A Methodology for the Construction of Scheduled Systems *

K. Altisen, G. Gößler, and J. Sifakis

Abstract

We study a methodology for constructing scheduled systems by restricting successively the behavior of the processes to be scheduled. Restriction is used to guarantee the satisfaction of two types of constraints: schedulability constraints guaranteeing that timing properties of the processes are satisfied, and constraints characterizing particular scheduling algorithms including process priorities, non-idling, and preemption.

The methodology is based on a controller synthesis paradigm. The main results deal with the characterization of scheduling policies as safety constraints and the simplification of the synthesis process by applying a composability principle.

1 Introduction

Scheduling coordinates the execution of application and system activities, so as requirements about their temporal behavior are met. Guaranteeing correctness of schedulers is essential for the development of dependable real-time systems. In many application areas, well established theory and scheduling algorithms have been successfully applied to real-time systems development.

Existing scheduling theory is limited because it requires the system to fit into the mathematical framework of the schedulability criterion (e.g. all tasks are supposed periodic, worst case execution times are known). Studies to relax such hypotheses have been carried out but they generalize one hypothesis at a time, and no unified approach has been proposed.

To overcome limitations of scheduling theory, it is important to study its connections to specification theory and take advantage of their complementarity [8, 13]. The specification based approach consists in building a timed model of the scheduled system or of an abstraction of it. Then, timed analysis tools are used either to check that the exact model meets scheduling requirements or to extract from the abstraction a scheduler [6, 10].

A major difficulty in applying this approach is the generation of the timed model from some description of the scheduling method. In fact, scheduling deals with the very dynamic nature of real-time systems, and behavior modeling requires a deep understanding of mechanisms such as priorities and preemption, as well of concepts such as urgency, idling, timeliness.

In this paper we propose a methodology for modeling scheduling algorithms that constructs compositionally the scheduled system from a global timed model based on

1. A functional description of the processes to be scheduled, their resources, and the associated synchronization and management constraints;

*submitted to FTRTFT'00

2. Timing requirements added to the functional description and relating in particular execution speed with the dynamics of the external environment;

3. A description of a scheduling algorithm consisting of three types of requirements about

   - fixed or dynamic priorities, for choosing between pending requests of the processes;
   - possibility of idling, meaning that the scheduler may not satisfy a pending request anticipating the satisfaction of a forthcoming higher priority request;
   - preemption, that is, for a given preemption order between processes, a process of lower priority is preempted when a process of higher priority raises a request.

In previous papers [4, 3] we have shown how a functional description can be extended into a timed one by preserving progress properties. In this paper we study a methodology for constructing a scheduled system from scheduling requirements and a timed specification of the processes to be scheduled. The methodology is based on the controller synthesis paradigm [11, 9, 1]. A scheduler is considered as a controller of the processes to be scheduled which restricts their behavior by triggering their controllable actions. The restricted behavior must respect the timing constraints of the processes as well as constraints characterizing the scheduling requirements.

We have shown in [1] how schedulers can be computed by applying a synthesis algorithm to timed automata. The synthesis algorithm computes iteratively from a constraint K characterizing scheduling requirements, the maximal *control invariant* K', $K' \Rightarrow K$. The latter denotes the set of states from which K is guaranteed. The behavior of the scheduled system is obtained by restricting the controllable actions of the processes so as to respect the control invariant K'.

The application of synthesis techniques is limited for two reasons. First, the practical complexity of the synthesis algorithm is high even in the case of timed automata without scheduling policy constraints. Second, scheduling with preemption requires the use of automata with integrators [5] which implies that iterative computation of control invariants may not terminate.

The proposed methodology allows to decompose the global controller synthesis procedure into the application of simpler steps. At each step a control invariant corresponding to a particular class of constraints is applied to further restrict the behavior of the system of processes to be scheduled. The presented results can be summarized as follows:

1. Global scheduling requirements can be characterized by a constraint K of the form $K = K_{algo} \wedge K_{sched}$ where K_{algo} specifies a particular scheduling algorithm, and K_{sched} characterizes schedulability requirements of the processes. Furthermore, K_{algo} is a conjunction of constraints about the scheduling policy, the possibility of non-idling, and preemption;

2. A step of the method corresponds to the computation of a controller for some constraint. The control invariant corresponding to a constraint can be computed in a straightforward manner (without iterative fixpoint computation);

3. The scheduled system can be obtained by successive applications of steps restricting the process behavior by control invariants implying all the scheduling constraints, provided that some composability conditions are satisfied. In fact, the restriction by a control invariant does not necessarily preserve previously imposed control invariants.

The methodology allows an incremental construction of a scheduled system, or of an abstraction of it if some steps fail.

The paper is composed of two sections. The first section presents basic results about control invariants and their composability. The second section shows how scheduling requirements can be expressed as constraints which are control invariants in some cases. The application of the methodology is illustrated by examples.

2 Control Invariants and Composability

2.1 Timed system

To model scheduling algorithms, we use reactive timed systems with two kinds of actions as in [1]: controllable actions that can be triggered by the scheduler, and uncontrollable actions that can be considered as internal actions of the processes to be scheduled. Controllable actions are typically resource allocations and process preemption, while uncontrollable actions are process arrival and termination.

Both controllable and uncontrollable actions are submitted to timing constraints expressed in terms of real-valued variables called *timers*. The derivatives of timers may take the values 0 or 1, as specified by a boolean vector.

Definition 2.1 *(X-constraint)*
Let X be a finite set of timers, $\{x_1, \ldots, x_m\}$, real-valued variables defined on the set of non-negative reals $\mathbb{R}_+$. A predicate C generated by the grammar $C ::= x \# d \mid x - y \# d \mid C \wedge C \mid \neg C$, where x, $y \in$ X, d is an integer, and $\# \in \{\leq, <\}$, is called a *X-constraint*.

Definition 2.2 *(falling edge)*
Let C be a X-constraint, and b be a boolean derivative vector of $\{0, 1\}^m$. The *closed* (resp. *open*) *falling edge* of C w.r.t. b, written $\overset{\bullet}{\downarrow}_b C$ (resp. $\overset{\circ}{\downarrow}_b C$) is defined as $\forall x \in \mathbb{R}_+^m$:

$$\begin{aligned} \overset{\bullet}{\downarrow}_b C(x) &\Leftrightarrow C(x) \wedge \exists t > 0 \,.\, \forall t' \in (0, t] \,.\, \neg C(x + t'b) \\ \overset{\circ}{\downarrow}_b C(x) &\Leftrightarrow \neg C(x) \wedge \exists t > 0 \,.\, \forall t' \in (0, t] \,.\, C(x - t'b) \end{aligned}$$

Example 2.1 Let X $= \{x_1, x_2\}$ be the set of real valued variables. $C = x_1 \leq 3$ and $C' = 2 \leq x_1 < 6 \wedge x_1 - x_2 \leq 4$ are X-constraints. For $b = (1, 1)$ and $b' = (1, 0)$, we have:

$\overset{\bullet}{\downarrow}_b C = (x_1 = 3)$	$\overset{\circ}{\downarrow}_b C = false$	$\overset{\bullet}{\downarrow}_b C' = false$	$\overset{\circ}{\downarrow}_b C' = (x_1 = 6) \wedge (x_2 \geq 6)$
$\overset{\bullet}{\downarrow}_{b'} C = (x_1 = 3)$	$\overset{\circ}{\downarrow}_{b'} C = false$	$\overset{\bullet}{\downarrow}_{b'} C' = (x_1 - x_2 = 4) \wedge (x_2 < 2)$	$\overset{\circ}{\downarrow}_{b'} C' = (x_1 = 6) \wedge (x_2 \geq 6)$

Definition 2.3 *(timed system)*
A *timed system* is:

- an untimed labeled transition system (S, A, T) where S is a finite set of control states; A is a finite vocabulary of actions partitioned into two sets of controllable and uncontrollable actions noted A^c and A^u; T $\subseteq$ S $\times$ A $\times$ S is an untimed transition relation;
- a finite set of timers X $= \{x_1, \ldots, x_m\}$, as in definition 2.1;
- a function b mapping S into $\{0, 1\}^m$. The image of $s \in$ S by b denoted b_s is a boolean derivative vector;
- a labeling function h mapping untimed transitions of T into timed transitions: $h(s, a, s') = (s, a, g, \tau, r, s')$, where the guard g is a X-constraint; the reset $r \subseteq$ X is a set of timers to be reset; $\tau \in \{\lambda, \delta, \epsilon\}$ is an urgency type, respectively lazy, delayable, eager.

Semantics. A timed system defines a *transition graph* $(\mathcal{V}, \mathcal{E})$ constructed as follows. $\mathcal{V} = \mathrm{S} \times \mathbb{R}_+^m$, that is, vertices (s, x) are states of the timed system.

The set $\mathcal{E} \subseteq \mathcal{V} \times (\mathrm{A} \cup \mathbb{R}_+^*) \times \mathcal{V}$ of the edges of the graph is partitioned into three classes of edges: $\mathcal{E}^c$ controllable, $\mathcal{E}^u$ uncontrollable, and $\mathcal{E}^t$ timed, corresponding respectively to the case where the label is a controllable action, an uncontrollable action, and a positive real.

Given $s \in \mathrm{S}$, let J be the set of indices such that $\{(s, a_j, s_j)\}_{j \in J}$ is the set of all the untimed transitions departing from s. Also let $h(s, a_j, s_j) = (s, a_j, g_j, \tau_j, r_j, s_j)$.

For all $j \in J$, $((s, x), a_j, (s_j, x[r_j])) \in \mathcal{E}^c \cup \mathcal{E}^u$ iff $g_j(x)$ and $x[r_j]$ is the timer valuation obtained from x when all the timers in r_j are set to zero and the others are left unchanged.

To define $\mathcal{E}^t$, we use the predicate φ, called *time progress function*. The notation $\varphi((s, x), t)$ means that time can progress from state (s, x) by t.

$$\varphi((s, x), t) \Leftrightarrow \bigwedge_{j \in J} \begin{cases} \tau_j = \delta \Rightarrow & \forall t' \in [0, t) \,.\, \neg \overset{\bullet}{\downarrow}_{b_s} g_j(x + t' b_s) \wedge \\ & \forall t' \in (0, t] \,.\, \neg \overset{\circ}{\downarrow}_{b_s} g_j(x + t' b_s) \\ \tau_j = \epsilon \Rightarrow & \forall t' \in [0, t) \,.\, \neg g_j(x + t' b_s) \end{cases}$$

If $\varphi((s, x), t)$, then $((s, x), t, (s, x + t b_s)) \in \mathcal{E}^t$ where $x + t b_s$ is the valuation obtained from x by increasing by t the timer values for which b_s elements are equal to one.

The above definition means that at control state s, time cannot progress whenever an eager transition is enabled, or beyond the falling edge of a delayable guard.

We will usually denote by TS a timed system. TS^c (resp. TS^u) represents the timed system composed of the controllable (resp. uncontrollable) transitions of TS only.

Property 2.1 If φ, φ^c, and φ^u are respectively the time progress functions of TS, TS^c, and TS^u then $\varphi = \varphi^c \wedge \varphi^u$.

Example 2.2 *(A periodic process)*
Let us model a periodic non-preemptible process P as a timed system. P is of period T and uses the CPU for an execution time E. It also has a relative deadline of D $(D \leq T)$.

As shown in fig. 2, the timed system has three control states, s, w, and e where P is respectively sleeping, waiting for the CPU, and executing on the CPU. The actions a, b, and f stand for arrive, begin, and finish. The timer x is used to measure execution time while the timer t measures the time elapsed since the process has arrived. In all states, both timers progress. The only controllable action is b.

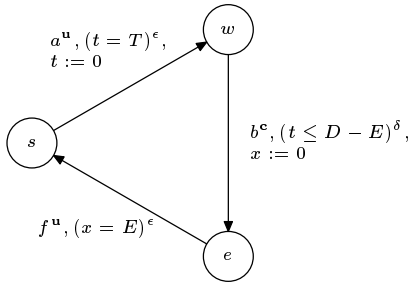

Fig. 2: A periodic process.

By convention, transition labels are of the form $a^{\mathbf{x}}, g^{\tau}, r$, where $\mathbf{x}$ can be $\mathbf{u}$ (uncontrollable) or $\mathbf{c}$ (controllable), and τ is an urgency type. The set r is omitted if it is empty.

Notice that since the transition b is delayable, the process might wait for a non-zero time although the CPU is free: idling is permitted. A non-idling process is modeled by changing the urgency type of the transition b to eager (see example 2.5 for further details).

A preemptive periodic process is modeled in section 3.3.

2.2 Restriction and control invariants

Definition 2.4 *(constraint)*
Given a timed system with a set of timers X and a set of control states $\{s_1, \ldots, s_n\}$, a *constraint* is a state predicate represented as an expression of the form $\bigvee_{i=1}^{n} s_i \wedge C_i$ where C_i is a X-constraint and s_i is (also) the boolean denoting presence at state s_i.

Definition 2.5 *(restriction)*
Let TS be a timed system and K be a constraint. The *restriction* of TS by K denoted TS/K, is the timed system TS where each guard g of a controllable transition $(s,\ a,\ g,\ \tau,\ r,\ s')$, is replaced by

$$g'(x) = g(x) \wedge \mathrm{K}(s',\ x[r])$$

Notice that in the restriction TS/K, the states reached right after execution of a controllable transition satisfy K. Moreover, it follows from the definition that $(\mathrm{TS}/\mathrm{K}_1)/\mathrm{K}_2 = \mathrm{TS}/(\mathrm{K}_1 \wedge \mathrm{K}_2)$.

Definition 2.6 *(proper invariant)*
Let TS be a timed system and K be a constraint. We say that K is a *proper invariant* of TS, denoted by $\mathrm{TS} \vDash \mathsf{inv}(\mathrm{K})$, if K is preserved by the edges of $\mathcal{E}$, i.e., $\forall(s,\ x)\ .\ \mathrm{K}(s,\ x) \Rightarrow \forall((s,\ x),\ \gamma,\ (s',\ x')) \in \mathcal{E}\ .\ \mathrm{K}(s',\ x')$.

Proper invariants, called simply invariants for closed systems, are constraints preserved by all the transitions of the system. We use the term "proper" to distinguish them from control invariants introduced in the following definition. Control invariants are constraints that are satisfied by the restricted system.

Definition 2.7 *(control invariant)*
Let TS be a timed system and K be a constraint. K is a *control invariant* of TS if $\mathrm{TS}/\mathrm{K} \vDash \mathsf{inv}(\mathrm{K})$.

Property 2.2 If K is a proper invariant of a timed system TS, then K is a control invariant of TS.

This property follows from the trivial observation that if TS and TS/K are initialized in K, then they have the same behavior. However, notice that control invariants are not proper invariants, in general.

Property 2.3 For any timed system TS and constraint K such that $\mathrm{TS}^u \vDash \mathsf{inv}(\mathrm{K})$, K is a control invariant of TS (i.e. $\mathrm{TS}/\mathrm{K} \vDash \mathsf{inv}(\mathrm{K})$).

Proof. *(sketch)* Assume $\mathrm{K}(s,\ x)$ for some state $(s,\ x)$. To prove $\mathrm{TS}/\mathrm{K} \vDash \mathsf{inv}(\mathrm{K})$ it must be shown that K is preserved in TS/K by (1) controllable, (2) uncontrollable, and (3) timed edges of TS/K. By construction of TS/K, (1) is true. From $\mathrm{TS}^u \vDash \mathsf{inv}(\mathrm{K})$, (2) and (3) follow. □

Definition 2.8 *(Timed system of processes)*
A timed system of processes is a timed system $\mathrm{TS} = (\mathrm{S},\ \mathrm{A},\ \mathrm{T},\ \mathrm{X},\ b,\ h)$ obtained by composition of processes where a process P_i is a timed system $(\mathrm{S}_i,\ \mathrm{A}_i,\ \mathrm{T}_i,\ \mathrm{X}_i,\ b_i,\ h_i)$. TS is the timed system of n processes $\{P_1, \ldots, P_n\}$ if

$$\begin{array}{l}
\mathrm{S} = \mathrm{S}_1 \times \ldots \times \mathrm{S}_n\ ;\ \mathrm{A} = \mathrm{A}_1 \cup \ldots \cup \mathrm{A}_n\ ;\ \mathrm{X} = \mathrm{X}_1 \cup \ldots \cup \mathrm{X}_n\ ; \\
\text{For } s = (s_1 \ldots s_n) \in \mathrm{S} \text{ and } x \in \mathrm{X}_i,\ b_s[x] = b_{i,s_i}[x]; \\
\text{For } s = (s_1 \ldots s_i \ldots s_n),\ \text{ and } s' = (s_1 \ldots s'_i \ldots s_n) \in \mathrm{S}, \\
\quad t = (s,\ a_i,\ s') \in \mathrm{T} \quad \Leftrightarrow \quad t_i = (s_i,\ a_i,\ s'_i) \in \mathrm{T}_i\ ; \\
\quad h(t) = (s,\ a_i,\ g_i,\ \tau_i,\ r_i,\ s') \quad \Leftrightarrow \quad h_i(t_i) = (s_i,\ a_i,\ g_i,\ \tau_i,\ r_i,\ s'_i)
\end{array}$$

We assume that processes have disjoint sets of control states, and timers. Moreover, we accept that guards are general constraints on timers and control states as in the definition 2.4.

Example 2.3 *(Mutual exclusion)*
Consider a timed system of n periodic non-preemptible processes $\{P_1, \ldots, P_n\}$, instances of the generic process of fig. 2, and the constraint

$$\mathrm{K}_{\mathsf{mutex}} = \bigwedge_{i \neq j} \neg e_i \vee \neg e_j$$

expressing mutual exclusion. It is trivial to check that $\mathrm{K}_{\mathsf{mutex}}$ is a control invariant, as $\mathrm{TS}^u \vDash \mathsf{inv}(\mathrm{K}_{\mathsf{mutex}})$. In fact, $\mathrm{K}_{\mathsf{mutex}}$ is time invariant and is preserved by uncontrollable transitions.

If TS is the timed system of two processes of fig. 3 and $\mathrm{K}_{\mathsf{mutex}} = \neg e_1 \vee \neg e_2$, then $\mathrm{TS}_1 = \mathrm{TS}/\mathrm{K}_{\mathsf{mutex}}$ is obtained by restricting the controllable guards g_{b_1} and g_{b_2} to

$$g'_{b_1} = (t_1 \leq D_1 - E_1) \wedge \neg e_2 = (t_1 \leq 10) \wedge \neg e_2$$
$$g'_{b_2} = (t_2 \leq D_2 - E_2) \wedge \neg e_1 = (t_2 \leq 3) \wedge \neg e_1.$$

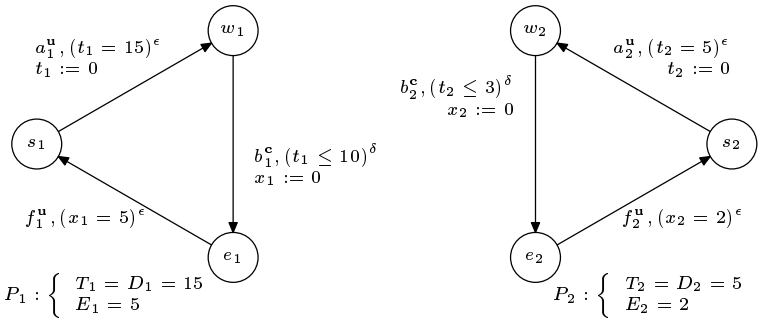

Fig. 3: A timed system of two processes.

2.3 Control Invariants and Synthesis

Following ideas in [11], synthesis is used to partially restrict the non-determinism of a system so as it satisfies a given invariant.

Problem 2.1 (SYNTH)
Solving the *synthesis problem* for a timed system TS and a constraint K amounts to giving a non-empty control invariant K' of TS which implies K, i.e. $\mathrm{K}' \Rightarrow \mathrm{K}$, $\mathrm{TS}/\mathrm{K}' \vDash \mathsf{inv}(\mathrm{K}')$.

We assume that the processes to be scheduled and their timing constraints are represented by a timed system of processes TS. Furthermore, we consider that scheduling requirements can be expressed as a constraint (safety property) K. A scheduled system can be obtained by solving the synthesis problem for TS and K, as explained in [1]. If K' is a control invariant implying K, then TS/K' describes a scheduled system.

We assume that the constraint K is in general the conjunction of two constraints $\mathrm{K} = \mathrm{K}_{\mathsf{algo}} \wedge \mathrm{K}_{\mathsf{sched}}$. $\mathrm{K}_{\mathsf{algo}}$ is an optional constraint characterizing a particular scheduling algorithm. We provide in section 3, a general framework for the decomposition of $\mathrm{K}_{\mathsf{algo}}$ and the modeling of different scheduling policies.

K_{sched} expresses the fact that the timing requirements of the processes are satisfied. We consider that the processes to be scheduled are structurally timelock-free [3]. This property means that time always eventually progresses. It is implied by the fact that at any control state, if no action is enabled then time can progress, and the requirement that in any circuit of the control graph a timer is reset and tested against some positive lower bound. For example, the periodic process of example 2.2 is structurally timelock-free.

Notice that structural timelock-freedom is preserved by restriction. For timelock-free timed systems, K_{sched} can be formulated as a constraint expressing the property that each process always eventually executes some action. This property implies fairness of the scheduling algorithm.

Definition 2.9 *($\Diamond$)*
Let C be a X-constraint, $s \in S$ a control state, and $k \in I\!N \cup \{\infty\}$. We will use the notation

$$(\Diamond_k^s C)(x) = \exists t \in [0,\, k]\ .\ C(x + tb_s)$$

to express the property "eventually C within k in s". If the state s is clear from the context, we write $\Diamond_k$ instead of $\Diamond_k^s$. We use $(\Diamond C)(x)$ for $\exists t \geq 0\ .\ C(x+t)$.

For a timed system of processes as in definition 2.8,

$$K_{sched} = \bigwedge_{P_i} K_{sched_i} \quad \text{where} \quad K_{sched_i} = \bigvee_{s \in S_i} s \wedge \Big(\bigvee_{(s,\,a,\,s') \in T_i} \Diamond g_a \Big)$$

It can be shown that in general, K_{sched} is not a control invariant. We have shown in [1] how maximal schedulers for timed automata and their schedulability constraints can be computed. The synthesis algorithm has been implemented in the KRONOS tool.

Example 2.4 *(Schedulability)*
The schedulability constraint for the timed system of n periodic processes TS as in example 2.3 is:

$$K_{sched} = \bigwedge_{P_i} (s_i \wedge \Diamond g_{a_i} \vee e_i \wedge \Diamond g_{f_i} \vee w_i \wedge \Diamond g_{b_i})$$

We consider the timed system of two processes described in fig. 3 where the mutual exclusion constraint has been applied. We have:

$$K_{sched} = \begin{bmatrix} & s_1 \wedge t_1 \leq 15 \\ \vee & e_1 \wedge x_1 \leq 5 \\ \vee & w_1 \wedge t_1 \leq 10 \end{bmatrix} \bigwedge \begin{bmatrix} & s_2 \wedge t_2 \leq 5 \\ \vee & e_2 \wedge x_2 \leq 2 \\ \vee & w_2 \wedge t_2 \leq 3 \end{bmatrix}$$

The maximal control invariant implying K_{sched} computed by KRONOS is:

$$K'_{sched} = \begin{bmatrix} & (s_1 \wedge s_2 \ \wedge\ t_1 <= 15 \wedge t_2 <= 5) \\ \vee & (w_1 \wedge s_2 \ \wedge\ (t_2 <= 3 \wedge t_1 \leq 10 \ \vee\ t_2 <= 5 \wedge t_1 <= t_2 + 3)) \\ \vee & (s_1 \wedge w_2 \ \wedge\ t_1 <= 15 \wedge t_2 <= 3) \\ \vee & (e_1 \wedge s_2 \ \wedge\ t_2 <= 5 \wedge x_1 <= 5 \wedge t_1 <= x_1 + 10 \wedge t_2 <= x_1 + 3) \\ \vee & (w_1 \wedge w_2 \ \wedge\ (t_1 <= 8 \wedge t_2 <= 1 \ \vee\ t_2 <= 3 \wedge t_1 <= t_2 + 3)) \\ \vee & (s_1 \wedge e_2 \ \wedge\ t_1 <= 15 \wedge x_2 <= 2 \wedge t_2 <= x_2 + 3) \\ \vee & (e_1 \wedge w_2 \ \wedge\ x_1 <= 5 \wedge t_1 <= x_1 + 10 \wedge t_2 + 2 <= x_1) \\ \vee & (w_1 \wedge e_2 \ \wedge\ (x_2 <= 2 \wedge t_1 <= x_2 + 8 \wedge t_2 <= x_2 + 1 \vee \\ & \ \ x_2 <= 2 \wedge t_1 <= t_2 + 3 \wedge t_2 <= x_2 + 3)) \end{bmatrix}$$

In the rest of the paper, we show how to construct control invariants for some frequently used scheduling algorithms *without* fixpoint computation.

2.4 Control Invariant Composability

Contrary to proper invariants, control invariants are not composable by conjunction. In general, it can not be inferred from $TS/K_i \models \mathsf{inv}(K_i)$, $i = 1, 2$ that $TS/(K_1 \wedge K_2) \models \mathsf{inv}(K_1 \wedge K_2)$. We study a notion of control invariant composability.

Definition 2.10 *(composable invariant)*
Let TS be a timed system and K_1 be a constraint. K_1 is a *composable invariant* of TS if for all constraints K_2, K_1 is a control invariant of TS/K_2 (i.e. if $TS/(K_1 \wedge K_2) \models \mathsf{inv}(K_1)$).

Property 2.4 Let TS be a timed system and K_1 be a constraint on TS. K_1 is a composable invariant of TS iff $TS^u \models \mathsf{inv}(K_1)$.

Proof. Let K_1 be a composable invariant of TS. By applying definition 2.10 with $K_2 = false$, we obtain: $TS/false = TS^u \models \mathsf{inv}(K_1)$.

Conversely, assume that $TS^u \models \mathsf{inv}(K_1)$ and let K_2 be some constraint. We show that $TS/(K_1 \wedge K_2) \models \mathsf{inv}(K_1)$. Let (s, x) be a state of TS such that $K_1(s, x)$. (1) If there exists a controllable edge $((s, x), a_c, (s', x'))$ in the transition graph of $TS/(K_1 \wedge K_2)$, then by definition 2.5 of restriction, $(K_1 \wedge K_2)(s', x')$, thus $K_1(s', x')$. (2) An uncontrollable edge $((s, x), a_u, (s', x'))$ of $TS/(K_1 \wedge K_2)$ is also an uncontrollable edge of TS^u, thus $K_1(s', x')$. (3) Let $\varphi_{(K_1 \wedge K_2)}$ be the time progress function of $TS/(K_1 \wedge K_2)$. According to the property 2.1, we have

$$\varphi_{(K_1 \wedge K_2)} = \varphi^c_{(K_1 \wedge K_2)} \wedge \varphi^u_{(K_1 \wedge K_2)} = \varphi^c_{(K_1 \wedge K_2)} \wedge \varphi^u.$$

If $((s, x), t, (s, x+tb_s))$ is a timed edge of $TS/(K_1 \wedge K_2)$, then it is also a timed edge of TS^u because $\varphi_{(K_1 \wedge K_2)} = \varphi^c_{(K_1 \wedge K_2)} \wedge \varphi^u$. Thus, $K_1(s, x+tb_s)$ from $TS^u \models \mathsf{inv}(K_1)$. □

Corollary 2.5 For a timed system TS and constraints K_1 and K_2, $TS^u \models \mathsf{inv}(K_1)$ and $(TS/K_1)/K_2 \models \mathsf{inv}(K_2)$ implies that $TS/(K_1 \wedge K_2) \models \mathsf{inv}(K_1 \wedge K_2)$.

That is, if K_1 is composable and if K_2 is a control invariant of TS/K_1 then $(K_1 \wedge K_2)$ is control invariant of TS.

This corollary justifies the incremental methodology for restricting a timed system. To impose a control invariant $K_1 \wedge K_2$ on TS, if K_1 is a composable invariant of TS, the restriction by a control invariant K_2 does not destroy the invariance of K_1.

Example 2.5 *(Non-idling constraint)*
A scheduling algorithm is said to be non-idle if the CPU cannot remain free when there is a pending request. Let us consider the timed system of n processes as in example 2.3. As $TS^u \models \mathsf{inv}(K_{\mathsf{mutex}})$, K_{mutex} is composable which means that K_{mutex} is a proper invariant of any system obtained by restriction of $TS_1 = TS/K_{\mathsf{mutex}}$.

In order to model non-idling, as remarked in example 2.2, all transitions b_i must have the urgency type *eager*. The non-idling constraint $K_{\mathsf{non\text{-}idle}}$ specifies that an enabled b_i action is fired as soon as the CPU is free.

$$K_{\mathsf{non\text{-}idle}} = \bigvee_{P_i} (e_i \vee x_i = E_i) \vee \bigwedge_{P_j} (s_j \vee w_j \wedge t_j = 0)$$

This means that in a non-idling system, if no process P_i is executing or has just finished its execution, then any process P_j is either sleeping or waiting for zero time.

It can be shown that $K_{\mathsf{non\text{-}idle}}$ is a proper invariant of TS_1. However, it fails to be composable, in general. For the example described in fig. 3, the constraint $K_{\mathsf{non\text{-}idle}}$ becomes:

$$\mathrm{K}_{\mathsf{non\text{-}idle}} = \left[\begin{array}{ll} & (e_1 \vee e_2) \ \vee \ (x_1 = 5 \vee x_2 = 2) \\ \vee & (s_1 \vee w_1 \wedge t_1 = 0) \wedge (s_2 \vee w_2 \wedge t_2 = 0) \end{array} \right.$$

Notice that $\mathrm{TS}_1/\mathrm{K}_{\mathsf{non\text{-}idle}} = \mathrm{TS}_1$, that is, restricting by $\mathrm{K}_{\mathsf{non\text{-}idle}}$ does not change controllable transitions of TS_1. It is easy to check that $\mathrm{TS}_1/(\mathrm{K}_{\mathsf{non\text{-}idle}} \wedge \mathrm{K}_{\mathsf{sched}}) \not\models \mathsf{inv}(\mathrm{K}_{\mathsf{non\text{-}idle}})$: consider for instance the eager transition b_1 from the control state $(w_1 s_2)$ to $(e_1 s_2)$ with guard $g'_{b_1} = \underbrace{t_1 \leq 10}_{g_{b_1}} \wedge \underbrace{t_2 \leq 3}_{\mathrm{K}'_{\mathsf{sched}}}$. When the system reaches the state $(w_1 s_2)$ with timer values $(t_1 = 0,\ t_2 = 4)$, the action b_1 is not enabled although the CPU is free due to the restriction $t_2 \leq 3$ imposed by $\mathrm{K}_{\mathsf{sched}}$. Thus, $\mathrm{K}_{\mathsf{non\text{-}idle}}$ is violated.

Imposing $\mathrm{K}_{\mathsf{sched}}$ has destroyed the property of the system to be non-idle. Thus the non-idling constraint is not composable. This is a consequence of the observation that a given scheduling problem with an idling solution may have no non-idling schedule.

The notion of composability described in this section allows to apply restrictions sequentially to build a system more and more close to the correct scheduler at each step.

3 Modeling scheduling algorithms

Timed systems with priorities are timed systems of processes with an associated set of priority orders on actions. They have been defined and studied in [4, 3]. We show how to model scheduling algorithms by specifying a timed system with priorities and that applying priorities is equivalent to restricting by a composable invariant.

3.1 Timed systems with priorities

Definition 3.1 *(priority order)*
Let $\prec \subseteq \mathrm{A} \times (\mathbb{N} \cup \{\infty\}) \times \mathrm{A}$ be a relation. $a_1 \prec_k a_2$ is written for $(a_1,\ k,\ a_2) \in \prec$. The relation $\prec$ is a *priority order* if $\forall k \in \mathbb{R}_+ \cup \{\infty\}$,

- $\prec_k$ is a partial order;
- $a_1 \prec_k a_2 \Rightarrow \forall k' < k \ .\ a_1 \prec_{k'} a_2$;
- $a_1 \prec_k a_2 \ \wedge \ a_2 \prec_l a_3 \Rightarrow a_1 \prec_{k+l} a_3$.

Definition 3.2 *(timed system with priorities)*
A timed system with priorities (TS, pr) is the timed system of processes TS equipped with a *priority rule*, i.e., a finite set of pairs $pr = \{(C^i,\ \prec^i)\}_i$, where $\prec^i$ is a priority order, and C^i is a X-constraint that specifies when the priority order applies, such that

- $C^i \wedge C^j \neq false \ \Rightarrow \prec^i \cup \prec^j$ is a priority order;
- No uncontrollable action is dominated in $\prec^i$;
- $(C^i,\ \prec^i) \in pr$ and $(a,\ k,\ b) \in \prec^i$ imply that transitions labeled by a do not reset any timer occurring in C^i.

For each state $s \in \mathrm{S}$, let $\{(s,\ a_i,\ s_i)\}_{i \in I}$ be the set of the transitions departing from s, and $h(s,\ a_i,\ s_i) = (s,\ a_i,\ g_i,\ \tau_i,\ r_i,\ s_i)$. The timed system with priorities

(TS, pr) represents a timed system TS$'$ obtained from TS by replacing the guards g_j of TS by g'_j defined as follows:

$$g'_j = g_j \wedge \bigwedge_{(C,\prec)\in pr} \left(\neg C \vee \bigwedge_{\substack{\exists i \in I. \\ a_j \prec_k a_i}} \neg \Diamond^s_k g_i\right)$$

This formula says that an action a_j is allowed if there is no transition a_i leaving s that has priority over a_j, and that will become enabled within a delay of k.

Example 3.1 *(*edf *policy)*
Consider the timed system TS_1 of n non-preemptible periodic processes, on which $\mathrm{K}_{\mathsf{mutex}}$ has already been applied, as in example 2.3.

We show how the basic earliest deadline first (edf, [7]) mechanism can be specified by using a priority rule. A scheduler follows an edf policy if the CPU is granted to the waiting process that is closest to its relative deadline.

The edf policy is partially specified as follows:

$$pr^1_{\mathsf{edf}} \;=\; \{(D_i - t_i < D_j - t_j, \quad \{b_j \prec_0 b_i\})\}_{i\neq j}$$

i.e., whenever there are two processes P_i and P_j waiting for the CPU, the action b_i has immediate priority over the action b_j if P_i is closer to its relative deadline than P_j (namely, $D_i - t_i < D_j - t_j$).

It is easy to check that pr satisfies the requirements of definition 3.2. In particular, note that the constraints $D_i - t_i < D_j - t_j$ define a partial order on the set of b_i actions. The complete specification of the edf policy is given in example 3.3.

3.2 Priorities as restriction

We show that applying a priority rule amounts to restricting by a particular constraint. To obtain this result, we construct from (TS, pr) a timed system TS$'$ that is *strongly equivalent* to TS, and a constraint K_{pr} such that (TS, pr) is strongly equivalent to TS$'$/K_{pr}. Strong equivalence means that for any state of TS there exists a state of TS$'$ such that the transition graphs are strongly bisimilar from these states, and conversely. The construction has only a theoretical interest and is used to show that K_{pr} is a composable invariant.

Let (TS, pr) be a timed system with priorities. In order to interpret priorities on TS as a constraint, we have to identify the states reached right after firing a restricted transition.

For this we transform TS = (S, A, T, X, b, h) into a strongly equivalent timed system TS$'$ = (S$'$, A, T$'$, X, b', h') with S$'$ $\subseteq$ S $\cup$ (S $\times$ A), by iterative application of a state splitting procedure which creates for each transition a unique target control state.

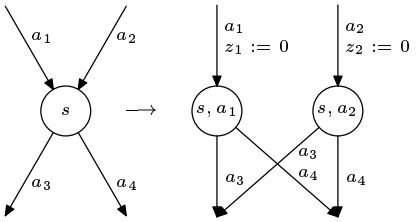

Fig. 5: The splitting procedure.

For each state $s \in$ S with an incident transition of the form $t = (ss, a_j, s)$ where $ss \in$ S$'$ and $a_j \in$ A, the splitting procedure removes t and creates a new transition $t' = (ss, a_j, (s, a_j))$. t' is labeled as t with in addition a reset of a new timer z_j. Notice that in TS$'$ the set of states reached right after the execution of a_j is characterized by $((s, a_j) \wedge z_j = 0)$. For all states $s \in$ S$'$, we take $b'_s[z_j] = 1$.

Property 3.1 Let (TS, pr) be a timed system with priorities, and TS$'$ be the result of the splitting procedure on TS. The constraint

$$\mathrm{K}_{pr} = \bigwedge_{s \in \mathrm{S}'} \bigwedge_{(C,\prec) \in pr} \bigwedge_{\substack{i,j \in I \\ a_j \prec_k a_i}} \Big(s_j \wedge z_j = 0 \Rightarrow (\neg \Diamond_k^s g_i \vee \neg C) \Big)$$

is a composable invariant of TS$'$, and (TS$'$, pr) = TS$'$/K$_{pr}$, where for a given s, $\{(s,\, a_i,\, g_i,\, \tau_i,\, r_i,\, s_i)\}_{i \in I}$ is the set of transitions departing from s.

Proof. Notice that K$_{pr}$ contains all the states but the ones that would be reached by firing a transition violating the priority rule.

(TS$'$, pr) = TS$'$/K$_{pr}$ is obtained immediately by comparing syntactically the result of restriction by K$_{pr}$ with the application of the priority rule pr.

To prove composability, we show that TS$'^u \vDash \mathsf{inv}(\mathrm{K}_{pr})$. Let $(s,\, x)$ be a state of TS$'^u$ such that K$_{pr}(s,\, x)$. (2) If there exists an uncontrollable edge $((s,\, x),\, a_u,\, (s',\, x'))$ in TS$'^u$, then K$_{pr}$ cannot contain a constraint of the form $s' \wedge z = 0 \Rightarrow \neg C \vee \neg \Diamond_k^{s''} g$, since a_u is the only transition leading to s' in TS$'^u$. Thus, K$_{pr}(s',\, x')$. (3) If time can progress by $t > 0$ from $(s,\, x)$ in TS$'^u$, then K$_{pr}(s,\, x + tb_s)$ obviously holds. □

Corollary 3.2 Let (TS, pr) be a timed system with priorities, K be a control invariant of (TS, pr), ((TS, pr)/K)$'$ be the result of the splitting procedure on (TS, pr)/K, and K$_{pr}$ the constraint associated to pr. Then ((TS, pr)/K)$' \vDash \mathsf{inv}(\mathrm{K}_{pr})$.

These results say that applying a priority rule can be seen as a restriction of a strongly equivalent timed system by a control invariant. Furthermore, whenever some other control invariant K is applied to (TS, pr), then (TS, pr)/K still satisfies the priority rule pr.

In some cases, the property 3.1 holds without applying the splitting procedure, as shown in the following examples.

3.3 Basic scheduling algorithms

We model scheduling policies by using priorities and constraints. We consider the first in first out policy, the earliest deadline first policy, and the fixed priority policy with preemption. The system to schedule is the timed system of n processes $\{P_1, \ldots, P_n\}$ as in the previous examples, where K$_{\mathsf{mutex}}$ has already been applied.

Example 3.2 *(*fifo *policy)*
A scheduler follows a first in first out policy (fifo) if the CPU is granted to the process that has been waiting for the longest time. For non-preemptible processes, fifo is specified by using priorities as follows:

$$pr_{\mathsf{fifo}} = \{(t_j < t_i,\, \{b_j \prec_0 b_i\})\}_{i \neq j}$$

This means that whenever two processes P_i and P_j are both waiting for the CPU, b_i has priority over b_j if process P_i has been waiting for longer time than process P_j, i.e. $t_j < t_i$.

The following property shows that the constraint

$$\mathrm{K}_{\mathsf{fifo}} = \bigwedge_{i \neq j} \big(w_i \wedge e_j \wedge x_j = 0 \;\Rightarrow\; t_i \leq t_j \big)$$

associated with pr_{fifo} is also a composable control invariant of TS.

Property 3.3 (TS, pr_{fifo}) = TS/$\mathrm{K}_{\mathsf{fifo}}$, and $\mathrm{K}_{\mathsf{fifo}}$ is a composable control invariant for TS.

Proof. As TS is already restricted by $\mathrm{K}_{\mathsf{mutex}}$, among the transitions that reach a state, there is at most one transition dominated in pr, thus its guard in TS/K_{pr} is the same as the corresponding guard in (TS, pr).

Then, we show that $\mathrm{TS}^u \vDash \mathsf{inv}(\mathrm{K}_{\mathsf{fifo}})$: assume $\mathrm{K}_{\mathsf{fifo}}(s, x)$ for some state (s, x) of TS^u. (2) By taking an uncontrollable transition from (s, x) this leads to (s', x'). If it is an action a_i then (s', x') satisfies $w_i \wedge t_i = 0$. If there exists j such that (s, x) satisfies $e_j \wedge x_j = 0$ then (s', x') satisfies $t_i \leq t_j$ since $t_j = 0$. Hence, $\mathrm{K}_{\mathsf{fifo}}(s', x')$. If it is an action f_i then $\mathrm{K}_{\mathsf{fifo}}(s', x')$ obviously holds. (3) If time can elapse by $t > 0$ from (s, x) then $\mathrm{K}_{\mathsf{fifo}}(s, x + tb_s)$. □

Example 3.3 *(*edf *policy)*
We showed in example 3.1 how to model partially the edf policy on TS as a priority rule, pr^1_{edf}. But this specification has to be completed since in case a process P_i arrives (transition a_i) exactly when the decision to allot the CPU to another process is made, this might be wrong depending on whether P_i was taken into account or not. This confusion situation can be prevented by a priority rule ensuring that the set of waiting processes is up to date before any decision is made. Therefore, processes arrival actions a_i are given priority over b_j actions:

$$pr_{\mathsf{edf}} = pr^1_{\mathsf{edf}} \cup \{(t_i = T_i, \quad \{b_j \prec_0 a_i\})\}_{i \neq j}$$

Let $\mathrm{K}_{\mathsf{edf}}$ be the constraint associated with pr_{edf}:

$$\begin{array}{rcl} \mathrm{K}_{\mathsf{edf}} = & \bigwedge_{i \neq j} & w_i \wedge e_j \wedge x_j = 0 \;\Rightarrow\; D_j - t_j \leq D_i - t_i \\ & \wedge & s_i \wedge e_j \wedge x_j = 0 \;\Rightarrow\; t_i \neq T_i \end{array}$$

Property 3.4 (TS, pr_{edf}) = TS/$\mathrm{K}_{\mathsf{edf}}$, and $\mathrm{K}_{\mathsf{edf}}$ is a composable control invariant.

The proof is slightly more complex but similar to the previous one.

Preemptive fixed-priority scheduling

Preemptive fixed-priority scheduling assigns the CPU according to some fixed priority order between the processes to be scheduled. If the CPU is free, the highest priority process among the waiting processes is scheduled. An arriving process can preempt a running process of lower priority.

Fig. 7 shows the model of a preemptible process. It has an additional control state p (preempted), and two more transitions: pr (preempt) and rs (resume). The timer x is stopped in control state p, i.e. $b_p[x] = 0$. Everywhere else, all timers progress. The timer x_{pr} measures the time elapsed since the process has been preempted.

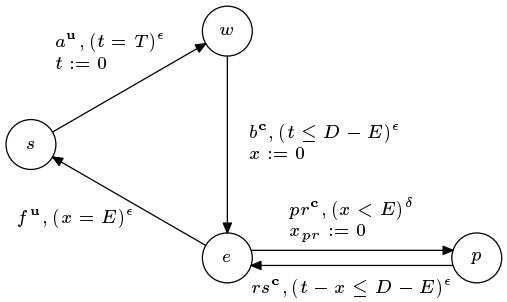

Fig. 7: A preemptible process.

Consider system of n processes P_1, $\ldots, P_n$ as shown in fig. 7 with the given fixed priorities $\pi_1, \ldots, \pi_n$, where $\pi_i < \pi_j$ means that P_j has priority over P_i. As before, mutual exclusion is achieved by application of $\mathrm{K}_{\mathsf{mutex}}$. We construct the scheduled system of these processes according to the preemptive policy with the priorities $\pi_1, \ldots, \pi_n$ as follows.

Process priorities. Priorities between the processes are specified by the priority rule pr_π on the CPU allocating actions b and rs:

$$\begin{array}{lll} pr_\pi = & \{(true, & \{b_i \prec_0 b_j,\ b_i \prec_0 rs_j,\ rs_i \prec_0 b_j,\ rs_i \prec_0 rs_j\}), \\ & (t_j = T_j, & \{b_i \prec_0 a_j,\ rs_i \prec a_j\})\}_{\pi_i < \pi_j} \end{array}$$

The first line says that the CPU is granted — by an action b_j or rs_j — to a process P_j that has highest priority among the waiting processes. Here, the constraint that specifies when the priority order applies is *true*, since the priorities are fixed and do not depend on timer valuations. The second line guarantees that the set of waiting processes is up to date before a new process is scheduled.

It is easy to show that pr_π satisfies the definition of a priority rule.

Property 3.5 Let K_{pr_π} be the constraint associated to pr_π. Then, $(\mathrm{TS}, pr_\pi) = \mathrm{TS}/\mathrm{K}_{pr_\pi}$, and K_{pr_π} is a composable control invariant of TS.

This property can be proven by using the same kind of arguments as for the fifo example.

pr_π only specifies the CPU allocation policy, but not the mechanism preempting a running process, which will be imposed by a further constraint $\mathrm{K}_{\mathsf{pmtn}}$.

Preemption. Preemption is enforced by the constraint

$$\mathrm{K}_{\mathsf{pmtn}} = \bigwedge_i \left(p_i \wedge x_{pr_i} = 0 \Rightarrow \exists j \ .\ \pi_j > \pi_i \wedge t_j = 0\right)$$

It means that a process P_i must not take the pr_i action unless there is a higher priority process P_j that has just arrived. Notice that for given process priorities $\pi_1, \ldots, \pi_n$, the term $\exists j \ .\ \pi_j > \pi_i \wedge t_j = 0$ is a X-constraint.

Applying $\mathrm{K}_{\mathsf{pmtn}}$ restricts the guard of pr_i actions to

$$g_{pr_i} = x_i < E_i \wedge \bigvee_{\exists j \ .\ \pi_j > \pi_i} t_j = 0$$

In other words, a running process is preempted as soon as a process of higher priority arrives. Immediately after that, since the a_i actions are eager, the CPU is assigned to a waiting process according to pr.

$\mathrm{K}_{\mathsf{pmtn}}$ is a control invariant for TS, thus from corollary 2.5, $\mathrm{K}_{\mathsf{pmtn}} \wedge \mathrm{K}_\pi$ is also a control invariant of TS. But $\mathrm{K}_{\mathsf{pmtn}}$ is not composable, and neither is $\mathrm{K}_{\mathsf{pmtn}} \wedge \mathrm{K}_\pi$.

Example 3.4 *(*rms *with preemption)*
The algorithm of preemptive rate-monotonic scheduling (rms, [7]) assigns to each process a fixed priority such that processes with shorter period have higher priority, i.e., $T_i > T_j \Rightarrow \pi_i < \pi_j$.

The invariant K_π can be obtained from pr_π as before. As remarked above, $\mathrm{K}_{\mathsf{pmtn}} \wedge \mathrm{K}_\pi$ is not composable. However, the rms policy makes the scheduled system $(\mathrm{TS}, pr_\pi)/\mathrm{K}_{\mathsf{pmtn}}$ nearly deterministic since π defines a total order. Therefore, there is no need to further restrict the system — it is either schedulable or not.

In the same way, the deadline monotonic policy [2] is modeled by specifying the fixed process priorities as $D_i > D_j \Rightarrow \pi_i < \pi_j$.

4 Conclusion

This work aims at bridging the gap between scheduling theory and timed systems specification and analysis. From the general idea that a scheduler is a controller

of the scheduled processes, we elaborate a methodology for the construction of a scheduled system. The methodology is illustrated on periodic processes but it can be applied to arbitrary systems of structurally timelock-free processes.

A contribution of this work is the decomposition of scheduling requirements into classes of requirements that can be expressed as safety constraints. We believe that the decomposition allows better understanding of scheduling problems and clarification of the differences between the two approaches. Scheduling theory studies sufficient conditions guaranteeing K_{sched} for particular scheduling algorithms characterized by some K_{algo}. On the contrary, timed systems specification and analysis have focused so far on the extraction of behaviors satisfying K_{sched} from a global model.

This work relates controller synthesis by means of the notion of control invariant, to a methodology for constructing a scheduled system satisfying given requirements. The existence of composable control invariants allows the automatic application of the corresponding synthesis steps. Not surprisingly, finding control invariants for schedulability is the hard problem that deserves further investigation. Possible directions are the development of specific synthesis algorithms or the use of constructive correctness techniques as in [3].

This work is developed in the framework of a project on real-time systems modeling and validation. We have applied the methodology to the description of the ceiling protocol [12] and are currently developing tools supporting the methodology.

References

[1] K. Altisen, G. Gößler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In IEEE Computer Society, editor, *RTSS 1999 proceedings*, pages 154–163, 1999.

[2] N.C. Audsley, A. Burns, and M.F. Richardson. Hard real-time scheduling: the deadline monotonic approach. In *Workshop on real-time operating systems and software*, 1991.

[3] S. Bornot, G. Gößler, and J. Sifakis. On the construction of live timed systems. In *TACAS 2000 proceedings*, volume 1785 of *LNCS*. Springer-Verlag, 2000.

[4] S. Bornot and J. Sifakis. On the composition of hybrid systems. In *International NATO School on "Verification of Digital and Hybrid Systems"*, LNCS. Springer Verlag, June 1997.

[5] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: A class of decidable hybrid systems. *Information and Computation*, 1992. LNCS 736, Springer-Verlag.

[6] H.-H. Kwak, I. Lee, A. Philippou, J.-Y. Choi, and O. Sokolsky. Symbolic schedulability analysis of real-time systems. In IEEE Computer Society, editor, *RTSS 1999 proceedings*, pages 409–418, 1998.

[7] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.

[8] Z. Liu and M. Joseph. Specification and verification of fault-tolerance, timing, and scheduling. *ACM Transactions on Programming Languages and Systems*, 21(1):46–89, January 1999.

[9] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In E.W. Mayr and C. Puech, editors, *proceedings of STACS'95*, pages 229–242. LNCS 900, Springer Verlag, 1995.

[10] P. Niebert and S. Yovine. Computing optimal operation schemes for chemical plants in multi-batch mode. In Springer Verlag, editor, *Proceedings of Hybrid Systems, Computation and Control*, volume 1790 of *LNCS*, March 2000.

[11] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event systems. *Journal of Control and Optimization*, 25(1):206–230, 1987.

[12] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 1990.

[13] S. Vestal. Modeling and verification of real-time software using extended linear hybrid automata. In *Fifth NASA Langley Formal Methods Workshop*, 2000.