wedge d_2 \vee g_1 \wedge d_2 \\ &= d_1 \wedge m(g_2) \vee m(g_1) \wedge d_2 \end{aligned}$$

□

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 for $G_i = (g_i, d_i)$, $i = 1, 2$.

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

Proposition 21. *For extended guards $G_i = (g_i, d_i)$, $i = 1, 2, 3$, and $\mid$ a synchronization mode, the following laws hold*

$(G_1 \mid G_2) = (G_2 \mid G_1)$
$(G_1 \mid G_2) \mid G_3 = G_1 \mid (G_2 \mid G_3)$
$(G_1 \mathit{or}\ G_2) \mid G_3 = (G_1 \mid G_3) \mathit{or}\ (G_2 \mid G_3)$

Proof. Due to the previous proposition, $(G_1 \mid G_2) = (G_2 \mid G_1)$ is trivial.
Let us prove associativity. Remember that by definition of m, $m(g_1 \mid g_2) = m(g_1) \wedge m(g_2)$. This implies :

$$\begin{aligned}(G_1 \mid G_2) \mid G_3 &= ((g_1 \mid g_2), m(g_1) \wedge d_2 \vee d_1 \wedge m(g_2)) \mid (g_3, d_3) \\ &= ((g_1 \mid g_2) \mid g_3, m(g_1 \mid g_2) \wedge d_3 \vee \\ &\quad (m(g_1) \wedge d_2 \vee d_1 \wedge m(g_2)) \wedge m(g_3)) \\ &= ((g_1 \mid g_2) \mid g_3, m(g_1) \wedge m(g_2) \wedge d_3 \\ &\quad \vee m(g_1) \wedge d_2 \wedge m(g_3) \vee d_1 \wedge m(g_2) \wedge m(g_3))\end{aligned}$$

As the operator $\mid$ is associative on guards, this is equal to $G_1 \mid (G_2 \mid G_3)$ too.
The last equality is derived from the definitions :

$$\begin{aligned}(G_1 \mathit{or}\ G_2) \mid G_3 &= (g_1 \vee g_2, d_1 \vee d_2) \mid (g_3, d_3) \\ &= ((g_1 \vee g_2) \mid g_3, m(g_1 \vee g_2) \wedge d_3 \vee (d_1 \vee d_2) \wedge m(g_3) \\ &= ((g_1 \mid g_3) \vee (g_2 \mid g_3), \\ &\quad m(g_1) \wedge d_3 \vee m(g_2) \wedge d_3 \vee d_1 \wedge m(g_3) \vee d_2 \wedge m(g_3) \\ &= (g_1 \mid g_3, m(g_1) \wedge d_3 \vee d_1 \wedge m(g_3)) \mathit{or} \\ &\quad (g_2 \mid g_3, m(g_2) \wedge d_3 \vee d_2 \wedge m(g_3)) \\ &= (G_1 \mid G_3) \mathit{or}\ (G_2 \mid G_3)\end{aligned}$$

□

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 22. *Let B be a set of timed actions on a vocabulary A as in paragraph 4.2. $(B, |)$ 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 $G_1|G_2$.*

Proof. From the various definitions and from proposition 21, it follows that $|$ is associative and commutative on each component of the timed actions, so it is commutative and associative on timed actions. Moreover, $\perp$ (action) is the absorbing element on A, so $\perp$ (timed action) is the absorbing element on B. □

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

Proposition 23. *The congruence induced by the following laws on timed systems on $(B, |)$ is compatible with observational equivalence, i.e. if two terms are congruent then they are observationally equivalent.*

- $\widehat{+}$ *is associative, commutative, idempotent, and Nil is the neutral element.*
- $\|$ *is associative, commutative, distributive with respect to $\widehat{+}$, and Nil is the neutral element.*
- $\perp.s = Nil$
- *if $b_1 = b_2$ then $b_1.s = b_2.s$.*
- $(a, G_1 \text{ or } G_2, f).s = (a, G_1, f).s \widehat{+} (a, G_2, f).s$ *(which means that an action can be split into two actions of same label and reset, and whose union of guards is the initial guard)*
- *if all actions interleave and b is such that $b|b_j = \perp$ for any timed action b_j in B then*

$$b.s \widehat{+} \widehat{\sum_{i \in I}} b_i.s_i = b \backslash \{b_i\}_{i \in I}.s \widehat{+} \widehat{\sum_{i \in I}} b_i.s_i$$

Proof. This proof can be separated into two parts. The first part consists in checking that the rules are compatible with observational equivalence; this is trivial and left to the reader. The second part consist in checking that the induced congruence is compatible with observational equivalence, that is if $t_1 = t'_1$

and $t_2 = t'_2$, due to one of the rules, then $t_1 \widehat{+} t_2$ and $t_1 \parallel t_2$ are respectively observationally equivalent to $t'_1 \widehat{+} t'_2$ and $t'_1 \parallel t'_2$. Using the fact that we consider equivalences, we will only show that if $t_1 = t_2$, then for any timed system t, $t_1 \widehat{+} t$ is observationally equivalent to $t_2 \widehat{+} t$ and $t_1 \parallel t$ is observationally equivalent to $t_2 \parallel t$.

If $t_1 = t_2$ due to properties of $\widehat{+}$ or $\parallel$ then this property holds (see properties on section 3 and 4, respectively).

For, the rest of the rules, it is trivial to check that if $t_1 = t_2$ then for any t, $t_1 \widehat{+} t$ is observationally equivalent to $t_2 \widehat{+} t$. It is also trivial to check that for any s and t, $\perp.s \parallel t$ is observationally equivalent to $Nil \parallel t$ (which is equal to t), and if $b_1 = b_2$ then $b_1.s \parallel t$ is observationally equivalent to $b_2.s \parallel t$. We will only consider the last to cases.

Knowing that $(a, G_1 \, or \, G_2, f).s = (a, G_1, f).s \widehat{+} (a, G_2, f).s$, consider the term $(a, G_1 \, or \, G_2, f).s \parallel p$ with $p = \widehat{\sum}_{i \in I} b_i.s_i$ and $b_i = (a_i, G_i, f_i)$, $i \in I$. From properties of parallel of parallel composition and choice operators we have

$$
\begin{aligned}
(a, G_1 \, or \, G_2, f).s \parallel p &= (a, G_1 \, or \, G_2, f).(s \parallel p) \widehat{+} \widehat{\sum}_{i \in I} b_i.((a, G_1 \, or \, G_2, f).s \parallel s_i) \widehat{+} \\
&\quad \widehat{\sum}_{i \in I} (a, G_1 \, or \, G_2, f) | b_i.(s \parallel s_i) \\
&= (a, G_1, f).(s \parallel p) \widehat{+} (a, G_2, f).(s \parallel p) \widehat{+} \\
&\quad \widehat{\sum}_{i \in I} b_i.((a, G_1 \, or \, G_2, f).s \parallel s_i) \widehat{+} \\
&\quad \widehat{\sum}_{i \in I} (a|a_i, (G_1 \, or \, G_2)|G_i, f|f_i).(s \parallel s_i) \\
&= (a, G_1, f).(s \parallel p) \widehat{+} (a, G_2, f).(s \parallel p) \widehat{+} \\
&\quad \widehat{\sum}_{i \in I} b_i.((a, G_1 \, or \, G_2, f).s \parallel s_i) \widehat{+} \\
&\quad \widehat{\sum}_{i \in I} (a|a_i, (G_1|G_i) \, or \, (G_2|G_i), f|f_i).(s \parallel s_i) \\
&= (a, G_1, f).(s \parallel p) \widehat{+} (a, G_2, f).(s \parallel p) \widehat{+} \\
&\quad \widehat{\sum}_{i \in I} b_i.((a, G_1 \, or \, G_2, f).s \parallel s_i) \widehat{+} \\
&\quad \widehat{\sum}_{i \in I} (a|a_i, (G_1|G_i), f|f_i).(s \parallel s_i) \widehat{+} \\
&\quad \widehat{\sum}_{i \in I} (a|a_i, (G_2|G_i), f|f_i).(s \parallel s_i)
\end{aligned}
$$

For $((a, G_1, f).s \widehat{+} (a, G_2, f).s) \parallel p$ we get the same terms with the difference that in the second summand $(a, G_1 \, or \, G_2, f).s$ is replaced by $(a, G_1, f).s \widehat{+} (a, G_2, f).s$. The rest of the proof is standard and closely follows techniques given [Mil83] proving uniqueness of the solution of well-guarded equations.

Suppose now that all action interleave and b does not synchronize (for any $b_j \in B$, $b|b_j = \perp$). Consider the term $(b \backslash \{b_i\}_{i \in I}.s \widehat{+} \widehat{\sum}_{i \in I} b_i.s_i) \parallel p$ with $p =$

$\widehat{\sum}_{j\in J} b_j.s_j$. We have :

$$\begin{aligned}
(b\backslash\{b_i\}_{i\in I}.s \,\widehat{+}\widehat{\sum}_{i\in I} b_i.s_i) \parallel p \\
&= b\backslash\{b_i\}_{i\in I}.(s \parallel p)\widehat{+}\widehat{\sum}_{i\in I} b_i.(s_i \parallel p) \\
&\quad \widehat{+}\widehat{\sum}_{j\in J} b_j.((b\backslash\{b_i\}_{i\in I}\widehat{+}\widehat{\sum}_{i\in I} b_i.s_i) \parallel s_j) \\
&\quad \widehat{+}\widehat{\sum}_{j\in J}(b\backslash\{b_i\}_{i\in I})|b_j.(s \parallel s_j)\widehat{+}\widehat{\sum}_{i,j\in I\times J}(b_i|b_j).(s_i \parallel s_j) \\
&= b.(s \parallel p)\widehat{+}\widehat{\sum}_{i\in I} b_i.(s_i \parallel p)\widehat{+}\widehat{\sum}_{j\in J} b_j.((b\backslash\{b_i\}_{i\in I}\widehat{+}\widehat{\sum}_{i\in I} b_i.s_i) \parallel s_j) \\
&\quad \widehat{+}\widehat{\sum}_{j\in J}\bot.(s \parallel s_j)\widehat{+}\widehat{\sum}_{i,j\in I\times J}(b_i|b_j).(s_i \parallel s_j)
\end{aligned}$$

For $(b\widehat{+}\widehat{\sum}_{i\in I} b_i.s_i) \parallel p$ we get the same terms with the difference that in the third summand $b\backslash\{b_i\}_{i\in I}.s\widehat{+}\widehat{\sum}_{i\in I} b_i.s_i$ is replaced by $b\widehat{+}\widehat{\sum}_{i\in I} b_i.s_i$, and we can conclude as in previous case. □.

5.5 Typed Timed Actions

Given an extended guard $G = (g, d)$, it can be decomposed into $G = (g \wedge \neg d, \mathit{false})\, \mathit{or}\ (d, d)$. 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 24. *For $\alpha \in \{\epsilon, \lambda\}$ and a synchronization mode $g_1|g_2 = g_1 \wedge m(g_2) \vee m(g_1) \wedge g_2$,*

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

Proof. – Let us show that $g_1^\alpha|g_2^\alpha = (g_1|g_2)^\alpha$, for $\alpha \in \{\epsilon, \lambda\}$.

$$\begin{aligned}
g_1^\epsilon|g_2^\epsilon &= (g_1|g_2, m(g_1) \wedge g_2 \vee g_1 \wedge m(g_2)) \\
&= (g_1|g_2, g_1|g_2) = (g_1|g_2)^\epsilon \\
g_1^\lambda|g_2^\lambda &= (g_1|g_2, m(g_1) \wedge \mathit{false} \vee \mathit{false} \wedge g_2) \\
&= (g_1|g_2, \mathit{false}) = (g_1|g_2)^\lambda
\end{aligned}$$

- $g_1^\epsilon \mathit{or}\ g_2^\lambda = (g_1 \vee g_2, g_1) = (g_1, g_1) \mathit{or}\ (g_2 \wedge \neg g_1, \mathit{false}) = g_1^\epsilon \mathit{or}\ (g_2 \wedge \neg g_1)^\lambda$
  This provides a canonical decomposition for union (or) of typed guards.
- By applying the definitions :

$$\begin{aligned}
g_1^\epsilon|g_2^\lambda &= (m(g_1) \wedge g_2 \vee g_1 \wedge m(g_2), m(g_1) \wedge \mathit{false} \vee g_1 \wedge m(g_2)) \\
&= (m(g_1) \wedge g_2, \mathit{false}) \mathit{or}\ (g_1 \wedge m(g_2), g_1 \wedge m(g_2)) \\
&= (g_1 \wedge m(g_2))^\epsilon \mathit{or}\ (m(g_1) \wedge g_2)^\lambda
\end{aligned}$$

□

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

$$\begin{aligned} g_1^\delta \, and \, g_2^\delta &= (g_1 \wedge g_2)^\delta \\ g_1^\delta \, max \, g_2^\delta &= (g_1 \wedge \diamond\!\!\!\!- \, g_2)^\delta \, or \, (\diamond\!\!\!\!- \, g_1 \wedge g_2)^\delta \\ g_1^\delta \, min \, g_2^\delta &= (g_1 \wedge \diamond g_2)^\delta \, or \, (\diamond g_1 \wedge g_2)^\delta \end{aligned}$$

Proof. We will use the properties of the falling edge operator to prove this result. Namely, $(g_1 \wedge g_2)\downarrow = g_1 \wedge g_2 \downarrow \vee g_1 \downarrow \wedge g_2$, $(\diamond\!\!\!\!- \, g)\downarrow = false$ and $(\diamond g)\downarrow \Rightarrow g\downarrow$.

- For *and* , we have $m(g) = g$. It follows that :

$$\begin{aligned} g_1^\delta \, and \, g_2^\delta &= (g_1, g_1\downarrow) \, and \, (g_2, g_2\downarrow) \\ &= (g_1 \wedge g_2, g_1 \wedge g_2 \downarrow \vee g_1 \downarrow \wedge g_2) \\ &= (g_1 \wedge g_2, (g_1 \wedge g_2)\downarrow) \\ &= (g_1 \wedge g_2)^\delta \end{aligned}$$

- For *max* , $m(g) = \diamond\!\!\!\!- \, g$.

$$\begin{aligned} g_1^\delta \, max \, g_2^\delta &= (g_1, g_1\downarrow) \, max \, (g_2, g_2\downarrow) \\ &= (\diamond\!\!\!\!- \, g_1 \wedge g_2 \vee g_1 \wedge \diamond\!\!\!\!- \, g_2, \diamond\!\!\!\!- \, g_1 \wedge g_2 \downarrow \vee g_1 \downarrow \wedge \diamond\!\!\!\!- \, g_2) \\ &= (\diamond\!\!\!\!- \, g_1 \wedge g_2, \diamond\!\!\!\!- \, g_1 \wedge g_2 \downarrow) \, or \, (g_1 \wedge \diamond\!\!\!\!- \, g_2, g_1 \downarrow \wedge \diamond\!\!\!\!- \, g_2) \\ &= (\diamond\!\!\!\!- \, g_1 \wedge g_2, (\diamond\!\!\!\!- \, g_1 \wedge g_2)\downarrow) \, or \, (g_1 \wedge \diamond\!\!\!\!- \, g_2, (g_1 \wedge \diamond\!\!\!\!- \, g_2)\downarrow) \\ &= (\diamond\!\!\!\!- \, g_1 \wedge g_2)^\delta \, or \, (g_1 \wedge \diamond\!\!\!\!- \, g_2)^\delta \end{aligned}$$

- For *min* , $m(g) = \diamond g$.

$$\begin{aligned} g_1^\delta \, min \, g_2^\delta &= (g_1, g_1\downarrow) \, min \, (g_2, g_2\downarrow) \\ &= (\diamond g_1 \wedge g_2 \vee g_1 \wedge \diamond g_2, \diamond g_1 \wedge g_2 \downarrow \vee g_1 \downarrow \wedge \diamond g_2) \end{aligned}$$

  From $(\diamond g_1)\downarrow \Rightarrow g_1\downarrow$ and $g_2 \Rightarrow \diamond g_2$, it follows that $(\diamond g_1)\downarrow \wedge g_2 \Rightarrow g_1\downarrow \wedge \diamond g_2$ and symmetrically $(\diamond g_2)\downarrow \wedge g_1 \Rightarrow g_2\downarrow \wedge \diamond g_1$. The previous equality can be rewritten :

$$\begin{aligned} g_1^\delta \, min \, g_2^\delta &= (\diamond g_1 \wedge g_2 \vee g_1 \wedge \diamond g_2, \\ &\quad \diamond g_1 \wedge g_2 \downarrow \vee g_1 \downarrow \wedge \diamond g_2 \vee (\diamond g_1)\downarrow \wedge g_2 \vee g_1 \wedge (\diamond g_2)\downarrow) \\ &= (\diamond g_1 \wedge g_2, \diamond g_1 \wedge g_2 \downarrow \vee (\diamond g_1)\downarrow \wedge g_2) \\ &\quad or \, (g_1 \wedge \diamond g_2, g_1 \downarrow \wedge \diamond g_2 \vee g_1 \wedge (\diamond g_2)\downarrow) \\ &= (\diamond g_1 \wedge g_2)^\delta \, or \, (g_1 \wedge \diamond g_2)^\delta \end{aligned}$$

□

Using typed timed actions, drastically simplifies the general model. Furthermore, the most commonly used type, in practice, is delayable.

6 Examples

We provide two examples illustrating the use of priority choice and synchronization modes to compositionally specify systems. The first example shows how priorities can be used to achieve mutual exclusion. The second illustrates the compositional description of a traffic light controller for tramways crossing by using min and max synchronizations.

6.1 Mutual exclusion

Consider a family of cyclic processes sharing in mutual exclusion a common resource. The i-th process has period T_i and goes successively through three states w_i (wait), e_i (execute), s_i (sleep). We suppose that execution e_i takes E_i time units. A process is represented as a timed system with actions a_i (awake), p_i (proceed), r_i (release). Two clocks t_i and x_i are used respectively to enforce the period and the execution time. In figure 5 we represent two such processes. The constant D_i is taken $D_i = T_i - E_i$.

We want to construct a scheduler guaranteeing mutual exclusion for execution. A classical solution consists in restricting the behavior of the processes by a semaphore with two actions p and r by taking $p_i|p \neq \perp$, $r_i|r \neq \perp$ and $\mu(p_i) = \mu(p) = \mu(r_i) = \mu(r) = and$.

An equivalent solution can be obtained by simply assuming priorities between actions. Consider that $p_i \prec_\infty r_j$ for any pair $(i,j), i \neq j$ and take the interleaving product of the processes. It can be shown that if mutual exclusion is respected in the initial state, then it is preserved forever. Consider for instance, the interleaving product of the processes 1 and 2 under this priority restriction shown in figure 6. It is easy to check that due to priorities, the action p_1 and p_2 will never be enabled from states e_1w_2 and w_1e_2, respectively. Their guards will be restricted to states for which $\Box\neg(x_1 = E_1) = x_1 > E_1$ and $\Box\neg(x_2 = E_2) = x_2 > E_2$ hold respectively. It is trivial to verify that for correctly initialized processes $x_1 \leq E_i$ at states w_i, which implies that transitions leading to states violating mutual exclusion will never be taken.

6.2 Traffic light for tramway crossing

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 7a).

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 7b 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.

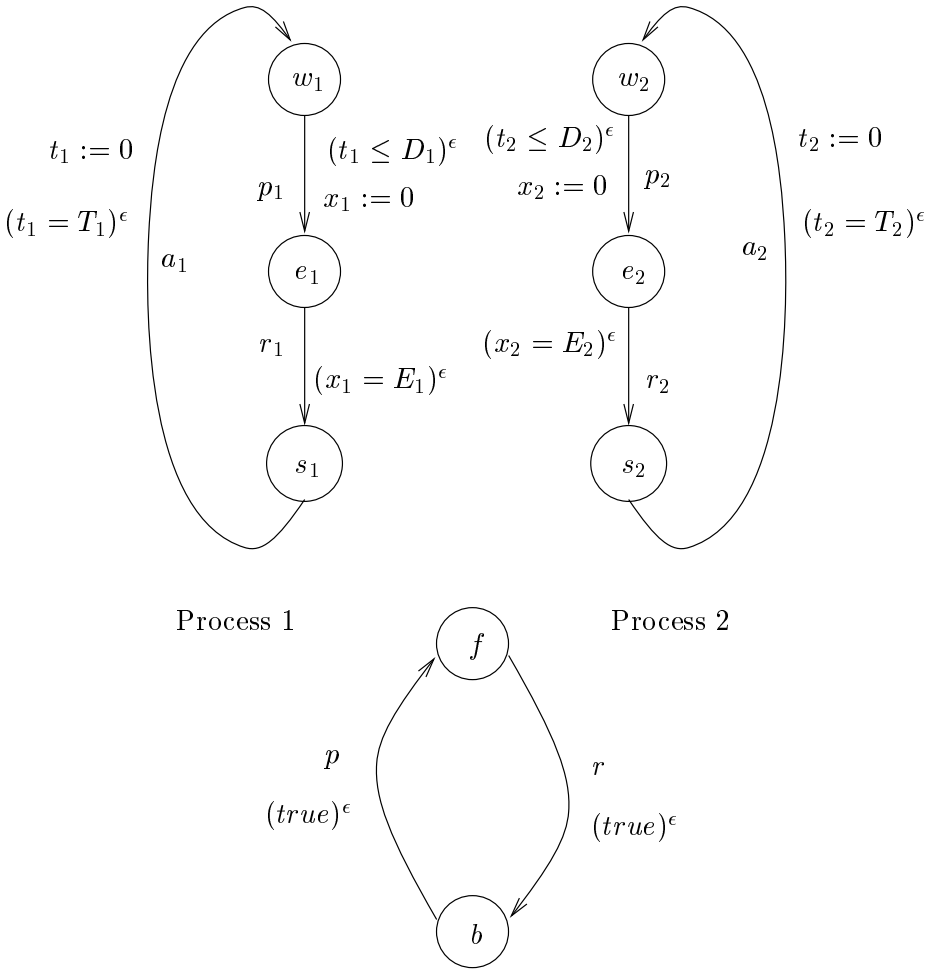


Fig. 5. Mutual exclusion for two processes

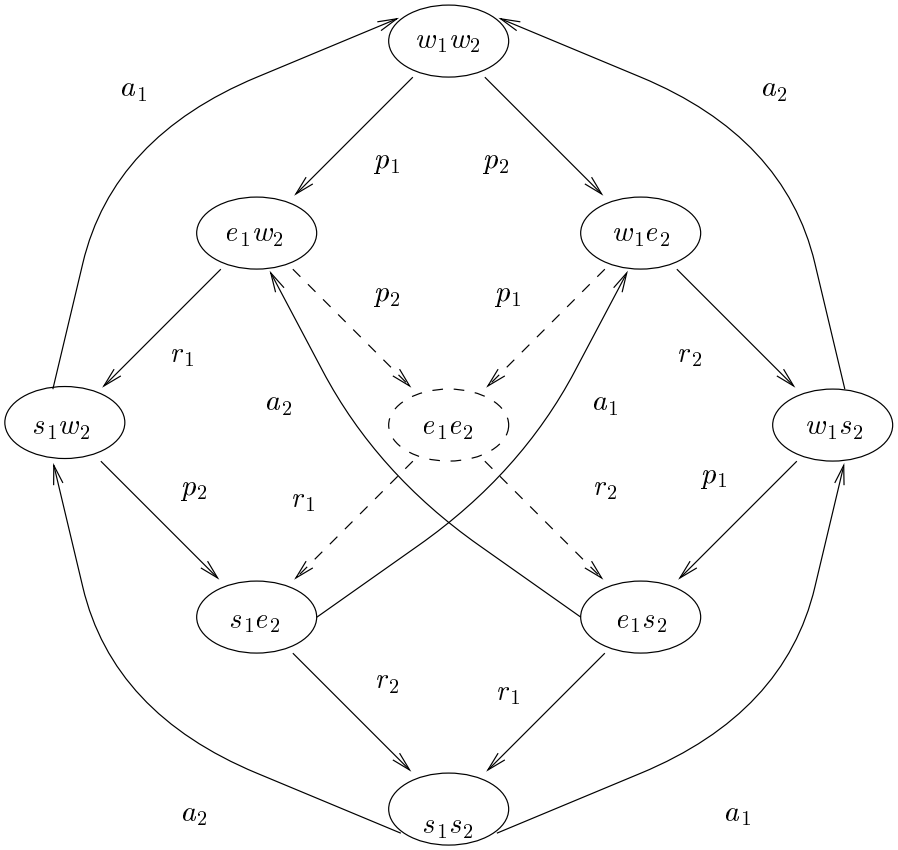
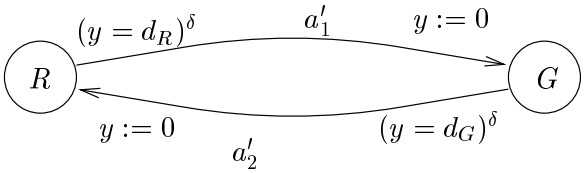
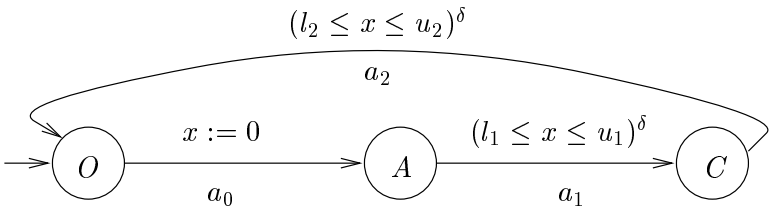


Fig. 6. Product of process 1 and 2



a. Traffic light



b. A Tramway

Fig. 7. Traffic light and Tramway

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 8. 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:

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

7 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 which is based on strong synchronization for time progress and implies preservation of components urgency. Preserving time reactivity requires sometimes relaxing urgency constraints.

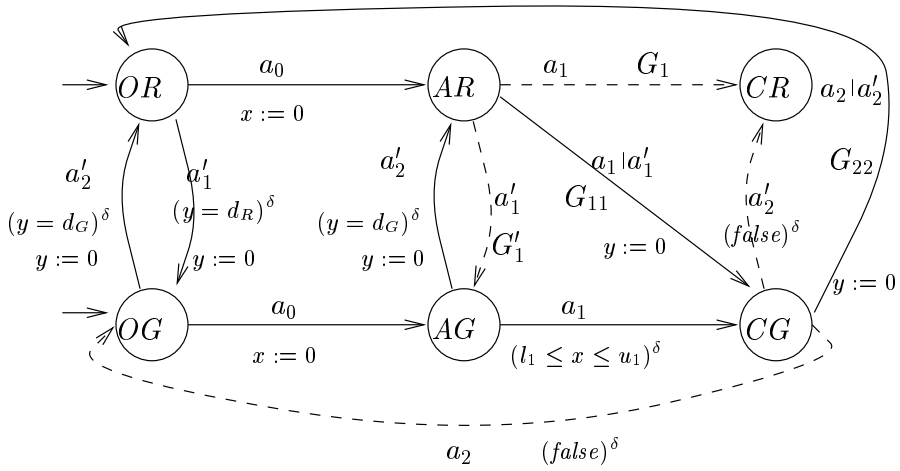


Fig. 8. Controller for a tramway

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 and in particular to scheduling.

8 Related Work

The problem of compositional description in languages with priorities has been principally studied for process algebras. The first work is, to our knowledge [BBK86], where is defined an untimed process algebra with a priority order on its set of actions. Later, in several papers, Cleaveland and his colleagues show

the interest of priority for the specification and the verification of distributed untimed systems [CH90,CLNS96,CLN96,CLN98]. Our work is closer to the work by Insup Lee and his colleagues, [BGL97,BACC+98] on the timed process algebra ACSR. The latter is a timed algebra with priorities and mutual exclusion constraints with value passing communication and dynamic priorities. It has been used for schedulability analysis of real-time systems. However, this work does not tackle compositionality issues concerning both the associativity of priority choice operators and property preservation. Another important difference is that although our priority order is static, it allows anticipation which is essential for achieving maximal progress for timed systems.

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.

[BACC+98] H. Ben-Abdallah, J.-Y. Choi, D. Clarke, Y.S. Kim, I. Lee, and H.-L. Xie. A process algebraic approach to the schedulability analisys of real-time systems. *Real-time Sytems, 15*, pages 189–219, 1998.

[BBK86] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae IX (2)*, pages 127–168, 1986.

[BGL97] P. Bremond-Gregoire and I. Lee. A process algebra of communicating shared resources with dense time and priorities. *Theoretical Computer Science, 189*, 1997.

[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. In *First International Workshop Hybrid Systems : Computation and Control HSCC'98*, pages 49–63, Berkeley, March 1998. Lecture Notes in Computer Science 1386, Spinger-Verlag.

[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.

[CH90] R. Cleaveland and M. Hennessy. Priorities in process algebra. *Information and Computation, 87(1/2)*, pages 58–77, 1990.

[CLN96] R. Cleaveland, G. Luttgen, and V. Natarajan. A process algebra with distributed priorities. In U. Montanari and V. Sassone, editors, *CONCUR '96*, pages 34–49. LNCS 1119, Springer-Verlag, August 1996.

[CLN98] R. Cleaveland, G. Luttgen, and V. Natarajan. A process algebra with distributed priorities. *Theoretical Computer Science, 195(2)*, pages 227–258, March 1998.

[CLNS96] R. Cleaveland, G. Luttgen, V. Natarajan, and S. Sims. Modeling and verifying distributed systems using priorities: A case study. *Software Concepts and Tools 17*, pages 50–62, 1996.

[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.

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

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

[Par81] D. Park. Concurrency and automata on infinite sequences. In *5th GI Conference*, Berlin, 1981. LNCS 104, Springer.

[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.