On the Construction of Live Timed Systems

Sébastien Bornot, Gregor Gößler, and Joseph Sifakis

VERIMAG, 2 rue Vignate, 38610 Gières, France

Abstract. We present a method that allows to guarantee liveness by construction of a class of timed systems. The method is based on the use of a set of structural properties which can be checked locally at low cost. We provide sufficient conditions for liveness preservation by parallel composition and priority choice operators. The latter allow to restrict a system's behavior according to a given priority order on its actions.
We present several examples illustrating the use of the results, in particular for the construction of live controllers.

1 Introduction

Building systems which satisfy given specifications is a central problem in systems engineering. Standard engineering practice consists in decomposing the system to be designed into a set of cooperating components or processes. A key problem is the coordination of the components so that the global behavior satisfies given specifications. Usually, ad hoc design methodologies are used leading to solutions that must be validated by verification and testing. In some cases, it is possible to solve the coordination problem by synthesizing a controller or supervisor that restricts the behavior of the components [3, 1]. Both validation and synthesis techniques have well-known limitations due to their inherent complexity or undecidability, and cannot be applied to complex systems. As an alternative to cope with complexity, compositional description techniques have been studied. However, the results obtained so far for reactive systems are in general difficult to exploit. They boil down either to heuristics of limited application or to general methods formulated as systems of rules with undecidable premises.

Timed systems are models of real-time systems consisting of a discrete control structure (automaton) extended with clocks, variables measuring the time elapsed since their initialization. At semantic level, they can be considered as transition systems that can perform either discrete timeless actions or time steps of some real-valued duration. For a timed system to model a real-time system, it is necessary that it is *timelock-free* that is, in any maximal run time diverges. Another essential property for timed systems modeling real-time applications such as controllers, schedulers, etc. is that any maximal run contains infinitely many actions. We call this property *livelock-freedom* as it implies deadlock-freedom and excludes indefinite waiting.

We call *live* a timed system which is both timelock-free and livelock-free. We propose a method for building live systems as the composition of live components by using parallel composition and priorities.

The method is based on a key idea that motivated several papers on the compositional description of timed systems [6–8]. It consists in enforcing the satisfaction of properties by appropriate structural restrictions preserved by composition operations. This leads to consider structural properties, intrinsic properties of the system which can be checked locally at low cost. We define a structural property called *structural liveness* which implies liveness and can be easily checked on components as the conjunction of three more elementary structural properties. We combine two kinds of constructs to build structurally live systems from structurally live components.

– Parallel composition operators defined in [6–8]. We provide sufficient structural liveness preservation conditions for and-synchronization.
– Priorities allowing to restrict the behavior of a timed system according to a given order relation on its actions. We consider timed systems with priority orders defined in [6–8] and show that priority orders preserve structural liveness. This is a basic result used to build live timed systems, as priority orders play a central role in our approach. They are used to achieve coordination in a system by appropriately restricting the behavior of its components. As an illustration of this idea, we show how priority orders can be used to specify mutual exclusion constraints by preserving structural liveness.

The use of the results for the design of live real-time controllers is illustrated by several examples.

The paper is organized as follows. Section 2 presents the properties of liveness and structural liveness, as well as sufficient conditions for guaranteeing this property. Section 3 presents priority orders, their properties and results about structural liveness preservation when priorities are applied. Section 4 presents compositionality results for systems of communicating processes. Section 5 presents a method for the compositional description of mutual exclusion properties by using priorities.

2 Timed Systems and Their Properties

2.1 Background

Let X be a set of real-valued variables called clocks. Clocks will be used as state variables measuring time progress. Their valuations will be denoted by the letter v. $true$ (resp. $false$) denotes the predicate that is true (resp. false) for any valuation v. For any non-negative real t, we represent by $v+t$ the valuation obtained from v by increasing by t the values of all the clocks.

Definition 1 (Left- and right-closure). *A predicate p is called left-closed if*

$$\forall v \ . \ \neg p(v) \Rightarrow \exists \epsilon > 0 \ . \ \forall \epsilon' \leq \epsilon \ . \ \neg p(v + \epsilon')$$

It is called right-closed if it satisfies the previous expression where $p(v+\epsilon')$ is replaced by $p(v-\epsilon')$.

Notice that these two definitions correspond to the usual notions if we consider p as a function of time, where v is a clock valuation.

Definition 2 (Rising and falling edge). *Given a predicate p on clocks X, we define the rising edge of p, noted $p\uparrow$ by:*

$$\begin{aligned} p\uparrow(v) = p(v) \wedge \exists \epsilon > 0 \ . \ \forall \epsilon' \in (0, \epsilon] \ . \ \neg p(v - \epsilon') \vee \\ \neg p(v) \wedge \exists \epsilon > 0 \ . \ \forall \epsilon' \in (0, \epsilon] \ . \ p(v + \epsilon') \end{aligned}$$

The falling edge of p, noted $p\downarrow$, is defined by the same formula where $v - \epsilon'$ and $v + \epsilon'$ are exchanged.

Definition 3 (Modal operators). *Given a predicate p on real-valued variables X, we define the modal operator $\Diamond_k \ p$ ("eventually p within k") for $k \in \mathbf{R}_+ \cup \{\infty\}$.*

$$\Diamond_k \ p \ (v) \ \mathit{if} \ \exists t \in \mathbf{R}_+ \ 0 \leq t \leq k. \ p(v + t)$$

We write $\Diamond p$ for $\Diamond_\infty \ p$ and $\Box p$ for $\neg\Diamond\neg p$.

Notice that the operators $\Diamond_k$ are just a notation for existential quantifications over time and should not be confused with temporal logic operators. Expressions with modal or edge operators can be reduced to predicates on X whenever quantification over time can be eliminated e.g., when the operators are applied to linear constraints on X.

2.2 Timed Systems

Definition 4 (Timed systems). *A Timed System is:*

- *An untimed labeled transition system $(S, \rightarrow, A)$ where*
  - *S is a finite set of* control states
  - *A is a finite vocabulary of* actions
  - *$\rightarrow \subseteq S \times A \times S$ is an untimed transition relation*
- *A finite set X of clocks, real-valued variables defined on the set of non negative reals $\mathbf{R}_+$. The set of the valuations of X, isomorphic to $\mathbf{R}_+^n$ for some n, is denoted V.*
- *A labeling function h mapping untimed transitions of $\rightarrow$ into* timed transitions*: $h(s, a, s') = (s, (a, g, d, f), s')$, where*
  - *g and d are predicates on X called respectively the* guard *and the* deadline *of the transition. We require that $d \Rightarrow g$.*
  - *$f : V \rightarrow V$ is a* jump.

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

Definition 5 (Semantics of timed systems). *A* state *of a timed system is a pair (s, v), where $s \in S$ is a control 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 are* discrete transitions *while transitions labeled by non-negative reals are* time steps.

Given $s \in S$, if $\{(s, a_i, s_i)\}_{i \in I}$ is the set of all the untimed 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* $c_s = \neg \bigvee_{i \in I} d_i$.

For the state s, we denote by $guard_s$ *and* $deadline_s$ *respectively the predicates* $\bigvee_{i \in I} g_i$ *and* $\bigvee_{i \in I} d_i$.

Notice that for time steps we have the following *time additivity* property. If for some $t_1, t_2 \in \mathbf{R}_+$ and some state (s, v), $(s, v) \xrightarrow{t_1+t_2} (s, v + (t_1 + t_2))$ then $(s, v) \xrightarrow{t_1} (s, v + t_1) \xrightarrow{t_2} (s, v + (t_1 + t_2))$, and conversely. Due to this property any sequence of time steps can be reduced into a time step of cumulated duration. If from some state (s, v) indefinite waiting is allowed, we write $(s, v) \xrightarrow{\infty} (s, \infty)$.

Timed systems are a variant of TAD [6] with an additional relaxation of usual syntactical restrictions ensuring decidability. 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 within 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 the action). The differences between these actions are illustrated in fig. 1.

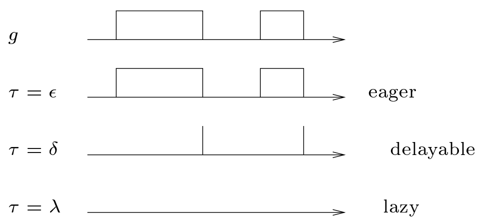

Fig. 1. Types for guards

It has been shown in [7] that any timed system can be described by a bisimilar system with only eager and lazy timed actions. In practice, we use the notation g^ϵ, g^δ and g^λ to denote a guard g of an action that is respectively eager ($d = g$), delayable ($d = g\!\downarrow$) and lazy ($d = false$).

The containment of deadlines in guards ($d \Rightarrow g$) is necessary for avoiding timelocks as it will be explained later.

Example 1. Consider the timed system of fig. 2 representing a process of period T and execution time E. The process has three control states s (sleep), w (wait) and e (execute). The clocks t and x are used to impose the period $T > 0$ and the execution time $E \leq T$, respectively. The guards $(t = T)$, $(t \leq T - E)$ and

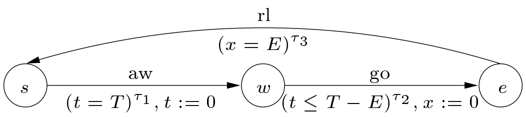

Fig. 2. A simple process.

$(x = E)$ specify when discrete transitions can occur. According to the type of urgency for the actions (denoted by τ_1, τ_2, and τ_3 in the figure), the waiting times at control states may change. For instance, if all guards are lazy then it is possible for the system to remain stuck forever at one of the states s, w, or e. When all guards are eager, discrete transitions are taken as soon as they are enabled, which means in particular that the action *go* is always executed when $t = 0$ (no waiting allowed at w). On the contrary, when *go* is delayable, this action is possible at any time t, $0 \leq t \leq T - E$. Finally, notice that the behavior remains unchanged if eager punctual guards such as $x = E$ are considered as delayable.

Definition 6 (Initial clock valuations). *Let s be a control state of a timed system and $\{(s_i, b_i, s)\}_{i \in I}$ with $b_i = (a_i, g_i, d_i, f_i)$ the non-empty set of the ingoing timed transitions. The set of the* initial clock valuations *at s is defined as the predicate*

$$in_s = \bigvee_{i \in I} post(b_i)$$

where $post(b_i)$ is the most liberal post-condition of the i-th transition defined by

$$post(b_i)(v) = \exists v',\ \ g_i(v') \wedge v = f_i(v')$$

When $I = \emptyset$, we take in_s to correspond to the valuation where all the clocks are set to zero.

Notice that in most practical cases where guards are linear constraints and jumps are linear functions, the quantifier in the expression of $post(b_i)$ can be eliminated. For the process of fig. 2, $in_s = (x = E)$, $in_w = (t = 0)$ and $in_e = (0 \leq t \leq T - E) \wedge (x = 0)$.

Definition 7 (Run). *A run of a timed system is a maximal sequence of alternating time steps and discrete transitions starting from (s_0, v_0) such that $in_{s_0}(v_0)$.*

$$(s_0, v_0) \xrightarrow{t_0} (s_0, v_0 + t_0) \xrightarrow{a_1} (s_1, v_1) \xrightarrow{t_1} \ldots (s_i, v_i) \xrightarrow{t_i} (s_i, v_i + t_i) \xrightarrow{a_{i+1}} (s_{i+1}, v_{i+1}) \ldots$$

Notice that due to time additivity of the transition system, any execution sequence of the model can be represented as a run (where some time steps may be of duration zero).

2.3 Structurally Live Timed Systems

In this section, we study three basic structural properties of timed systems.

Definition 8 (Timelock-freedom). *A timed system is timelock-free if in any run* $(s_0, v_0) \xrightarrow{t_0} (s_0, v_0 + t_0) \xrightarrow{a_1} \ldots \xrightarrow{t_i} (s_i, v_i + t_i) \xrightarrow{a_{i+1}} \ldots, \quad \sum_i t_i$ *diverges.*

Definition 9 (Livelock-freedom). *A timed system is livelock-free if in any run some action occurs infinitely often.*

Definition 10 (Liveness). *A timed system is called live if it is both timelock-free and livelock-free.*

Definition 11 (Structural non-Zenoness). *A timed system is structurally non-Zeno if in any circuit of the discrete transition graph at least one clock is reset, and it is tested against some positive lower bound.*

Structural non-Zenoness implies that there is a positive lower bound to the execution time of any circuit, and is the same as strong non-Zenoness in [16]. The periodic process of fig. 2 is structurally non-Zeno since $T > 0$.

Definition 12 (Local timelock-freedom). *A timed system is called locally timelock-free if for any state* s*, for any timed transition* (a, g, d, f) *exiting from* s, $d\!\uparrow\, \Rightarrow guard_s$.

Notice that timed systems with left-closed deadlines are locally timelock-free, as $d\!\uparrow\, \Rightarrow d \Rightarrow g$. Consequently, the timed system of fig. 2 is timelock-free independently of the type of urgency of the actions.

Lemma 1. *At any state* (s, v) *in a locally timelock-free timed system, time can progress or some action is enabled. (Proof omitted.)*

Lemma 2. *Any structurally non-Zeno and locally timelock-free timed system is timelock-free. (Proof omitted.)*

Definition 13 (Local livelock-freedom). *A timed system is called locally livelock-free if for any state* s, $in_s \Rightarrow \Diamond deadline_s$*, i.e., from any initial state some transition will be eventually taken.*

Proposition 1. *Any locally timelock-free and locally livelock-free timed system is livelock-free. (Proof omitted.)*

Definition 14 (Structural liveness). *A timed system is structurally live if it is locally timelock-free, locally livelock-free, and structurally non-Zeno.*

Clearly, structural liveness is a particular case of liveness that can be characterized by means of three properties easy to check.

Example 2. The process of fig. 2 is not locally livelock-free if one of the actions is lazy. Furthermore, even when the actions are delayable or eager, the requirement for local livelock-freedom fails for state s, since $in_s = (x = E)$, which does not imply $\Diamond(t = T) = (t \leq T)$. However, if the guard of rl is strengthened to $(x = E \wedge t \leq T)$, the behavior is not modified, and the system is locally livelock-free. So, the system is structurally live for $\tau_i \in \{\delta, \epsilon\}, i = 1, 2, 3$.

3 Timed Systems with Priorities

3.1 Motivation

In system specification, it is often convenient to consider that some priority is applied when from a state several actions are enabled. This amounts to restricting the guards of the action of lower priority to leave precedence to actions of higher priority.

Consider for example, two timed transitions $(s,(a_i,g_i,d_i,f_i),s_i)$, $i=1,2$ with common source state s. If a_1 has lower priority than a_2, then the transition labeled by a_1 becomes $(s,(a_1,g_1',d_1',f_1),s_1)$ with $g_1' \Rightarrow g_1$ and $d_1' \Rightarrow d_1$ while the other remains unchanged. Commonly, g_1' is taken equal to $g_1 \wedge \neg g_2$ which means that when a_1 and a_2 are simultaneously enabled in the system without priorities only a_2 is enabled in the prioritized system. However, for timed systems it is possible to define priority relations leaving precedence to an action if it is known that it will be enabled within some finite time.

Coming back to the previous example, we can take $g_1' = g_1 \wedge \neg\Diamond_k g_2$ for some finite k, or even $g_1' = g_1 \wedge \neg\Diamond g_2$. 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 if a_2 is disabled forever.

This motivates a notion of priority within a given delay. As an example, consider that $g_1 = 0 \leq x \leq 3 \vee 5 \leq x \leq 8$ and $g_2 = 2 \leq x \leq 7$ for some clock x. We get the following decreasing values for g_1' as the priority delay increases.

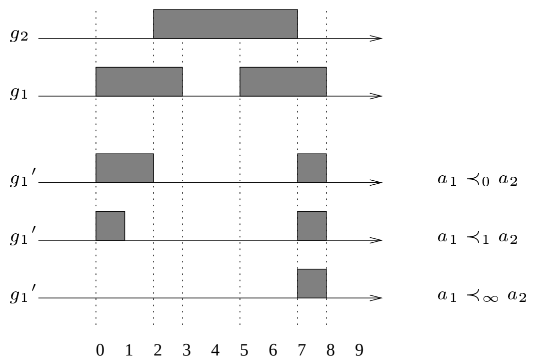

Fig. 3. Different priorities for a_2 over a_1

$g_1' = g_1 \wedge \neg g_2 = 0 \leq x < 2 \vee 7 < x \leq 8$ (immediate priority)
$g_1' = g_1 \wedge \neg\Diamond_1 g_2 = 0 \leq x < 1 \vee 7 < x \leq 8$ (priority within a delay of 1)
$g_1' = g_1 \wedge \neg\Diamond g_2 = 7 < x \leq 8$ (priority within an unbounded delay).
Fig. 3 illustrates the above example.

The following definition of priority order has been introduced and studied in [7].

Definition 15 (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* $\forall k \in \mathbf{R}_+ \cup \{\infty\}$*:*

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

Property: The relation $a_1 \prec\!\!\prec a_2 = \exists k\ a_1 \prec_k a_2$ is a partial order relation.

Definition 16 (Timed system with priorities). *A timed system with priorities is a timed system* $TS = (S, \rightarrow, A, X, h)$ *having all its guards and deadlines left- and right-closed, equipped with a* priority function pr*. The priority function* pr *associates with each state* $s \in S$ *a priority order* $pr(s)$ *such that if* $\{(a_i, g_i, d_i, f_i)\}_{i \in I}$ *is the set of the timed actions labeling transitions issued from* s*, then* $\Diamond g_i \Rightarrow \Diamond d_i$ *for any* a_i *which is not a minimal element of* $pr(s)$*.*

A timed system with priorities (TS, pr) *represents the timed system* $TS' = (S, \rightarrow, A, X, h')$ *with the same discrete transition structure and such that if* $h(s_1, a, s_2) = (s_1, (a, g, d, f), s_2)$ *then* $h'(s_1, a, s_2) = (s_1, (a, g', d', f), s_2)$ *where* g' *is defined in the following manner.*

For a given state s*, let* $\prec$ *denote the priority order* $pr(s)$*, and* $\{(a_i, g_i, d_i, f_i)\}_{i \in I}$ *be the set of the timed actions labeling transitions of* TS *exiting from* s*. The corresponding set of* prioritized timed actions *in* TS' *is then* $\{(a_i, g'_i, d'_i, f_i\}_{i \in I}$ *defined by*

$$g'_i = g_i \wedge \bigwedge_{\substack{a_j \in I,\ k \in \mathbf{R}_+ \cup \{\infty\} \\ a_i \prec_k a_j}} \neg \Diamond_k g_j \qquad\qquad d'_i = d_i \wedge g'_i$$

This definition simply says that the guard g'_i *of a prioritized action* a_i *is not enabled if there is some action* a_j *such that* $a_i \prec_k a_j$ *that will become enabled within* k *time units.*

The requirement $\Diamond g \Rightarrow \Diamond d$ for non-minimal actions means that they cannot be disabled forever without becoming urgent. It is necessary to avoid overriding of deadlines to preserve local livelock-freedom. In fact, the restriction of guards (and deadlines) of low priority would violate local livelock-freedom if actions of higher priority were lazy. Notice that for typed timed actions, it is sufficient to consider priority orders where non-minimal elements are either eager or delayable actions.

We call TS' the *prioritized* timed system corresponding to (TS, pr). We denote by $guard'_s$ and $deadline'_s$ the restrictions of $guard_s$ and $deadline_s$ in TS'.

3.2 Preservation of the Structural Properties

Theorem 1. *If* TS *satisfies one of the structural properties local timelock-freedom, local livelock-freedom, or structural non-Zenoness, then* (TS, pr) *satisfies the same property. Thus, priority orders preserve structural liveness.*

Proof. – Local timelock-freedom: priority orders transform a left-closed guard g either into left-closed guards g' or into a guard g' such that another guard g_1 of higher priority is true at the rising edge of g' (see definition of timed system with priorities). Thus $d\!\uparrow\; \Rightarrow g \vee g_1$. By induction on the priority order it is then possible to show that local timelock-freedom is preserved.

– Local livelock-freedom: priority orders do not change the discrete transition structure of timed systems, and restrict guards of timed actions. Consequently, for each state s, the set of transitions entering s does not change, and in_s is restricted. If we note in'_s the set of initial clock values of s in the prioritized system, and as the non prioritized system is locally livelock-free, we have $in'_s \Rightarrow in_s \Rightarrow \Diamond deadline_s$.
  If a deadline d of an action a is restricted to a deadline $d' \neq d$, then it is restricted by some transition (a_1, g_1, d_1, f_1) such that $a \prec_k a_1$, for some $k \in \mathbf{R}_+ \cup \{\infty\}$. This implies $d \wedge \neg d' \Rightarrow \Diamond_k g_1 \Rightarrow \Diamond g_1$. Since a_1 is not minimal, $\Diamond g_1 = \Diamond d_1$. Thus $d \Rightarrow d' \vee \Diamond d_1$. It follows that $\Diamond deadline_s = \Diamond deadline'_s$. Thus local livelock-freedom is preserved.

– Strong non-Zenoness: priority orders do not change the discrete transition structure of timed systems, and do not affect jumps. So any circuit in the prioritized system is a circuit in the initial one, and the clock resets are the same. Moreover, guards are restricted by priority orders, so a lower bound in a guard may only increase. Consequently, if the non prioritized system is structurally non-Zeno, then the prioritized one is structurally non-Zeno, too. ∎

It is often desirable to restrict a timed system TS with respect to several priority functions pr_i. At this end, we define a partial operation on priority orders.

Given $\prec^1$, $\prec^2$ priority orders on A, we represent by $\prec^1 \oplus \prec^2$ the least priority order, if it exists, that contains both $\prec^1$ and $\prec^2$, i.e.,

– $\prec^1 \cup \prec^2 \subseteq \prec^1 \oplus \prec^2$
– if $(a_1, k_1, a_2), (a_2, k_2, a_3) \in \prec^1 \oplus \prec^2$, then $(a_1, k_1 + k_2, a_3) \in \prec^1 \oplus \prec^2$.

The partially defined operator $\oplus$ is associative and commutative. We extend $\oplus$ on priority functions pr_i: $\forall s \in S.(pr_1 \oplus pr_2)(s) = pr_1(s) \oplus pr_2(s)$.

In order to simplify notations, we extend priority orders to sets of actions: $\mathcal{A}^1 \prec_k \mathcal{A}^2 \;:\Leftrightarrow\; \forall a_1 \in \mathcal{A}^1 \forall a_2 \in \mathcal{A}^2 . a_1 \prec_k a_2$.

In the rest of the paper, we show how to build live systems from live components.

4 Systems of Communicating Processes

We use the following general framework for the composition of timed systems studied in [6, 7] and based on the use of an associative and commutative parallel composition operator $\|$.

Consider timed systems of the form $TS_i = (S_i, A_i, \rightarrow_i, X_i, h_i)$. For sake of simplicity, we assume that they have disjoint sets of control states S_i, disjoint sets of actions A_i, and disjoint sets of clocks X_i. Furthermore, we consider an action vocabulary A, $A_i \subseteq A$, with an operator $|$ such that $(A, |)$ is a commutative semigroup with a distinguished absorbing element $\perp$. The action $a_1|a_2$ represents the action resulting from the synchronization of a_1 and a_2 (if $a_1|a_2 \neq \perp$).

Definition 17 (Parallel composition). *The parallel composition* $(TS_1, pr_1)\|$ (TS_2, pr_s) *of two timed systems with priorities* (TS_1, pr_1) *and* (TS_2, pr_2) *is the timed system with priorities* (TS, pr) *defined by*

- $TS = (S_1 \times S_2, A, \rightarrow, X_1 \cup X_2, h)$ *where if* $s_i \overset{a_i}{\rightarrow}_i s'_i$ *and* $h_i(s_i, a_i, s'_i) = (s_i, b_i, s'_i)$ *with* $b_i = (a_i, g_i, d_i, f_i), i = 1, 2$*, then*
  - $(s_1, s_2) \overset{a_1}{\rightarrow} (s'_1, s_2)$, $(s_1, s_2) \overset{a_2}{\rightarrow} (s_1, s'_2)$, *and* $(s_1, s_2) \overset{a_1|a_2}{\rightarrow} (s'_1, s'_2)$ *if* $a_1|a_2 \neq \perp$. *That is, the transitions of* $\rightarrow$ *are obtained by interleaving or by synchronization.*
  - *If* $a_1|a_2 \neq \perp$*, then* $\Diamond g_i = \Diamond d_i, i = 1, 2$.
  - $h((s_1, s_2), a_1, (s'_1, s_2)) = ((s_1, s_2), b_1, (s'_1, s_2))$
    $h((s_1, s_2), a_2, (s_1, s'_2)) = ((s_1, s_2), b_2, (s_1, s'_2))$
    $h((s_1, s_2), a_1|a_2, (s'_1, s'_2)) = ((s_1, s_2), b_1|b_2, (s'_1, s'_2))$
    where $b_1|b_2$ *is an extension of* $|$ *on timed actions, defined later.*
- $pr = pr_1 \oplus pr_2 \oplus pr_{12}$*, where* $\forall (s_1, s_2) \in S_1 \times S_2$.
  $pr_{12}(s_1, s_2) = \{\{a_1, a_2\} \prec_\infty a_1|a_2 \mid a_1|a_2 \neq \perp \wedge \exists s'_1, s'_2 \,.\, s_1 \overset{a_1}{\rightarrow} s'_1 \wedge s_2 \overset{a_2}{\rightarrow} s'_2\}$
  if $pr_1 \oplus pr_2 \oplus pr_{12}$ *is defined.*

The composition principle is illustrated in fig. 4. Priorities are used to achieve

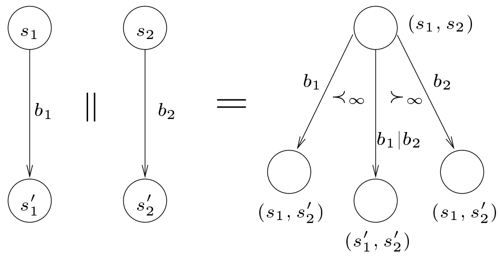

Fig. 4. Composition principle

maximal progress, that is, interleaving transitions will be taken only when synchronization is not possible.

In [6, 7] a general method is provided to extend the operator $|$ on timed actions, preserving associativity and commutativity. We consider here a particular case of timed action synchronization, called *and*-synchronization, which is defined as follows:

For $b_i = (a_i, g_i, d_i, f_i), i = 1, 2$ if $a_1|a_2 \neq \perp$, then $b_1|b_2 = (a_1|a_2, g_1 \wedge g_2, (g_1 \wedge g_2) \wedge (d_1 \vee d_2), f_1 \cup f_2)$. This simply means that synchronization takes place only when both actions are enabled. The synchronization action becomes urgent whenever it is enabled, and one of the synchronizing actions becomes urgent. Finally, $f_1 \cup f_2$ denotes the composition of jumps (for disjoint state spaces of the components).

Notice that $\|$ is an associative and commutative partial operation on timed systems with priorities. We trivially consider a timed system TS as a timed system with priorities $(TS, pr_\emptyset)$, where $pr_\emptyset$ is the priority function associating the empty priority order with any state.

Theorem 2. *If TS_1 and TS_2 are structurally live, then $TS_1 \parallel TS_2$ is structurally live.*

Proof. If $(TS, pr) = TS_1 \| TS_2$, then it is sufficient to consider TS by application of Theorem 1, as for synchronizing actions (which are not minimal in $pr(s)$ for all s), $\Diamond g_i = \Diamond d_i$. We show that if TS_1 and TS_2 are structurally live, then TS is structurally live.

- Structural non-Zenoness: since each transition (interleaving or synchronization) corresponds to a transition in TS_1 or TS_2 or both, each circuit in the product contains a set of transitions forming a circuit in TS_1 or TS_2. If TS_1 and TS_2 are structurally non-Zeno, then in all these circuits some clock is reset and tested against a positive lower bound. Then this is the case for all the circuits of TS too. The bounds of a synchronization action may increase, but can not decrease.
- Local timelock-freedom: conjunction and disjunction preserve closure of guards and deadlines, so guards and deadlines of synchronizations are closed as well as those of interleaving actions. This guarantees local time-lock freedom.
- Local livelock-freedom: since interleaving transitions are transitions of the components, and synchronization guards are conjunctions of guards of the components, if TS_1 and TS_2 are locally livelock-free we have

$$\begin{aligned} in_{(s_1,s_2)} \Rightarrow in_{s_1} \vee in_{s_2} &\Rightarrow \Diamond deadline_{s_1} \vee \Diamond deadline_{s_2} \\ &\Rightarrow \Diamond deadline_{(s_1,s_2)} \end{aligned}$$

Thus, the product system is locally livelock-free. ∎

Example 3. Consider two structurally live processes P_1 and P_2 with execution times E_i and periods T_i, $i = 1, 2$, running in parallel, as shown in fig. 5. Each process P_i can procrastinate the go_i action, which models the start of its execution, as long as enough time remains for execution before the end of the period.

In order to let the processes exchange data, however, one wishes to coordinate them by a common $go_1|go_2$ action whenever this is possible without violating the timing constraints of any of P_1 and P_2. This is done by synchronizing the go_1 and go_2 actions. Maximal progress is ensured in (w_1, w_2) by the priorities $go_1 \prec_\infty go_1|go_2$ and $go_2 \prec_\infty go_1|go_2$, and guarantees that the two processes will synchronize if they manage to be in (w_1, w_2) at the same time. The resulting

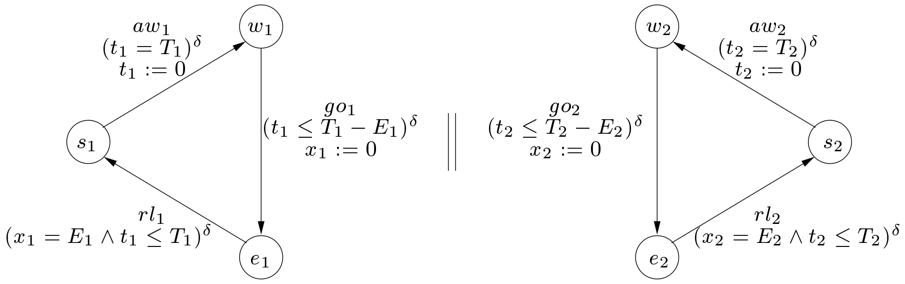

Fig. 5. Two processes with flexible synchronization.

timed actions in the product system with priorities have typed guards g_1', g_2', and g_{12}, respectively, for go_1, go_2, and $go_1|go_2$:

$$\begin{aligned}
g_1' &= ((x_1 \le T_1 - E_1) \wedge \neg\Diamond(x_2 \le T_2 - E_2))^\delta = ((x_1 \le T_1 - E_1) \wedge (x_2 > T_2 - E_2))^\delta \\
g_2' &= ((x_2 \le T_2 - E_2) \wedge \neg\Diamond(x_1 \le T_1 - E_1))^\delta = ((x_2 \le T_2 - E_2) \wedge (x_1 > T_1 - E_1))^\delta \\
g_{12} &= ((x_1 \le T_1 - E_1) \wedge (x_2 \le T_2 - E_2))^\delta.
\end{aligned}$$

The product system is structurally live.

An important property in the previous example is individual livelock-freedom of the components in the product system. Liveness of the product does not imply that in any run, an action of some component occurs infinitely often. This remark motivates the following definition and theorem.

Definition 18 (Individual livelock-freedom). *A component* (TS_i, pr_i) *is* live-lock-free *in a timed system* $(TS, pr) = (TS_1, pr_1)\|(TS_2, pr_2)$ *if in each run of* (TS, pr)*, some action of* (TS_i, pr_i) *occurs infinitely often.*

Theorem 3. *If both* (TS_1, pr_1) *and* (TS_2, pr_2) *are structurally live and each synchronizing guard* g *is bounded (that is,* $\Diamond\Box\neg g$*), then both* TS_1 *and* TS_2 *are livelock-free in* $(TS_1, pr_1)\|(TS_2, pr_2)$*. (Proof omitted.)*

Example 4. Both processes of Example 3 are livelock-free in the product system.

Remark 1. Notice that for liveness preservation, the systematic use of "escape" actions (aborted communications) in the product system is instrumental. Our parallel composition operator allows a process to proceed unsynchronized on any action for which the communication will not be ready in its current state sometime in the future. This is different from usual "strong" parallel composition where synchronization is enforced by the use of restriction operators as in CCS [15] or by allowing interleaving only for internal (non communicating) actions as in CSP [12]. Such operators enforce synchronization at the risk of deadlock, particularly in the case of timed systems.

5 Mutual Exclusion

Consider a timed system initially composed of a set of interacting components. The goal is to restrict the behavior of the components by using priorities so that the global behavior satisfies a given mutual exclusion constraint. We study a method to obtain a structurally live system satisfying mutual exclusion from structurally live components.

The following notion of persistence will be central in this section.

Definition 19 (Persistence). *A control state s is called* persistent *if $in_s \Rightarrow \Diamond\Box guard_s$.*

This property means that at state s, maybe after some waiting, it is always possible to execute an action. It is instrumental for avoiding deadlocks when guards are restricted by using priorities.

Consider a timed system $(TS, pr) = TS_1\| \ldots \|TS_n$, where $TS_i = (S_i, A_i, \rightarrow_i, X_i, h_i)$, and $TS = (S, A, \rightarrow, X, h)$, as in section 4. We suppose that a mutual exclusion constraint is specified as a set $M \subseteq \bigcup_i S_i$ containing pairwise mutually exclusive states. We define two predicates on the product space S:

- $bad_M(s_1, \ldots, s_n) \ = \ \exists i, j, i \neq j. s_i \in M \wedge s_j \in M$
- $critical_M(s_1, \ldots, s_n) \ = \ \neg bad_M(s_1, \ldots, s_m) \ \wedge$
  $\exists a \in A, (s'_1, \ldots, s'_n) \in S \ . \ (s_1, \ldots, s_n) \xrightarrow{a} (s'_1, \ldots, s'_n) \wedge bad_M(s'_1, \ldots, s'_n)$.

bad_M characterizes all the states violating the mutual exclusion constraint, whereas $critical_M$ characterizes all the legal states from which mutual exclusion can be violated by executing one transition.

For a given $M \subseteq \bigcup_i S_i$ we define $\bullet M$ and $M\circ$ as the set of actions entering M and the set of actions originating in M:

$$\bullet M = \{a \mid \exists i \ \exists s, s' \in S_i, s \notin M \wedge s' \in M \wedge s \xrightarrow{a}_i s'\}$$
$$M\circ = \{a \mid \exists i \ \exists s, s' \in S_i, s \in M \wedge s \xrightarrow{a}_i s'\}$$

The set of *waiting states* of component S_i with respect to M is the set of states from which some action entering M is issued: $\{s \in S_i - M | \exists a \in A_i, s' \in S_i \cap M \ s.t. \ s \xrightarrow{a}_i s'\}$.

Theorem 4 (Mutual exclusion). *Let $TS_1 \ldots, TS_n$ be a set of structurally live timed systems with persistent waiting states w.r.t. a mutual exclusion constraint $M \subseteq \bigcup_i S_i$, and $(TS, pr) = TS_1\| \ldots \|TS_n$ be the parallel composition of the components with $\forall a_1 \in \bullet M \ \forall a_2 \in A. a_1|a_2 = \bot$. Then, the timed system with priorities $(TS, pr \oplus pr_M)$, where*

$$\forall s \in S \ . \ pr_M(s) = \begin{cases} \bullet M \prec_\infty M\circ & \textit{if } critical_M(s) \\ \emptyset & \textit{otherwise} \end{cases} \tag{1}$$

is structurally live, and satisfies the mutual exclusion constraint.

Proof (sketch).

– Structural liveness: Let us first show that $(pr \oplus pr_M)(s)$ is a partial order for all $s \in S.\neg bad_M(s)$. Suppose that $a_1 \prec_\infty a_2 \in pr$ (i.e., a_2 is a synchronizing action), and that $a_2 \prec_\infty a_1 \in pr_M$. This is not possible, since synchronizing actions are maximal in pr_M. Hence, $pr \oplus pr_M$ is a priority function. Structural liveness follows from Theorems 2 and 1.
– Mutual exclusion: No synchronization action can lead $(TS, pr \oplus pr_M)$ from a safe state $s \in S.\neg(critical_M(s) \vee bad_M(s))$ into a bad_M state violating mutual exclusion.
Let $s = (s_1, \ldots, s_n)$ be a state of TS with $critical_M(s)$, such that there exist s_k, s_i, $s_k \in M$ and $\exists a \in A_i.s_i \xrightarrow{a} s'_i$ with $s'_i \in M$. Thus, a is an action leading (TS, pr) into a bad state. Since $a \in \bullet M$, we have $a_i \prec_\infty \{s_k\}\circ \subseteq pr_M(s)$. Let us show that from any initial clock valuation of s_k, some action issued from s_k will eventually be enabled, which means that a is disabled due to priority choice semantics.
From the livelock-freedom assumption about TS_k, $in_{s_k} \Rightarrow \Diamond deadline_{s_k}$, one can deduce that in the product (TS, pr), for any state satisfying in_{s_k} there exists an action in $A_{s_k} = \{s_k\} \circ \cup \{a_1|a_2 \mid a_1 \in \{s_k\} \circ \wedge a_2 \in A \wedge a_1|a_2 \neq \bot\}$ with deadline d such that $\Diamond d$. This property remains true as long as TS_k is in state s_k in (TS, pr). The same argument can be applied for s_k in $(TS, pr \oplus pr_M)$, as the actions of A_{s_k} are not restricted by the priority function pr_M. ■

Intuitively, equation (1) says that from critical states, M must be left before a new process can enter. The persistence of the waiting states makes sure that no process becomes deadlocked while waiting to enter M.

Remark 2. Notice that the above construction minimally restricts the untimed transition structure in the following manner: if from a state $s = (s_1, \ldots, s_n)$, $\neg bad_M(s)$ there exist transitions $s \xrightarrow{a_1} s'$ and $s \xrightarrow{a_2} s''$ with $a_1 \in \bullet M$ and $a_2 \in M\circ$, then $critical_M(s)$, and $bad_M(s')$.

Definition 20 (Individual deadlock-freedom). *A component TS_i of a timed system $(TS, pr) = TS_1 \| \ldots \| TS_n$ is* deadlock-free *in TS if for each run of TS, some action of TS_i is enabled infinitely often.*

Theorem 5. *Let $TS_1 \ldots, TS_n$ be a set of structurally live timed systems, and $M \subseteq \bigcup_i S_i$ a mutual exclusion constraint, as in Theorem 4. If for each TS_i, any run of TS_i contains infinitely many occurrences of states in $S_i - M$, the actions in $M\circ$ do not synchronize, and TS_i is livelock-free in $(TS, pr) = TS_1 \| \ldots \| TS_n$, then all components are deadlock-free in $(TS, pr \oplus pr_M)$. (Proof omitted.)*

Remark 3. Let TS_i be a structurally live timed system, and M a mutual exclusion constraint. A sufficient structural condition for the runs of TS_i to contain infinitely many occurrences of states in $S_i - M$ is that the untimed transition graph of TS_i has no cycles in M.

Theorem 6. *Let $\{M_i\}_{i \in I}$ be a set of mutual exclusion constraints on a structurally live timed system (TS, pr). If pr_{M_i} are the priority functions ensuring* *mutual exclusion according to Theorem 4, and $\pi = pr \oplus \bigoplus_{i \in I} pr_{M_i}$ is a priority function, then (TS, π) is structurally live and respects all mutual exclusion constraints M_i. (Proof omitted.)*

Example 5. Consider two processes of periods T_1, T_2 and execution times E_1, E_2 as in fig. 6. Each one of the processes is structurally live and remains livelock-

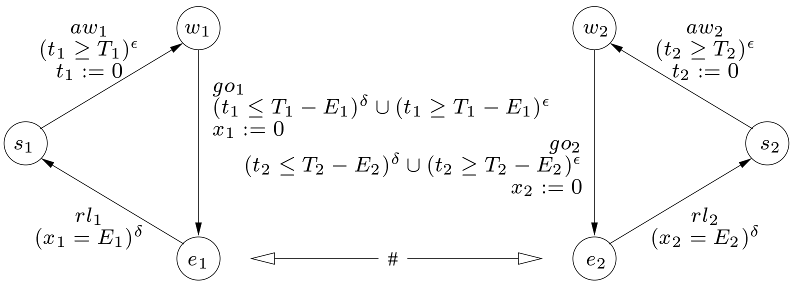

Fig. 6. Two processes with mutual exclusion between e_1 and e_2.

free in the interleaving product of the two processes, which is structurally live. We apply the mutual exclusion constraint $M = \{e_1, e_2\}$ on the interleaving product and obtain $\bullet M \prec_\infty M\circ$, i.e., $\{go_1, go_2\} \prec_\infty \{rl_1, rl_2\}$ for the critical states (w_1, e_2), (e_1, w_2). This practically means that release-actions rl have higher priority than begin-actions go. According to Theorem 4, the product system with priorities is structurally live, and both processes are deadlock-free in it, due to the persistent waiting states w_1 and w_2.

Notice that in order to compute $(TS_1 \| \dots \| TS_n, pr)$, it is not necessary to compute explicitly the product $T_1 \| \dots \| T_n$. Priority choice can be applied "on the fly" to states of the product.

The construction of Theorem 4 can be generalized for mutual exclusion constraints of the type *m-out-of-n* for $m < n$, where $critical_M(s)$ $(bad_M(s))$ denotes the set of control states where exactly m (more than m) components are in M.

Example 6 (Resource allocation). Consider the well-known example of fig. 7, where two interleaving processes P_1 and P_2 use two shared resources R_1 and R_2. P_1 allocates resource R_1, then R_2, while it still holds R_1, before releasing both resources. P_2 tries to allocate the resources in the inverse order.

An action p_{ij} means that process P_i allocates resource R_j, and a v_i-action means that process P_i frees both resources. Mutual exclusion over R_1 is modeled by the set $M_1 = \{s_2, s_3, s_7\}$, and mutual exclusion over R_2 by $M_2 = \{s_3, s_6, s_7\}$, as indicated by arrows in the figure. Critical states are those characterized by $critical_{M_1} = (s_2 \vee s_3) \wedge s_6 \vee s_1 \wedge s_7$, and $critical_{M_2} = s_3 \wedge s_5 \vee s_2 \wedge (s_6 \vee s_7)$ (where the name of a component state is used as a proposition which is true if the component system is in that state). Mutual exclusion on M_i is guaranteed for (TS, pr_{M_i}), if TS is the interleaving product of P_1 and P_2. The priority functions

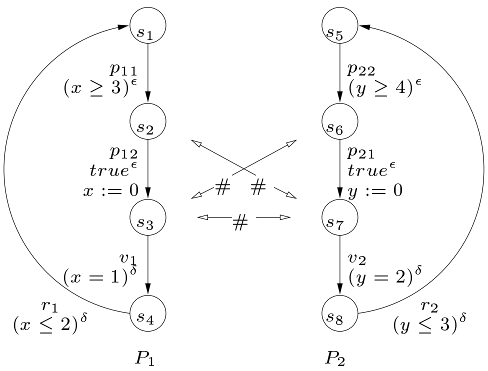

Fig. 7. Crossover resource allocations.

pr_{M_i} $(i = 1, 2)$ are defined by $pr_{M_i}(s) = \bullet M_i \prec_\infty M_i\circ$ for $critical_{M_i}(s)$, $\emptyset$ otherwise. We have $\forall s \in S$:

$$pr_{M_1}(s) = \begin{cases} \{p_{11}, p_{21}\} \prec_\infty \{p_{12}, v_1, v_2\} & \text{if } critical_{M_1}(s) \\ \emptyset & \text{otherwise} \end{cases}$$

$$pr_{M_2}(s) = \begin{cases} \{p_{12}, p_{22}\} \prec_\infty \{p_{21}, v_1, v_2\} & \text{if } critical_{M_2}(s) \\ \emptyset & \text{otherwise} \end{cases}$$

Both mutual exclusion constraints M_1 and M_2 are respected by $(TS, pr_{M_1} \oplus pr_{M_2})$, if $pr_{M_1} \oplus pr_{M_2}$ is defined. However, in state (s_2, s_6) — P_1 has allocated R_1 and waits for R_2, whereas P_2 has allocated R_2 and waits for R_1 —, one can see that priorities form a circuit with vertices p_{12} and p_{21}. This flaw in the specification (which means that the specification is intrinsically not deadlock-free) can be corrected by declaring states $\{s_2, s_6\}$ in mutual exclusion. The resulting mutual exclusion constraint is $M = \{s_2, s_3, s_6, s_7\}$, which means that the product state (s_2, s_6) is made unreachable. In practice, this means that the sequence consisting of the two resource allocations is atomic, and the obtained system is live.

Notice that each process eventually frees all resources, and the waiting states s_1 and s_5 are persistent, so that both processes remain individually deadlock-free after composition in this improved specification.

6 Discussion

We have presented a method for the construction of live timed systems based on the preservation of a set of structural properties by appropriately chosen composition operators. Structural properties verification can be done locally and does not require exploration of the global system's dynamic behavior. The set of initial states is also structurally determined.

An important question to be further investigated is applicability of the method. It concerns the expressivity of the class of structurally live systems as well as the possibility to apply in practice the liveness preservation theorems.

We believe that the two properties implying timelock-freedom correspond to completely natural common sense requirements of sanity. Structural non-Zenoness is a kind of "well-guardedness" property that is satisfied in practice. Local livelock-freedom is also a basic property to avoid problems in the interaction between time progress and discrete state changes. The main difficulty in the application of our method, is the satisfaction of the local livelock-freedom property. It may happen that the initial clock valuation at some state is too weak. In that case, the guards of the transitions entering this state must be strengthened in an appropriate manner (as in Example 2) and this is left to the user's ingenuity.

The proposed parallel composition operator can be enhanced by selective elimination of escape actions thanks to the use of additional priorities. If all non communicating actions are given infinitely higher priority than communicating actions then stronger interaction can be achieved and for this parallel composition operator Theorem 2 remains true. However, Theorem 3 does not hold and it is not easy to find conditions for individual liveness preservation.

The presented method uses a framework for the compositional description of timed systems [6–8]. This framework is based on "flexible" composition operations in the sense that composition allows only interaction resulting in a global behavior that preserves liveness. Priorities are essential for restricting appropriately the behavior of the cooperating components depending on their abilities to synchronize from their respective states. This contrasts with usual composition operations which are "constraint oriented" and consider parallel composition essentially as the intersection of "observable" behaviors. Such operations enforce action synchronization and respect components urgency at the risk of livelock or timelock. In the presence of timelock or livelock, the behavior of the components must be changed to get correct specifications. We prefer flexible composition, in spite of its deficiencies, because it is more appropriate for the "correct by construction" approach. To our knowledge, this approach has not been very much explored so far. [16] defines sufficient static conditions for deadlock- and timelock-freedom for the synchronized product of timed automata. Also, there exists some work about reactive systems such as [9, 13].

Concerning the use of priorities, there exist many papers introducing priorities to process algebras, mainly for the untimed case [5, 10, 14]. Our notion of timed systems with priorities is closer to the work of [14] on the timed process algebra ACSR. However, this work does not tackle problems of property preservation.

This work is part of a more general project which aims at developing techniques and tools for modeling and analyzing real-time systems. We implemented priority choice, parallel composition as well as verification of structural properties, in a prototype tool, which we are currently using for the description of scheduling algorithms.

References

1. K. Altisen, G. Gößler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A Frame-

work for Scheduler Synthesis. In IEEE *RTSS'99* proceedings, 1999.
2. R. Alur and D.L. Dill. A theory of timed automata. *TCS*, 126, pp. 183-235, 1994.
3. E. Asarin, O. Maler, and A. Pnueli. Symbolic Controller Synthesis for Discrete and Timed Systems. *Hybrid Systems II*, LNCS 999, Springer-Verlag, 1995.
4. P. C. Attie. Synthesis of Large Concurrent Programs via Pairwise Composition. In *CONCUR'99*.
5. 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)*, pp. 127-168, 1986.
6. S. Bornot and J. Sifakis. On the composition of hybrid systems. In *International Workshop "Hybrid Systems: Computation and Control"*, LNCS, pp. 49–63. Springer-Verlag, April 1998.
7. S. Bornot and J. Sifakis. An Algebraic Framework for Urgency. In *Calculational System Design*, NATO Science Series, Computer and Systems Science 173, Marktoberdorf, July 1998.
8. S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *COMPOS'97*, Malente, Germany. LNCS 1536, Springer-Verlag, 1998.
9. L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Latency Insensitive Protocols. In *CAV'99*, Trento, Italy. LNCS 1633, Springer-Verlag, 1999.
10. R. Cleaveland, G. Lüttgen, V. Natarajan, and S. Sims. Priorities for Modeling and Verifying Distributed Systems. In *TACAS'96*, LNCS 1055, pp. 278-297.
11. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. Proc. *7th Symp. on Logics in Computer Science (LICS'92)* and *Information and Computation* 111(2):193–244, 1994.
12. C. A. R. Hoare. Communicating Sequential Processes. Prentice Hall, 1985.
13. M. V. Iordache, J. O. Moody, and P. J. Antsaklis. A Method for Deadlock Prevention in Discrete Event Systems Using Petri Nets. Technical Report, University of Notre Dame, July 1999.
14. H. Kwak, I. Lee, A. Philippou, J. Choi, and O. Sokolsky. Symbolic schedulability analysis of real-time systems. In IEEE *RTSS'98*, Madrid, Spain, December 1998.
15. R. Milner. Communication and Concurrency. Prentice Hall, 1989.
16. S. Tripakis. Verifying Progress in Timed Systems. In *ARTS'99*, Bamberg, Germany, 1999 (to appear in LNCS series).