Suspension Automata: A Decidable Class of Hybrid Automata*

Jennifer McManis and Pravin Varaiya

University of California at Berkeley

Abstract. A hybrid automaton consists of a discrete state component represented by a finite automaton, coupled with a (vector) continuous state component governed by a differential equation. For hybrid automata it is possible to reduce certain verification problems to those of checking language containment or language emptiness. Here we present a class of hybrid automata called suspension automata for which conditions can be given under which these problems are decidable.

1 Introduction

The state of a hybrid system has two components. There is a discrete component which evolves as in a finite automaton, and a (vector) continuous component which is governed by a differential equation. The two components interact: the differential equation in force at any time is determined by the state of the automaton, and the occurrence of transitions of the automaton is determined by the value of the continuous component. Details of the modeling formalism depend on tradition and on the purpose at hand as seen in the recent volume [1]. The model used here is a modification of the 'hybrid automaton' of [2]. It is called the 'suspension automaton.'

Verification problems for hybrid automata are not always decidable. The most important decidable special case is the timed automaton [3, 4, 5]. A timed automaton comprises a finite automaton and a collection of 'timers.' A timer has a continuous positive component which increases at rate one. The value of the timer states enables or disables discrete transitions of the system. At such transitions, a timer's value may be reset to zero. Timed automata are used to impart real-time constraints on the transitions of finite state machines. Important digital circuit timing analysis and (bounded) timed Petri nets can be formulated in this way. Software implementations of decision procedures for these automata are being developed [6, 7].

For many other hybrid systems the timed automaton is not a sufficiently descriptive model. Among these are common scheduling, synchronization, and communication policies. These systems require that, at the minimum, one should be able to assign timers a rate of 0 as well as 1. The following is a simple example of this sort of system.

* Research supported by National Science Foundation under grants ECS111907 and IRI9120074.

Example 1. Scheduling Jobs Under Preemptive Least Time Remaining Policy

Suppose a single processor must perform several different types of tasks upon request. For each task type there is a lower bound on the time between requests and a fixed time needed to perform the task. If more than one task has been requested at a given time then the processor must have some policy for determining which task is performed first. One common policy is the *least time remaining* policy. Under this policy the processor performs the task requiring the least amount of time to complete. As a concrete example, suppose we have two task types with the following characteristics.

Task 1: interarrival time ≥ 8 and processing time $= 2$

Task 2: interarrival time ≥ 4 and processing time $= 3$

One question that is commonly asked is: 'Will the processor complete a task of a given type before the next task of that type is requested?' In this case it is easy to see that the answer to the question will be 'no.' If Task 2 is preempted for service of Task 1, then a second request for Task 2 may arrive before completion of service to the originally requested task.

Formally, problems such as those posed for Ex. 1 may be addressed by examining the logical sequencing of events possible for the system. One common technique is verification through language containment:

> Given a specification represented by a language (set of event sequences) $\mathcal{L}^{spec}$ and system behavior represented by a language $\mathcal{L}^{sys}$ check whether $\mathcal{L}^{sys} \subseteq \mathcal{L}^{spec}$?

This problem is decidable if both $\mathcal{L}^{sys}$ and $\mathcal{L}^{spec}$ are ω-regular, that is finite state representations exist for the languages. In general, the language $\mathcal{L}^{sys}$ for a hybrid system need not be ω-regular. This paper is devoted to defining conditions under which a finite state representation may be derived.

A suspension automaton is an extension of a timed automaton in which a timer can be 'suspended,' i.e., its rate of increase can be set to 0. A suspended timer can later be unsuspended to resume increasing at rate 1. A 'rate assignment' specifies which timers are suspended or unsuspended as a function of the discrete components of the state. Suspension automata can be used to represent scheduling, synchronization, and communication policies in a very natural fashion. Unfortunately, the untimed language of a suspension automata is not guaranteed to be ω-regular. In order to avoid this problem, additional restrictions must be made to the form of the automata. In this paper, a trick is introduced by which one may replace the suspension of a timer by the decrementation of its value. In Sect. 2 the suspension automaton is defined. It is shown in Sect. 3 that the system of Ex. 1 may be modeled using the suspension automaton. In Sect. 4 a condition is given under which the untimed language is guaranteed to be finite state. It is shown that the system in Ex.1 may be represented by an automaton

satisfying the conditions. However, two problems remain. First, the conditions given in Sect. 4 are not easily checked. Second, the automata representing the systems of interest which satisfy the conditions are not easily derived. In Sect. 5 a systematic method is given for transforming a class of suspension automata (including those such as the one given in Sect. 3) into automata guaranteed to satisfy the conditions given in Sect. 4. Together these results give a reasonable procedure for verifying an interesting class of hybrid systems.

2 The Suspension Automaton

The suspension automaton defined below is similar to the hybrid automaton of [2]. It limits the dynamic behavior of the timers to a rate of increase of either one or zero. In addition, it expands the timer reset operation to allow for the decrementation of timer values by integer amounts. A *suspension automaton* is a tuple $(\mathcal{A}, L_0, \mathcal{F})$ where $\mathcal{A}$ is a suspension transition system denoting the causal behavior of the system, $L_0 \subseteq L$ is a restriction on the initial conditions of the system, and $\mathcal{F} \subseteq \mathcal{P}(L)$ (where $\mathcal{P}(\cdot)$ represents the power set) is a restriction on the set of locations visited infinitely often.

Formally, $\mathcal{A}$ is a tuple $(L, \Sigma, T, R, V_{inc}, Edge)$ where:

- L is a finite set of locations representing the discrete state of the system.
- Σ is a finite event set.
- T is a collection of timers $\{T_i\}_{i=1}^N$. Each timer has associated with it $v_i \in \mathbb{R}$, the value of the timer. Let v be a vector of dimension N representing the timer values.
- $R = \{R^l : l \in L\}$ is the rate assignment where $R^l \subseteq \{0,1\}^N$ defines the possible rates of timers in location l. For any $r \in R^l$, r_i gives the rate of T_i. If R^l has a single element for each $l \in L$ say that the automaton is *rate deterministic.*
- $V_{inc} = \{V^l : l \in L\}$ where each $V^l \subseteq \mathbb{R}^N$ is an inclusion condition. While in any given location l, the timer values must remain in V^l.
- $Edge \subseteq L \times \Sigma \times L \times \mathcal{P}(\mathbb{R}^N) \times Reset$ is an edge relation. Each $e \in Edge$ is of the form $e = (l, \sigma, l', V^e, reset^e)$ where:
  - l is the current location, σ an event, and l' the next location. This corresponds to the discrete edge transition.
  - V^e is an enabling condition. In order to make a transition, the current timer values must be in V^e.
  - $reset^e$ is a collection of conditions each of which either resets the timer value to 0 (denoted by $v_i := 0$) or decrements the timer by some fixed integer constant (denoted by $v_i := v_i - c$).

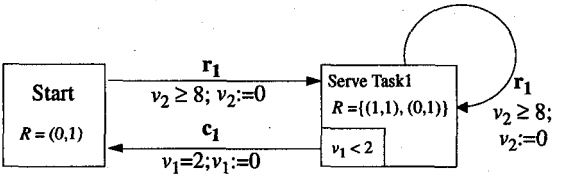
Only very specific forms of V^e and V^l will be considered. These will be called *simple regions.* Say that $V \subseteq \mathbb{R}^N$ is a simple region if there exists a set of constraint equations each of the form; $v_i \sim c$ or $v_i - v_j \sim c$ where $\sim$ is one of

$<$, $>$, $\leq$, $\geq$ or $=$, and $c \in \mathbb{Z}$, such that V is the set of all $v \in \mathbb{R}$ satisfying the constraints.

The suspension automaton may be depicted by a labeled directed graph. Ex. 2 shows this representation. Here, each location is given as a node with a location label, rate assignment, and inclusion set denoted as a collection of constraint equations. Edges are represented as directed arcs between nodes labeled with an event, a set of constraint equations representing V^e, and $reset^e$.

Example 2. The Representation of the Task 1 Component of the System

There are two events – the arrival of a request denoted by $\mathbf{r_1}$ and the completion of task service denoted by $\mathbf{c_1}$. The request interarrival time is regulated by an interarrival timer T_2 and the service time by a process timer T_1.



The constraint $v_2 \geq 8$ ensures that the task request event $\mathbf{r_1}$ occurs no more often that every 8 time units. The nondeterministic rate assignment indicates that the rate is not under the control of the task, but subject to external control. The inclusion condition $v_1 < 2$ together with the enabling condition $v_1 = 2$ ensures that the completion event $\mathbf{c_1}$ will occur when the task has received the required amount of service.

2.1 Behavior of Suspension Systems

Formally, the behavior of the system will be represented in terms of runs. The run describes the behavior of the system in terms of timer behavior as the system moves from location to location. Formally, given a sequence ρ drawn from Σ, a *run* for ρ with respect to $\mathcal{A}$ is an object of the following form:

$$v(0)(l(0), t(0))_{v'(0)} \xrightarrow{\rho(1)}_{v(1)} (l(1), t(1))_{v'(1)} \xrightarrow{\rho(2)}_{v(2)} (l(2), t(2))_{v'(2)} \ldots$$

where:

- $v(0) = 0$.
- for all $n \geq 0$:
  - $t(n) \geq 0$.
  - There exists $r \in R^l$ where $l = l(n)$ such that $v'(n) = v(n) + rt(n)$.
  - For all $0 \leq t < t(n)$, $v(n) + rt \in V^l$ where $l = l(n)$.
  - There exists an edge $e = (l, \sigma, l', V^e, reset^e)$ where:

* $l = l(n)$, $\sigma = \rho(n+1)$, and $l' = l(n+1)$.
* The enabling conditions are satisfied – $v'(n) \in V^e$.
* The reset conditions are satisfied.
  - If $v_i := 0 \in reset^e$, then $v_i(n+1) = 0$.
  - Else if $v_i := v_i - c \in reset^e$, then $v_i(n+1) = v_i'(n) - c$.
  - Else, $v_i(n+1) = v_i'(n)$.

A run is said to be an *accepting run* if $l(0) \in L_0$ and $inf(\{l\}) \in \mathcal{F}$ where $inf(\{l\})$ is the set of all locations visited infinitely often. Let $\mathcal{L}^{seq}(\mathcal{A}, L_0, \mathcal{F})$ be the set of all ρ with accepting runs.

3 Representation of Scheduling Using Suspension Automata

Scheduling may be represented quite naturally through rate assignment where a process timer is assigned rate 1 if its task is of highest priority and rate zero otherwise. Fig. 1 shows this for Ex. 1. For the sake of simplicity, the representation of interarrival times is omitted. In addition, edges representing requests arriving before completion of already requested tasks of the same type are not shown. The locations represent the status of the tasks in the system – waiting for service, receiving service, or not requested. There are only two timers, process timers T_1 for Task 1 and T_2 for Task 2. The timer values have the interpretation of processor time received. Thus the priority decision (and hence the rate assignment) may be formulated as a property of the timer values.

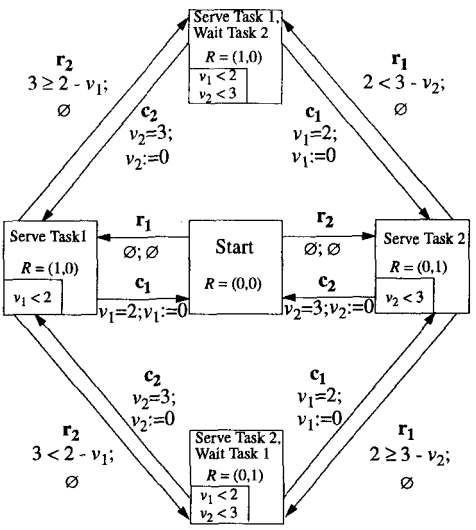


Fig. 1. Representation of least time remaining policy using timer suspension

The automaton in Fig. 2 represents an alternative way of representing scheduling. This automaton is not as straightforward to interpret, but can be show to have a finite state representation for its untimed language. In this case, preempted tasks have their process timers decremented by the service time of the preempting task. Although the automaton is similar to that of Fig. 1 several complications are introduced. Now a distinction must be made between a task request arriving and having to wait and a task which is receiving service being preempted. Also, the timer values when decremented no longer have the interpretation of processor time received. Although this does not affect transitions for two task systems, in general, this will make the priority decision harder to formulate in terms of timer values.

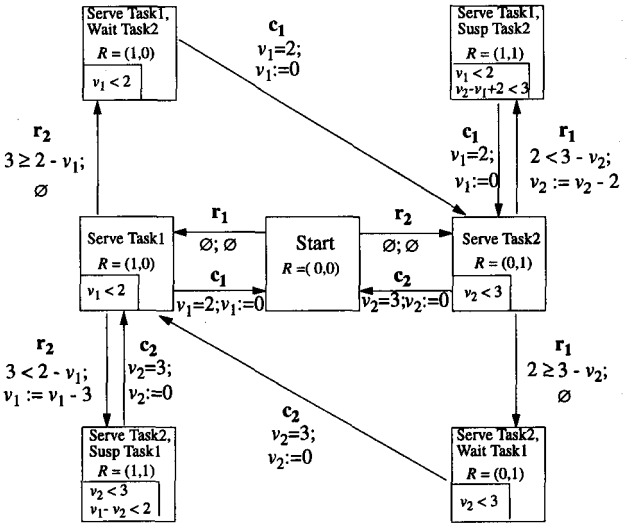


Fig. 2. Representation of least time remaining policy using timer decrementation

4 A Decidability Result for Decrementation Automata

In general, the untimed language of a suspension automaton need not be finite state. In this section one set of conditions is given under which a finite state representation is guaranteed. The result presented here is a simple extension of the result for timed automata [4, 5]. The reasoning is as follows. The full state of any suspension automaton is given by the location, current rate, and values of the timers. This state evolves continuously with time. However, for a given location and rate assignment, the evolution of the timer values may be fully predicted for the duration of the location knowing their values when the system first entered that location. Thus one may define a sampled transition system

on the state space $L \times \{0,1\}^N \times \mathbb{R}^N$ where only the values of the timers upon initially entering a location are tracked. This transition system is by no means finite state. However, one may combine states which are indistinguishable in terms of past and future logical behavior. The resulting automaton is called the *timer region automaton.*

The above approach is the one taken in [4] to show that timed automata have ω-regular languages. The proposed state equivalence groups timer values into *timer regions* based on the following two observations. First, in terms of the transitions taken, it is enough to know the integer components of the timer values and the relative ordering of the fractional components of the timer values. Second, there is some upper bound past which the value of the timer no longer matters at all.

This result may be extended by making two more observations:

- If $v_i = 0$, then the suspension of T_i is represented by keeping $v_i = 0$.
- If v_i is decremented by an integer amount, the relative ordering of the fractional parts of the timers does not change, and the change in the integer value is predictable.

Thus, under the above two conditions it should still be sufficient to track just the relative ordering of the fractional values and the integer values of the timers. What remains is to give conditions under which suitable upper and lower bounds on the region of interest for the timer values may be derived. The following theorem gives one such set of conditions. This set of conditions is not in itself checkable, but in Sect. 5 it is shown how to derive automata that are guaranteed to satisfy the conditions and model interesting systems such scheduling policies.

Theorem 1. *Given an accepting run for a rate deterministic automaton* $(\mathcal{A}, L_0, \mathcal{F})$ *let* $dec(i,n)$ *be the amount* v_i *was decremented by upon the* n^{th} *transition.*

Suppose for all accepting runs for each $T_i \in T$ *there exists* K_i^1 *and* K_i^2 *such that:*

- *If* $r_i = 0$, *then* $v_i = 0$.
- *For all* M, $\sum_{m=1}^{M} dec(i,m) - t(m) \leq K_i^1$
- *For all* M, *whenever* $v_i(M) \geq K_i^2$, v_i *is no longer decremented and for any edge taken, the constraints representing* V^e *do not involve the comparison of* v_i *to another timer.*

If the above conditions are satisfied, then $\mathcal{L}^{seq}(\mathcal{A}, L_0, \mathcal{F})$ *is* ω*-regular.*

Intuitively, the first condition ensures that timers may be suspended only if their value is fixed. The second condition gives the lower bound and the third condition gives the upper bound. A timer may never have a value less than K_i^1 and once it is greater than K_i^2 its actual value ceases to be of importance.

Note that the automaton of Fig. 2 satisfies the condition of Theorem 1. That is, $r_i = 0$ only when $v_i = 0$, and the values $K_1^1 = 3$, $K_1^2 = 2$, $K_2^1 = 2$, and $K_2^2 = 3$ satisfy the remaining conditions.

4.1 Defining A Timer Region Automaton

In order to prove Theorem 1, we will first define the timer regions and then specify the transition function between regions as a function of the edges of the suspension automaton. This is sufficient to define a finite state machine accepting $\mathcal{L}^{seq}(\mathcal{A}, L_0, \mathcal{F})$. The appropriate equivalence classes are similar to those in [4, 5]. For each timer T_i let C_i be the largest constant appearing in a constraint equation involving v_i defining either an enabling or inclusion condition. Let $U_i = max(K_i^2, C_i)$ and $L_i = -K_i^1$. Let $int(\cdot)$ be the integer component of a number and $fract(\cdot)$ the fractional component. Say that $v \approx v'$ if either:

- for some i, j, $v_i < L_i$ and $v'_j < L_j$
- or for all i, $v_i \geq L_i$ and for all i, j
  - either $int(v_i) = int(v'_i)$ or both $v_i > U_i$ and $v'_i > U_i$
  - and whenever $v_i \leq U_i$ and $v_j \leq U_j$
    $fract(v_i) \leq fract(v_j)$ if and only if $fract(v'_i) \leq fract(v'_j)$.

Let $[v]$ indicate the equivalence class containing v and $[\mathbb{R}^N]_{\approx}$ the set of all equivalence classes induced by $\approx$. These equivalence classes will be the timer regions. Note that there are only a finite number of timer regions. The key to constructing an automaton accepting $\mathcal{L}^{seq}(\mathcal{A}, L_0, \mathcal{F})$ is to describe the transition from one timer region to the next. Below, the succession of timer regions is described. Two sorts of transitions are possible for timer regions – an ϵ-transition denoting the passage of time and a σ transition denoting the occurrence of a discrete event. Assume the system is in location l.

- **Time Succession:** Say that $[v'] \neq [v]$ is a time successor of $[v]$ if for some $t > 0$, $v + tr^l \in [v']$,for all $0 \leq t' \leq t$, $v + t'r^l \in [v] \cup [v']$, and $[v'] \subseteq \overline{V^l}$ where $\overline{V^l}$ is the closure of V^l. That is, $[v']$ is the first timer region reachable from any point in $[v]$ by the passage of time.
- **Discrete Transition:** Say that $[v']$ is a σ successor of $[v]$ if there exists an edge $e = (l, \sigma, l', V^e, reset^e)$ such that:
  - $[v] \subseteq V^e$.
  - $\exists v'' \in [v']$ such that:
    - If $v_i := 0 \in reset^e$, then $v''_i = 0$.
    - Else if $v_i := v_i - c \in reset^e$, then $v''_i = v_i - c$.
    - Else $v''_i = v_i$.
  - $[v'] \subseteq V^{l'}$.

Lemma 2. *Given $(\mathcal{A}, L_0, \mathcal{F})$ satisfying the conditions of Theorem 1, the language accepted by the timer region automaton is exactly $\mathcal{L}^{seq}(\mathcal{A}, L_0, \mathcal{F})$.*

Proof. Let $(\mathcal{A}, L_0, \mathcal{F})$ be a suspension automaton satisfying the conditions of Theorem 1.

Suppose we are given a run for ρ with respect to $(\mathcal{A}, L_0, \mathcal{F})$. Note that for this run it is never the case that either $v_i < L_i$ or $v'_i < L_i$. It can be seen that $[v'(n)]$ is reachable from $[v(n)]$ by a series of ϵ-transitions denoting time

succession. Similarly, it can be seen that $[v(n+1)]$ may be reached from $[v'(n)]$ by a $\rho(n+1)$ transition.

Now suppose that we have a run for ρ through the timer region automaton. A run for $(\mathcal{A}, L_0, \mathcal{F})$ may be constructed as follows. Note that the timer region $[v]$ where for some T_i, $v_i < L_i$, is not reachable. Let $\{[v]\}$ be the sequence of timer regions visited. Define a subsequence $\{[v](k_n)\}$ where $[v](k_0) = [v](0)$ and $[v](k_{n+1})$ is the next timer region reached from $[v](k_n)$ after a series of time transitions by a σ-transition (specifically $\rho(n+1)$). Next define the subsequence $\{[v](k'_n)\}$ by $[v](k'_n) = [v](k_{n+1} - 1)$. In other words, $[v](k'_n)$ is the last timer region reached from $[v](k_n)$ by a series of timer transitions and for which $[v](k_{n+1})$ is a $\rho(n+1)$ successor. Let $v(0) \in [v](k_0)$. Recursively construct $\{v\}$ and $\{v'\}$ as follows:

- Given $[v](k_n)$ and $[v](k'_n)$ we can choose a $t(n)$ such that $v'(n) = v(n) + t(n)r^l \in [v](k'_n)$ where r^l is the current rate assignment.
- Given $v'(n)$ applying reset rules will lead to $v(n+1) \in [v](k_{n+1})$.

Now, since $v(n)$ and $v'(n)$ are in the appropriate equivalence classes it is ensured that the enabling and inclusion conditions are satisfied. □

5 Translation of Suspension To Decrementation

In this section, a procedure is given for transforming a suspension based automaton into a decrementation based automaton. Provided that certain conditions are met, the decrementation based automaton is guaranteed to satisfy the conditions of Theorem 1. This procedure is designed with the structure of scheduling, communication, and synchronization policies in mind. As a basic step it requires the definition of a precedence or priority relation among timers as a function of location. In general, the priority assignment can be linked to system concepts such as the notion of task priority.

To gain intuition, consider the evolution of v_2 for the automata of Figs. 1 and 2 in the case that Task 2 is requested at time 8 and Task 1 is requested at time 8.5. This evolution is shown in Fig. 3.

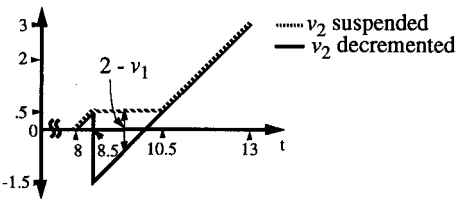


Fig. 3. A comparison of runs

Here, the suspension and decrementation of T_2 is closely linked to the behavior of T_1. That is, between times 8.5 and 10.5 we can think of T_1 as being in an

'active state.' T_2 is suspended while T_1 is in this state. Furthermore, the time T_1 is in this state is predictable (it is the time of service for Task 1) and this predictability may be used to determine the amount v_2 should be decremented by. Finally, note that given v_1, the decremented value of v_2 may be deduced from the suspended value and vice versa.

5.1 Precedence System

The precedence system formalizes the notion of timer behavior and interaction shown in Fig. 3. Suppose we are given an automaton $\mathcal{A}$. Say that a timer is *p_i-valued* for $\mathcal{A}$ if its behavior can be characterized in the following way. The timer has two states – an idle state where its rate and value are both zero, and an active state where its value and rate may be non-zero. The timer transitions from the active to idle state if and only if its value reaches p_i and the timer may not be reset or decremented while in the active state.

A *precedence assignment* for $\mathcal{A}$ is a collection of relations $Prec = \{Pr^l : l \in L\}$ with $Pr^l \subseteq T_\perp \times T$ (where $T_\perp = T \cup \{\perp\}$) denoting the relative priorities of the timers while in location l. $(\perp, T_i)$ is used to indicate timer T_i being of high precedence without there being a timer of lower precedence.

Fig. 4 gives a precedence assignment for the least time remaining policy. Only the top half is given, the bottom half may be defined in a similar fashion. Note that the precedence assignment corresponds to the task priority assignment as it is given in Fig. 1.

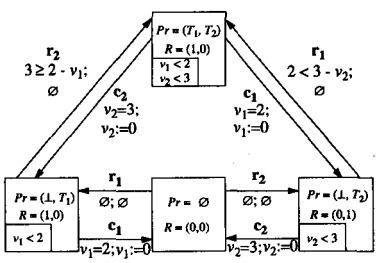


Fig. 4. Precedence for least time remaining policy

A *precedence system* provides a link between precedence assignment and rate assignment for p_i-valued timers. Let $T = T^{\prec} \cup T^{\not\prec}$ where $T^{\prec}$ is the set of timers involved in the precedence assignment. Require that all timers in $T^{\prec}$ be p_i-valued and that timers in $T^{\not\prec}$ are never decremented. Explicitly represent the state of the timers in $T^{\prec}$ as follows. Let $T0 \subseteq T^{\prec}$ be the set of idle timers. Require that a timer leave $T0$ the first time a location l is reached where $(T_i, T_j) \in Pr^l$ and there does not exist $(T_j, T_k) \in Pr^l$. Call the new automaton including the explicit representation of timer state the *precedence automaton*. Say that $Prec$ is *consistent* with R if the following condition holds:

$\forall l \in L$, $r_i^l = 0$ if and only if either $T_i \in T0$ or $\exists (T_i, T_j) \in Pr^l$.

A *precedence system* is a precedence automaton $\mathcal{A}$ where $Prec$ is consistent with R.

In order to define a precedence system for the automaton in Fig. 4 the only thing to do is make $T0$ an explicit part of the location. This is done in Fig. 5. For this automaton $Prec$ is consistent with R. In terms of discrete transitions, this brings the representation closer to that of Fig. 2. However, scheduling is still represented through the suspension of timers.

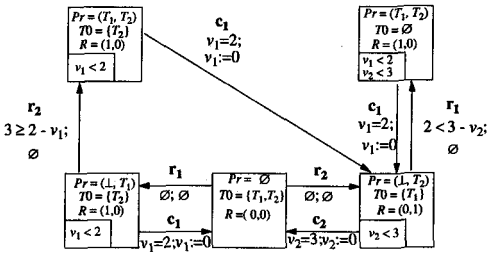


Fig. 5. Precedence system for least time remaining policy

In order to provide the final link between timer behavior and decrementation a few more conditions must be satisfied. Most importantly, it must be possible to identify which timer is suspending which at any time, and to ensure that this suspension relation doesn't change arbitrarily. This may be achieved by restricting the form of $Prec$. Say that $Prec$ is *prioritizing* if for all ρ the following condition holds for any run:

> Given a transition from $(l, T0)$ to $(l', T0')$, if $T_i, T_j \notin T0$ and $(T_i, T_j) \in Pr^l$, then either $(T_i, T_j) \in Pr^{l'}$ or $T_j \in T0'$.

Thus, during the active period of any two timers their precedence relation to each other remains fixed. The prioritizing condition may either be ensured by design or checked as a language property. Say that Pr^l is *tree-like* if whenever $(T_i, T_j) \in Pr^l$ and $(T_i, T_k) \in Pr^l$ then either $(T_j, T_k) \in Pr^l$ or $(T_k, T_j) \in Pr^l$. $Prec$ is tree-like if for all $l \in L$, Pr^l is tree-like. Finally, say that Pr^l is *complete* if whenever $(T_i, T_j) \in Pr^l$ and $(T_j, T_k) \in Pr^l$, then $(T_i, T_k) \in Pr^l$. Note that any precedence assignment may be completed.

It is also necessary to restrict the form of the enabling and inclusion conditions. Say that $V \subseteq \mathbb{R}^N$ is rectangular if there exists a set of constraint equations all of the form $v_i \sim c$ describing it. Note that the precedence system given in Fig. 5 satisfies the tree-like and prioritizing conditions and has rectangular enabling and inclusion regions.

5.2 Translation of Precedence Automaton to Decrementation

Suppose that $\mathcal{A}$ is a precedence system where *Prec* is prioritizing, tree-like, and complete, and for all l and e, V^l and V^e are rectangular. An automaton $Dec(\mathcal{A})$ may be produced as follows.

- **Decrementation:** Let e be an edge transitioning from $(l, T0)$ to $(l', T0')$. Suppose that $T_i \in T0$, but $T_i \notin T0'$. For all $T_j \notin T0'$ such that $(T_j, T_i) \in Pr^{l'}$, add $v_j := v_j - p_i$ to $reset^e$.
- **Rate Assignment:** Set the rate assignment so that timers are rate zero if and only if they are in $T0$.
- **Skewing V^l and V^e:** Given v, $T0$, and Pr^l, define $\overline{v} = skew(v, T0, Pr^l)$ as follows. If $T_i \in T0$ then $\overline{v}_i = 0$. Otherwise, $\overline{v}_i = v_i - \sum_{j \in J_i}(p_j - v_j)$ where $J_i = \{j : (T_i, T_j) \in Pr^l\}$.
  Let $skew(V, T0, Pr^l) = \{\overline{v} : \exists v \in V \text{ s.t. } \overline{v} = skew(v, T0, Pr^l)\}$.
  - For all $l \in L$ replace V^l by $skew(V^l, T0, Pr^l)$.
  - For all $e \in Edge$ replace V^e by $skew(V^e, T0, Pr^l)$.

  For any rectangular V, $skew(V, T0, Pr^l)$ may be represented as follows. For all $T_i \notin T0$ if there exists T_{j_i} such that $(T_i, T_{j_i}) \in Pr^l$ and there does not exist T_j such that $(T_{j_i}, T_j) \in Pr^l$, then replace all constraint equations of the form $v_i \sim c$ by $v_i - v_{j_i} + p_{j_i} \sim c$.

$Dec(\mathcal{A})$ for the precedence system of Fig. 5 is shown in Fig. 6. Two changes are made to $\mathcal{A}$ – in the case that Task 2 is preempted, its process timer is decremented and the inclusion condition is skewed.

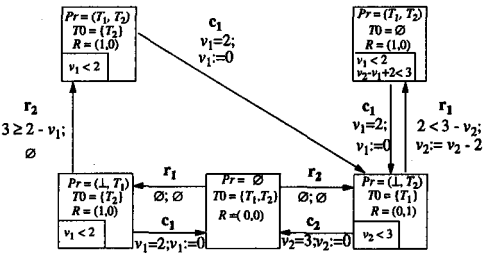


Fig. 6. $Dec(\mathcal{A})$ for least time remaining policy

Theorem 3. *Suppose $\mathcal{A}$ is a precedence system where Prec is prioritizing, tree-like, and complete, and for all l and e, V^l and V^e are rectangular. In this case, $\mathcal{L}^{seq}(\mathcal{A}) = \mathcal{L}^{seq}(Dec(\mathcal{A}))$.*

Proof. Given a run with respect to one of the automata, it is enough to construct the appropriate timer values to prove there exists a corresponding run with respect to the other. Let $\overline{v}$ denote the timer values for $Dec(\mathcal{A})$ and v denote the timer values for $\mathcal{A}$.

Given a run with respect to $\mathcal{A}$, let $\overline{v}(n) = skew(v(n), Pr^{l(n)}, T0(n))$ and $\overline{v}'(n) = skew(v'(n), Pr^{l(n)}, T0(n))$. It is easy to verify that this indeed defines a run with respect to $Dec(\mathcal{A})$.

Similarly, suppose we are given a run with respect to $Dec(\mathcal{A})$. Let $v_i(n) = \overline{v}_i(n) - \overline{v}_{j_i}(n) + p_{j_i}$ where j_i satisfies $(T_i, T_{j_i}) \in Pr^{l(n)}$ and there does not exist T_j such that $(T_{j_i}, T_j) \in Pr^{l(n)}$. Define $v'_i(n)$ in a similar fashion. It can be verified that this defines a run with respect to $\mathcal{A}$. □

Theorem 4. *Suppose $\mathcal{A}$ is a precedence system where Prec is prioritizing, tree-like, and complete, and for all l and e, V^l and V^e are rectangular. In this case, $Dec(\mathcal{A})$ satisfies the conditions of Theorem 1.*

Proof. By the definition of the precedence system and the construction of $Dec(\mathcal{A})$ it is clear that $r_i = 0$ only if $v_i = 0$. Let $J_i = \{j : \exists l \text{ s.t. } (T_i, T_j) \in Pr^l\}$. Then $K_i^1 = \sum_{j \in J_i} p_j$ satisfies the second condition of Theorem 1. $K_i^2 = p_i$ satisfies the final condition of Theorem 1. □

6 Discussion

In this paper, a class of hybrid automata for which language containment is decidable is defined. The conditions presented are somewhat restrictive, but they do allow for the representation of interesting systems that could not be represented using rate 1 automata. Future work should seek to relax these conditions, for instance, a relaxation of the requirement that timers are suspended for a fixed amount of time. In addition, techniques should be explored to minimize the state explosion problem inherent in all the current timer-based hybrid automata.

References

1. R. Grossman, A. Nerode, A Ravn, and H. Rischel, editors. *Hybrid Systems*, volume LNCS 736. Springer, 1993.
2. R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume LNCS 736, pages 209–229. Springer, 1993.
3. D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume LNCS 407. Springer, 1989.
4. R. Alur and D. Dill. Automata for modeling real-time systems. In *Proceedings of the 17th Annual Colloquium on Automata, Languages, and Programming*, 1990.
5. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *IEEE Proceedings of the Fifth Annual Symposium on Logic and Computer Science*, 1990.
6. A. Olivero and S. Yovine. Kronos: A tool for verifying real-time systems. user's guide and reference manual. draft 0.0. Preprint. VERIMAG, B.P. 53X, 38401 Grenoble, France, May 1993.
7. P. Tzounakis. Verification of real time systems: The extension of COSPAN in dense time. Master's thesis, University of Crete, 1992.