2010 17th International Symposium on Temporal Representation and Reasoning

Analysis of Timed Recursive State Machines

Massimo Benerecetti
Università degli studi di Napoli Federico II
Napoli, Italy
bene@na.infn.it

Stefano Minopoli
Università degli studi di Napoli Federico II
Napoli, Italy
minopoli@na.infn.it

Adriano Peron
Università degli studi di Napoli Federico II
Napoli, Italy
peron@na.infn.it

***Abstract*—The paper proposes a temporal extension of Recursive State Machines (RSMs), called Timed RSMs (TRSMs). A TRSM is an indexed collection of Timed Automata allowed to invoke other Timed Automata (procedural calls). The classes of TRSMs are related to an extension of Pushdown Timed Automata, called EPTAs, where an additional stack, coupled with the standard control stack, is used to store temporal valuations of clocks. A number of subclasses of TRSMs and EPTAs are considered and compared through bisimulation of their timed LTSs. It is shown that EPTAs and TRSMs can be used to recognize classes of timed languages exhibiting context-free properties not only in the untimed "control" part, but also in the associated temporal dimension. The reachability problem for both TRSMs and EPTAs is investigated, showing that the problem is undecidable in the general case, but decidable for meaningful subclasses. The complexity is stated for a TRSMs subclass.**

***Keywords*-Recursive State Machines; Timed Automata; Real Time Systems; Context Free Languages.**

I. Introduction

The formalism of *Recursive State Machines* (RSMs) has been introduced in [1] to model control flow in typical sequential imperative programming languages with recursive procedure calls. A RSM is an indexed collection of finite state machines (components). Components enhance the power of ordinary state machines by allowing vertices which correspond to invocations of other components in a potentially recursive manner. As shown in [1], RSMs are closely related to Pushdown Systems (PDSs). While PDSs have been widely studied in the literature within the field of program verification, RSMs seem to be more appropriate for visual modeling. In the context of state-transition formalisms, Timed Automata (TA) [2] are the reference framework to model real time systems. In this paper we consider a real time extension of RSMs (*Timed RSMs* or *TRSMs*), which allows to model real time recursive systems. Roughly speaking, a TRSM is an indexed collection of Timed Automata, with the additional ability of allowing states corresponding to invocations of other timed components. All the timed components refer to a common set of clocks used to constrain the behavior. Within a timed component, the only explicit update of clocks is the standard operation of *clock reset*. In addition, clock updates occur when transitions, corresponding to invocations of other components or returns from components, are performed. In particular, at invocation time one can choose the subset of clocks to reset, and at return time one can choose to restore a subset of clocks to the values they had at invocation time. The idea of extending formalisms which implicitly allow to model recursive systems (e.g., PDSs) with real time features has already been proposed in the literature (e.g., [4], [7]). For instance, [4] proposes the formalism of Pushdown Timed Automata (PTA), which are Timed Automata augmented with a pushdown control stack. However, besides that fact that TRSMs, similarly to RSMs, are more appropriate than PTAs for visual modeling, it seems that PTAs are not expressive enough to account for storing and restoring clock values. Indeed, the control stack of PTAs allows to trace the history of component invocation but cannot be exploited to record the history of clock values stored at invocation time and which are needed at the matching returns for restoration. For instance, the store/restore of clock values in correspondence of invocations and returns allows to model in a very natural way a notion of time local to a component. In other words, TRSMs can manage an evolution of time within a component, abstracting away the elapse of time within the invoked components. Since the number of recursive invocation can be unbounded, it seems that this notion of local time cannot be modeled by PTAs without an unbounded number of clocks. In the paper we also lift to the timed setting the correspondence between RSMs and PDSs, as stated in [1]. Since PTAs are not expressive enough to account for TRSMs, we propose Extended PTAs, which augment PTAs with an additional stack. The additional stack is used to save/restore clock valuations. The two stacks are independent, in the sense that the control stack is used to save control symbols and the valuation stack is used only for storing clock valuations. As a consequence, ETPAs, besides the standard clock reset operations, also allows for store and restore operations on clock valuations. We shall prove

[1]Partially supported by Italian MIUR Project "Integrating automated reasoning in model checking: towards push-button formal verification of large-scale and infinite-state systems."

[2]An extended version of this work is available at the URL http://people.na.infn.it/∼minopoli/TRSM.pdf

1530-1311/10 $26.00 © 2010 IEEE
DOI 10.1109/TIME.2010.10

that TRSMs are equivalent (via weak timed bisimulation) to a syntactic restriction of EPTAs (i.e.: EPTA_2). It is interesting to notice that EPTAs seem to be also a suitable framework to study timed languages exhibiting context-free properties both in the untimed "control" part and in the associated timestamps. For instance, we shall provide an example of a context-free timed language with mirror distribution of symbols and of temporal delays between consecutive symbols.

The main technical contributions of the paper are decidability and complexity results for the reachability problem in TRSMs and EPTAs. In particular, we show that the problem is undecidable for the general classes of TRSMs and EPTAs. However, decidability can be recovered by forcing to restore all the clock values at return time. The class of TRSMs satisfying this restriction is called TRSM_1. The equivalent class of EPTAs, which can be characterized with suitable syntactical restrictions, is denoted by EPTA_1. The complexity of reachability is given for the subclass TRSM_0 of TRSM_1, which further forces to reset all the clock values at invocation time. For this class, the reachability problem is shown to be PSPACE-complete, as in classical TA. The paper also provides a comprehensive picture of the expressive hierarchy and equivalence among TRSMs and EPTAs.

The structure of the paper is as follows. Section 2 defines syntax and the semantics of TRSMs. Section 3 introduces syntax and semantics of EPTAs. Section 4 establishes the correspondence between subclasses of TRSMs and EPTAs. Section 5 is devoted to the decidability and complexity analysis of the reachability problem for TRSMs and EPTAs. For the sake of space, proofs missing in the paper are reported in the complete version available at address http://people.na.infn.it/∼minopoli/TRSM.pdf.

II. Timed Recursive State Machines

In this section we define syntax and semantics of TRSMs. A TRSM is an indexed collection of Timed Automata, with the additional ability of distinguishing ordinary states (nodes) and states corresponding to invocation of other timed components (boxes). We preliminary recall some standard notions of Timed Automata.

A dense *clock* x is a variable over a dense domain $\mathcal{D}^{\geq 0}$ (as usual non negative reals $\mathcal{R}^{\geq 0}$ or rationals $\mathcal{Q}^{\geq 0}$). Let X be a set of dense clocks, a *clock valuation* is a function $\mathbf{v} : X \rightarrow \mathcal{D}^{\geq 0}$. For a set of clocks X, the set of *clock constraints* $\mathcal{C}(X)$ is defined by the following grammar:

$$\varphi{:=}x \sim c \mid x - y \sim c \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

where $x, y \in X$, $c \in \mathcal{Q}^{\geq 0}$ and $\sim \in \{<, \leq, =, \neq, \geq, >\}$.

Following [2], we write $\mathbf{v} \in \varphi$ when the clock valuation $\mathbf{v}$ satisfies the clock constraint $\varphi \in \mathcal{C}(X)$. The notion of constraint satisfaction by a clock valuation is defined in the standard way (e.g., see [2]). For $t \in \mathcal{D}^{\geq 0}$, $\mathbf{v}{+}t$ denotes the valuation $\mathbf{v}'$ such that $v'(x){=}v(x) + t$, for all $x \in X$ (*clock progress*). Given $r \subseteq X$, $\mathbf{v}\downarrow_r$ denotes the valuation resulting from the reset of the clocks in r, namely:

$$(v \downarrow_r)(i) = \begin{cases} 0 & \text{if } i \in r \\ v(i) & \text{otherwise.} \end{cases}$$

Moreover, given the valuations $\mathbf{v}, \overline{\mathbf{v}}$ and a set $r \subseteq X$, $\mathbf{v}\uparrow_{(r,\overline{\mathbf{v}})}$ denotes the valuation resulting from the update of the clocks in r, according to $\overline{\mathbf{v}}$, namely:

$$(v \uparrow_{(r,\overline{\mathbf{v}})})(i) = \begin{cases} \overline{\mathbf{v}}(i) & \text{if } i \in r \\ v(i) & \text{otherwise.} \end{cases}$$

We can now define Timed Recursive State Machines (TRSMs). A TRSM is an indexed collection of components which share a set of clocks X. The set of states of each component is partitioned into a set of nodes and a set of boxes. Boxes correspond to invocation and are associated with an index of a component. There are four kinds of transitions: (i) *internal transitions* connecting nodes of the same component; (ii) *call transitions* which lead to a box and an entry node of the called component; (iii) *return transitions* which lead from a box and an exit node of the callee to a node of the caller; (iv) *return-and-call transitions* which combines a return and a call transitions. Transitions are decorated with clock constraints as in TA. Internal transitions can only reset clocks as in TA. Call transitions can reset a subset of clocks at invocation time. Return transitions can update a subset of clocks by restoring the values they had at invocation time, and reset a possibly different subset of clocks.

Definition *1:* For an alphabet Σ, a TRSM is a tuple $\langle A_1, \ldots, A_n, X\rangle$ where, for all $1 \leq i \leq n$, A_i is a *component* and X is a finite set of clocks. A component A_i is $\langle N_i \cup B_i, Y_i, En_i, Ex_i, \delta_i\rangle$, where:

- N_i and B_i are the disjoint sets of nodes and *boxes*, respectively;
- $Y_i : B_i \rightarrow \{1,\ldots,n\}$ assigns to every box the index of a component;
- $En_i \subseteq N_i$, $Ex_i \subseteq N_i$ are the sets of *entry* and *exit* nodes, respectively;
- $\delta_i \subseteq (N_i \cup Retns_i) \times \Sigma \cup \{\tau\} \times C(X) \times 2^X \times 2^X \times (N_i \cup Calls_i)$ is the transition relation, where $Calls_i$=$\{(b,en)$: $b \in B_i$, $en \in En_{Y_i(b)}\}$, $Retns_i$=$\{(b,ex)$: $b \in B_i$, $ex \in Ex_{Y_i(b)}\}$.
  For each transition $\langle u_1,\sigma,\varphi,r_1,r_2,u_2\rangle \in \delta_i$:
  - u_1 (resp.: u_2) are the source (resp.: target) of the transition;
  - $\sigma \in \Sigma \cup \{\tau\}$ is an input symbol or the silent action;
  - $\varphi \in \mathcal{C}(X)$ is a constraint over the clocks in X;
  - r_1 (resp.: r_2) is the subset of clocks to reset (resp.: restore).

We use the following abbreviations: $N = \bigcup_i N_i$, $Calls = \bigcup_i Calls_i$, $Retns = \bigcup_i Retns_i$, $En = \bigcup_i En_i$ and $Ex =$

$\bigcup_i Ex_i$. In order to avoid confusion with the empty stack symbol, we use the symbol τ to represent the internal silent action (usually called ϵ-transition). We assume that only return and return-and-call transitions can have a non empty set r_2 of restored clocks.

A *global state* of a TRSM $\mathcal{T}$=$\langle A_1, \ldots, A_n, X\rangle$ is a tuple gs=$\langle b_1, \ldots, b_s, u, \mathbf{v}_1, \ldots, \mathbf{v}_s, \mathbf{v}\rangle$, where $b_1, \ldots, b_s$ are boxes, u is a node and $\mathbf{v}_1, \ldots, \mathbf{v}_s, \mathbf{v}$ are clock valuations. Intuitively, $b_1, \ldots, b_s$ represent the history of the invoked components, $\mathbf{v}_1, \ldots, \mathbf{v}_s$ are the corresponding valuations stored at invocation time, u is the current node, and $\mathbf{v}$ is the current clock valuation. A global state gs=$\langle b_1, \ldots, b_s, u, \mathbf{v}_1, \ldots, \mathbf{v}_s, \mathbf{v}\rangle$ such that $b_i \in B_{j_i}$, for $1 \leq i \leq r$ and $u \in N_j$ is *well-formed* if $Y_{j_i}(b_i)$=j_{i+1} for $1 \leq i < r$ and $Y_{j_s}(b_s)$=j. The set of well-defined global states is denoted by GS.

The semantics of a TRSM over the alphabet Σ is given by the timed Labelled Transition System (timed LTS) $\langle GS, GS_0, \Sigma_\tau^{\mathcal{D}^{\geq 0}}, \Delta\rangle$, where:

- $GS_0 \subseteq GS$ is the set of states of the form $\langle \epsilon, u, \mathbf{v}_0\rangle$, with $u \in En_1$ and $v_0(x)$=0, for all $x \in X$;
- $\Sigma_\tau^{\mathcal{D}^{\geq 0}}$ is the set $\Sigma \cup \{\tau\} \cup \{\tau(t) | t \in \mathcal{D}^{\geq 0}\}$, where Σ and $\mathcal{D}^{\geq 0}$ are disjoint;
- $\Delta \subseteq GS \times \Sigma_\tau^{\mathcal{D}^{\geq 0}} \times GS$ is the transition relation. For gs=$\langle b_1, \ldots, b_s, u, \mathbf{v}_1, \ldots, \mathbf{v}_s, \mathbf{v}\rangle$, with $u \in N_j$ and $b_s \in B_m$, $\langle gs,\sigma,gs'\rangle \in \Delta$ whenever one of the following holds:
  1) **progress transition**: $\sigma = \tau(t)$ and $gs' = \langle b_1, \ldots, b_s, u, \mathbf{v}_1, \ldots, \mathbf{v}_s, \mathbf{v}'\rangle$, with $\mathbf{v}' = \mathbf{v} + t$ and $t \in \mathcal{D}^{\geq 0}$;
  2) **reset transition**: if $\langle u,\sigma,\varphi,r_1,\emptyset,u'\rangle \in \delta_j$, with $u' \in N_j$, and $\mathbf{v} \in \varphi$, then $gs' = \langle b_1, \ldots, b_s, u', \mathbf{v}_1, \ldots, \mathbf{v}_s, \mathbf{v}'\rangle$, with $\mathbf{v}' = \mathbf{v}\downarrow_{r_1}$;
  3) **call transition**: if $\langle u,\sigma,\varphi,r_1,\emptyset,(b',en)\rangle \in \delta_j$, with $b' \in B_j$, and $\mathbf{v} \in \varphi$, then $gs' = \langle b_1, \ldots, b_s, b', en, \mathbf{v}_1, \ldots, \mathbf{v}_s, \mathbf{v}, \mathbf{v}'\rangle$, with $\mathbf{v}' = \mathbf{v}\downarrow_{r_1}$;
  4) **return transition**: if $\langle (b_s, u),\sigma,\varphi,r_1,r_2,u'\rangle \in \delta_m$, with $u' \in N_m$, $u \in Ex_j$, and $\mathbf{v} \in \varphi$, then $gs' = \langle b_1, \ldots, b_{s-1}, u', \mathbf{v}_1, \ldots, \mathbf{v}_{s-1}, \mathbf{v}'\rangle$, with $\mathbf{v}' = \mathbf{v}\uparrow_{(r_2,\mathbf{v}_s)}\downarrow_{r_1}$;
  5) **return-and-call transition**: if $\langle (b_s, u),\sigma,\varphi,r_1,r_2,(b', en)\rangle \in \delta_m$, with $b' \in B_m$, $u \in Ex_j$ and $\mathbf{v} \in \varphi$, then $gs' = \langle b_1, \ldots, b_{s-1}, b', en, \mathbf{v}_1, \ldots, \mathbf{v}_{s-1}, \mathbf{v}'_s, \mathbf{v}'\rangle$, with $\mathbf{v}'_s = \mathbf{v}\uparrow_{(r_2,\mathbf{v}_s)}$ and $\mathbf{v}' = \mathbf{v}'_s\downarrow_{r_1}$.

A progress transition occurs when the control remains in the same vertex and there is only a clock progress. A reset transition occurs when there is an internal transition inside a component, which can possibly reset a subset of clocks. A call transition occurs when a box is entered and a component is invoked. In this case the current clock valuation is stored and a subset of clocks are possibly reset. A return transition occurs when the control returns to the invoking component. In this case, the valuation at invocation time may be restored for that subset of clocks specified in the restore set decorating the transition, the constraint must be satisfied by the current clock valuation and, finally, some clocks may be reset. A return-and-call transition combines the effect of return and call in a single step. Notice that the above timed LTS, besides symbols in Σ, also has symbols representing the elapse of time, and satisfies the usual properties of temporal determinism, time additivity and 0-delay as defined in [3].

A run $\lambda_\mathcal{T}$ of $\mathcal{T}$ is a (possibly infinite) path in the LTS $\langle GS, GS_0, \Sigma_\tau^{\mathcal{D}^{\geq 0}}, \Delta\rangle$ for $\mathcal{T}$ having the form $\lambda_\mathcal{T}$=$gs_0 \stackrel{\sigma_0}{\rightarrow} gs_1 \stackrel{\sigma_1}{\rightarrow} gs_2 \ldots gs_i \stackrel{\sigma_i}{\rightarrow} \ldots$, where $gs_i \stackrel{\sigma_i}{\rightarrow} gs_{i+1} \in \Delta$, for all $i \geq 0$. It is an *initial* path, if $gs_0 \in S_0$. By $\Lambda_\mathcal{T}$ we denote the set of all paths of $\mathcal{T}$. For any two $gs_i, gs_j \in GS$, gs_j is *reachable* from gs_i, written $gs_i \rightarrow^* gs_j$, if there is a run $\lambda_\mathcal{T}$=$gs_i \stackrel{\sigma_i}{\rightarrow} gs_{i+1} \stackrel{\sigma_{i+1}}{\rightarrow} gs_{i+2} \ldots gs_{j-1} \stackrel{\sigma_{j-1}}{\rightarrow} gs_j$.

Notice that an RSM can be seen as a TRSM with an empty set of clocks, and where the only clock constraint is $True$. From the semantic point of view, the clock valuations occurring in a global state are meaningless, and progress transitions are omitted in the LTS.

Example *1:* In Fig. 1 we show a TRSM whose behavior is triggered by a timed sequence of symbols having the form $(a,t_1)\ldots(a,t_n)(b,t_{n+1})\ldots(b,t_{2n})$, such that $\Delta_{2n-i+1}+\Delta_i \leq k$, with k a fixed constant value, and, taking $t_0 = 0$, $\Delta_i = t_i - t_{i-1}$. Notice that the untimed part of the sequence above describes a context-free language (i.e., L=$\{a^n b^n : n \geq 1\}$).

The TRSM consists of a single component, which may be invoked recursively. The local behavior inside a component consists of a transition triggered by a symbol a followed by a symbol b. The transition triggered by b may be delayed by a recursive invocation. The component uses the clocks x and y: the first is reset at invocation time while the second is reset when the symbol b occurs. Both clocks are restored at return. The enforced requirement is, therefore, that the local behavior is performed within time k, abstracting away the time possibly spent by the recursive computation. It is meaningful to observe that even though a PTA can model invocations and returns by using its control stack, it could not guarantee the temporal requirement above, since, in order to check $\Delta_{2n-i+1} + \Delta_i \leq k$, it would require an unbounded number of clocks, one for each recursive call.

In the following we show a run over the timed word

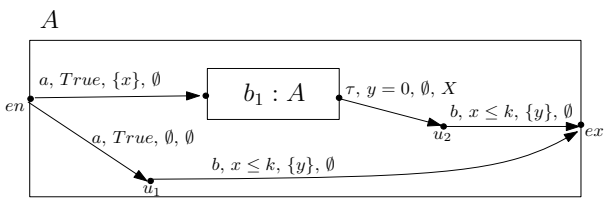

Figure 1. An example of TRSM

$(a,1)(a,3)(a,7)(b,8)(b,11)(b,15)$, where $\Delta_1 + \Delta_6 = k$, $\Delta_2 + \Delta_5 = k$ and $\Delta_3 + \Delta_4 = k$, with $k = 5$, according to the definition above:

$\langle en,(0,0)\rangle \xrightarrow{\Delta_1=1} \langle en,(1,1)\rangle \xrightarrow{a} \langle b,en,(1,1)(0,1)\rangle \xrightarrow{\Delta_2=2}$
$\langle b,en,(1,1)(2,3)\rangle \xrightarrow{a} \langle b,b,en,(1,1)(2,3)(0,3)\rangle \xrightarrow{\Delta_3=4}$
$\langle b,b,en,(1,1)(2,3)(4,7)\rangle \xrightarrow{a} \langle b,b,u_1,(1,1)(2,3)(4,7)\rangle \xrightarrow{\Delta_4=1}$
$\langle b,b,u_1,(1,1)(2,3)(5,8)\rangle \xrightarrow{b} \langle b,b,ex,(1,1)(2,3)(5,0)\rangle \xrightarrow{\tau}$
$\langle b,u_2,(1,1)(2,3)\rangle \xrightarrow{\Delta_5=3} \langle b,u_2,(1,1)(5,6)\rangle \xrightarrow{b} \langle b,ex,(1,1)(5,0)\rangle$
$\xrightarrow{\tau} \langle u_2,(1,1)\rangle \xrightarrow{\Delta_6=4} \langle u_2,(5,5)\rangle \xrightarrow{b} \langle ex,(5,0)\rangle$

We introduce now two subclasses of TRSMs obtained by suitably constraining the ability of resetting and restoring clocks associated with transitions. These subclasses will be compared in Sections 3 and 4 with respect to expressive power and decidability properties.

TRSM_1: the class of TRSMs, where return and return-and-call transitions are of the form $\langle (b,\ ex),\ \sigma,\ \varphi,\ r_1,\ X,\ u_2\rangle$, with $u_2 \in N \cup Calls$ (the whole set of clocks is restored at every return);

TRSM_0: the class of TRSM_1, where call and return-and-call transitions are of the form $\langle u_1,\ \sigma,\ \varphi,\ X,\ \emptyset,\ (b,\ en)\rangle$ and $\langle (b',ex),\ \sigma,\ \varphi,\ X,\ X,\ (b,\ en)\rangle$, respectively (the whole set of clocks is reset at every call).

Unlike the general TRSMs class, where subsets of clock values can be restored, in the TRSM_1 and TRSM_0 subclasses the entire set of clocks are restored at return time from the invoked component. Therefore, the clock valuations at return time are the same clock valuations at invocation time. This allows to model in a very natural way a notion of time local to a component, abstracting away the elapse of time within an invoked component. In the TRSM_0 class this abstraction is even stronger, because all clocks are reset at invocation time. Notice that, despite the fact that TRSM_1 and TRSM_0 abstract away the time spent into the invoked components, it does not seem possible to model them with PTAs. Indeed, since the number of recursive invocation can be unbounded, an unbounded number of clocks would be necessary.

Notice that the TRSM of Example 1 belongs to the class TRSM_1.

III. Extended Pushdown Timed Automata

In this section we introduce syntax and semantics of *Extended Pushdown Timed Automata* (EPTAs). An EPTA is a Pushdown Automaton enriched with a set of clocks and with an additional stack used to store/restore clock valuations. [1]

Definition *2:* An EPTA $\mathcal{P}$ over $\Sigma \cup \{\tau\}$ is a tuple $\langle Q,\ q_0,\ X,\ \Gamma,\ T\rangle$, where:

- Q is a finite set of states and $q_0 \in Q$ is the initial state;
- X is a finite set of clocks and Γ is a finite stack alphabet;
- $T \subseteq Q \times \Sigma \cup \{\tau\} \times \Gamma \cup \{\epsilon\} \times (\{\epsilon\} \cup \Gamma \cup \Gamma^2) \times \mathcal{C}(X) \times 2^X \times Op \times Q$ is the transition relation, with Op=$\{Reset, Store, Restore\}$.

Intuitively, for each transition of the form $\langle q_1,\ \sigma,\ \gamma_1,\ \gamma_2,\ \varphi,\ r,\ op,\ q_2\rangle \in T$, q_1 (resp.: q_2) is the source (resp.: the target) state; σ is the input symbol; γ_1 is the symbol on top of the control stack (ϵ denotes the empty stack); γ_2 is the string (of length at most 2) which replaces the symbol on top of the control stack; $\varphi \in C(X)$ is a clock constraint; op is the operation requested on the set r of clocks. If $op \in \{Reset, Store\}$, then r indicates the set of clocks to reset. In addition, if $op = Store$, the current clock valuation is stored on top of the valuation stack. If $op = Restore$, r indicates the set of clocks to restore, using the clock valuation on top of the valuation stack.

A configuration of an EPTA $\mathcal{P}$ is a tuple $\langle q,\mathbf{v},w,\mathbf{d}\rangle$, where $q \in Q$ is the current control state, $\mathbf{v}$ is the current clock valuation, $w \in \Gamma^*$ is the content of the control stack, and $\mathbf{d} \in (X \rightarrow \mathcal{D}^{\geq 0})^*$ is the content of the valuations stack. The set of all configurations is denoted by GC.

The semantics of EPTAs is given by a timed LTS $\langle GC,\ GC_0,\ \Sigma_\tau^{\mathcal{D}^{\geq 0}},\ \Delta\rangle$, where:

- GC_0 is $\{\langle q_0,\mathbf{v}_0,\epsilon,\epsilon\rangle\}$, with $v_0(x)$= 0, for every $x \in X$;
- $\Delta \subseteq GC \times \Sigma_\tau^{\mathcal{D}^{\geq 0}} \times GC$ is the transition relation. For $gc = \langle q,\mathbf{v},w,\mathbf{v}_s \cdot \mathbf{d}\rangle$, $\langle gc,\sigma,gc'\rangle \in \Delta$ whenever one of the following holds:
  1) **progress transition**: $\sigma = \tau(t)$ and $gc' = \langle q,\mathbf{v}',w,\mathbf{v}_s \cdot \mathbf{d}\rangle$, with $\mathbf{v}' = \mathbf{v} + t$ and $t \in \mathcal{D}^{\geq 0}$;
  2) **reset transition**: if $\langle q,\ \sigma,\ a,\ \gamma,\ \varphi,\ r,\ Reset,\ q'\rangle \in T$, with $\mathbf{v} \in \varphi$ and $w = a \cdot w''$, then $gc' = \langle q',\mathbf{v}',\gamma w'',\mathbf{v}_s \cdot \mathbf{d}\rangle$ with $\mathbf{v}' = \mathbf{v} \downarrow_r$;
  3) **store transition**: if $\langle q,\ \sigma,\ a,\ \gamma,\ \varphi,\ r,\ Store,\ q'\rangle \in T$, with $\mathbf{v} \in \varphi$ and $w = a \cdot w''$, then $gc' = \langle q',\mathbf{v}',\gamma w'',\mathbf{v} \cdot \mathbf{v}_s \cdot \mathbf{d}\rangle$ with $\mathbf{v}' = \mathbf{v} \downarrow_r$;
  4) **restore transition**: if $\langle q,\ \sigma,\ a,\ \gamma,\ \varphi,\ r,\ Restore,\ q'\rangle \in T$, with $\mathbf{v} \in \varphi$, $w = a \cdot w''$, then $gc' = \langle q',\mathbf{v}',\gamma w'',\mathbf{d}\rangle$, with $\mathbf{v}' = \mathbf{v} \uparrow_{r,\mathbf{v}_s}$.

When a progress transition occurs, there is only a clock progress; the control remains in the same state and the two stacks are left unchanged. A reset transition pops a symbol from the control stack, pushes a string of symbols on the control stack, and resets a set of clocks, while the valuation stack is left unchanged. A store transition behaves as a reset transition, except that, in addition, it pushes the current clock valuation on the valuation stack. A restore transition pops a control symbol from the stack, pushes a string on the control stack, and pops a valuation from the valuation stack, restoring a subset of the clock values according to the popped clock valuation. The notions of (initial) run and reachability among pairs of states can be defined exactly

[1] Actually an EPTA is a PTA (see [4]) for a syntactical definition where the control stack is coupled with a second stack for the clock valuations.

as in the case of TRSMs. $\Lambda_{\mathcal{P}}$ denotes the set of runs and $gc \rightarrow^*_{\mathcal{P}} gc'$ denotes a pair of reachable states.

Example *2:* In Fig. 2 we show an EPTA whose behavior is triggered by a timed sequence of symbols having the form: $(a,t_1)\ldots(a,t_n)(b,t_{n+1})\ldots(b,t_{2n})$ such that $\Delta_{2n-i}=\Delta_i$, with $\Delta_i = t_{i+1} - t_i$. Notice that the untimed part of the sequence above describes a context-free language (i.e. $L=\{a^n b^n : n > 1\}$). As for the timed part, the sequence is required to satisfy a mirror distribution of the delays between consecutive symbols. Notice that, the timed language above exhibits a context-free property both in the untimed part and in the temporal sequence of timestamps. This shows the main difference with respect to PTAs ([4]) where there is no mean to check context free properties on times.

In the following we show a run over the timed word $(a,1)(a,3)(b,6)(b,8)$, where $\Delta_1 = \Delta_3 = 2$ and $\Delta_3 = \Delta_3 = 3$, according to the definition above:

$\langle q_0,(0,0,0),\epsilon,\epsilon\rangle \xrightarrow{\Delta_0=1} \langle q_0,(1,1,1),\epsilon,\epsilon\rangle \xrightarrow{a} \langle q_1,(0,1,1),\gamma_1,$
$(1,1,1)\rangle \xrightarrow{\Delta_1=2} \langle q_1,(2,3,3),\gamma_1,(1,1,1)\rangle \xrightarrow{a} \langle q_1,(0,3,3),\gamma_2\gamma_1,$
$(2,3,3)(1,1,1)\rangle \xrightarrow{\Delta_2=3} \langle q_1,(3,6,6),\gamma_2\gamma_1,(2,3,3)(1,1,1)\rangle \xrightarrow{b}$
$\langle q_2,(3,0,6),\gamma_2\gamma_1,(2,3,3)(1,1,1)\rangle \xrightarrow{\Delta_3=2} \langle q_2,(5,2,8),\gamma_2\gamma_1,$
$(2,3,3)(1,1,1)\rangle \xrightarrow{b} \langle q_3,(5,2,0),\gamma_2\gamma_1,(2,3,3)(1,1,1)\rangle \xrightarrow{\tau}$
$\langle q_4,(2,0,0),\gamma_1,(1,1,1)\rangle \xrightarrow{\tau} \langle q_5,(1,1,1),\epsilon,\epsilon\rangle$

As in the case of TRSMs, we introduce three subclasses of EPTAs by suitably constraining the operations associated with transitions. These subclasses will be naturally related with the subclasses of the TRSMs defined in the previous section. The restrictions tightly couple the type of operations performed on the control stack and on the valuation stack.

EPTA _2 is the subclass of EPTAs where:

- reset transitions have the form $\langle q_1, \sigma, a_1, a_2, \varphi, r, Reset, q_2\rangle$, with $a_1, a_2 \in \Gamma$ (only a swap on the top of the control stack is allowed);
- store transitions have the form $\langle q_1, \sigma, a, \gamma, \varphi, r, Store, q_2\rangle$ with $a \in \Gamma \cup \{\epsilon\}$ and $\gamma \in \Gamma^2$ (only a swap followed by a push is allowed on the control stack);
- restore transitions have the form $\langle q_1, \sigma, a, \epsilon, \varphi, r, Restore, q_2\rangle$, with $a \in \Gamma$ (only a pop operation is allowed on the control stack);

EPTA _1 is the subclass of EPTA_2 where:

- restore transitions have the form $\langle q_1, \sigma, a, \epsilon, \varphi, X, Restore, q_2\rangle$, with $a \in \Gamma$ (the whole set of clocks is restored);

EPTA _0 is the subclass of EPTA_1 where:

- store transitions have the form $\langle q_1, \sigma, a, \gamma, \varphi, X, Store, q_2\rangle$, with $a \in \Gamma \cup \{\epsilon\}$, $\gamma \in \Gamma^2$ (the whole set of clocks is reset);

Notice that the EPTA of the Example 2 belongs to the class EPTA_2.

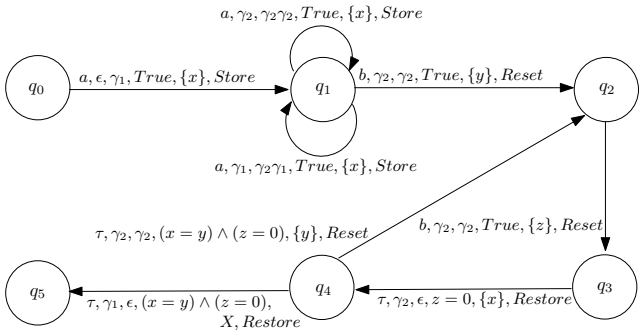

Figure 2. An example of EPTA

IV. Relationship between TRSMs and EPTAs

In this section we investigate the relationship between TRSMs and EPTAs and their subclasses. In order to show such a correspondence, we shall use the standard notion of (timed) *weak bisimulation* between timed LTS. The general picture is reported in Table I.

For the definition of weak bisimilarity for timed LTSs we follow the approach in [3]. Notice that bisimulation for Timed LTS is a nontrivial generalization of the untimed case, and requires some additional technicalities. Formally an LTS $\mathcal{L}=\langle S,S_0,\Sigma,\Delta\rangle$ *simulates* a transition system $\mathcal{L}'=\langle S',S'_0,\Sigma,\Delta'\rangle$ if there exists a *simulation relation* $\prec \subseteq S \times S'$ defined as follows:

- initialization: for all $s_0 \in S_0$, there is a $s'_0 \in S'_0$, with $s_0 \prec s'_0$;
- propagation: for all $\sigma \in \Sigma$, if $s_1 \prec s'_1$ and $s_1 \xrightarrow{\sigma} s_2$, then there exists $s'_2 \in S'$ such that $s'_1 \xrightarrow{\sigma} s'_2$ and $s_2 \prec s'_2$.

If the relation $\prec^{-1}$, defined as $x \prec^{-1} y \Leftrightarrow y \prec x$, is also a simulation relation, then $\prec$ is a called a *bisimulation relation*. Two LTSs $\mathcal{L}$ and $\mathcal{L}'$ are strongly bisimilar, written $\mathcal{L} \equiv_s \mathcal{L}'$, if there is a bisimulation relation between $\mathcal{L}$ and $\mathcal{L}'$.

In order to introduce a notion of weak bisimulation we suitably transform a timed transition system by abstracting sequences of τ-transitions. For a timed LTS $\mathcal{L}$, a *delay execution* is a run of the form $s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} s_2 \ldots \xrightarrow{\sigma_n} s_n$, such that, for every $1 \leq i \leq n$, either $\sigma_i = \tau$ or $\sigma_i = \tau(t_i)$, for some $t_i \in \mathcal{D}^{\geq 0}$. The *abstract transition system* associated to $\mathcal{L}$ is $\mathcal{L}_{abs} = \langle S, S_0, \Sigma_\tau^{\mathcal{D}^{\geq 0}} \setminus \{\tau\}, \Rightarrow\rangle$, where:

- $s \overset{a}{\Rightarrow} s'$ if $a \in \Sigma$ and there exists $s'' \in S$, $s \xrightarrow{\tau}{}^* s'' \xrightarrow{a} s'$;
- $s \overset{\tau(t)}{\Rightarrow} s'$ if there exists a delay execution $s = s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} s_2 \ldots \xrightarrow{\sigma_n} s_n = s'$ such that $t = \sum\{t_i \mid \sigma_i = \tau(t_i)\}$.

Table I
Expressiveness relations among TRSMs and EPTAs.

EPTA_0	$\subseteq$	EPTA_1	$\subset$	EPTA_2	$\subseteq$	EPTA
$\equiv_w$		$\equiv_w$		$\equiv_w$		
TRSM_0	$\subseteq$	TRSM_1	$\subset$	TRSM		

The relation $\xrightarrow{\tau}{}^*$ denotes the reflexive and transitive closure of $\xrightarrow{\tau}$. The transition system $\mathcal{L}_{abs}$ abstracts all the silent actions of $\mathcal{L}$. The relation $\xrightarrow{\tau}{}^*$ thus corresponds to $\overset{\tau(0)}{\Rightarrow}$. Notice also that the relation $\overset{a}{\Rightarrow}$ only abstracts silent actions that can be done before a. Of course, if $\mathcal{L}$ is the timed LTS of a TRSM or of a EPTA without silent actions, then $\mathcal{L}$ and $\mathcal{L}_{abs}$ are identical.

Two LTS $\mathcal{L}$ and $\mathcal{L}'$ are weakly bisimilar, written $\mathcal{L} \equiv_w \mathcal{L}'$, if there exists a bisimulation relation between $\mathcal{L}_{abs}$ and $\mathcal{L}'_{abs}$.

The following theorem shows the tight corresponding between TRSMs and EPTAs and their subclasses (proofs can be found in the complete version of the paper available at address http://people.na.infn.it/∼minopoli/TRSM.pdf).

Theorem *1:* For any TRSM (resp.: TRSM_1, TRSM_0) $\mathcal{T}$, there exists a weakly similar EPTA_2 (resp.: EPTA_1, EPTA_0) $\mathcal{P}$ and vice versa.

V. The reachability problem: decidability and complexity results

In this section we study the problem of reachability for TRSMs and EPTAs. In particular, we prove that the problem is undecidable for the general class of TRSMs and EPTAs, but that is decidable for TRSM_1 and EPTA_1. We state also the complexity of the problem for the class of TRSM_0 (and EPTA_0).

Given a global state $gs = \langle b_1, \ldots, b_s, u, \mathbf{v}_1, \ldots, \mathbf{v}_s, \mathbf{v}\rangle$ of a TRSM $\mathcal{T}$, we call the tuple $\langle b_1, \ldots, b_s, u\rangle$ an *untimed global state* of $\mathcal{T}$. The *reachability problem* for an untimed global state $\langle b_1, \ldots, b_s, u\rangle$ is to determine whether, for some clock valuations $\mathbf{v}_1$, …, $\mathbf{v}_s$, $\mathbf{v}$, the global state $\langle b_1$, …, b_s, u, $\mathbf{v}_1$, …, $\mathbf{v}_s$, $\mathbf{v}\rangle$ is reachable from an initial configuration state. A similar notion of reachability can be given for EPTAs.

The undecidability for TRSMs is stated by showing the undecidability of the reachability problem for the class EPTA_2. In fact, it is possible to show (see the complete work) that EPTA_2s allow to simulate increment and decrement of clocks. For instance, Fig. 3 shows an EPTA_2 which decrements the clock z, leaving the clock y unchanged (z and y can be viewed as the two counters). The path leading from $(p, q_{0\_})$ to $(p, q_{3\_})$, is used to store the value of z into x'', while y maintains the original value. In the next step, these values are stored in the valuation stack, and after an amount of time equal to $x'' - 1$ is elapsed, the original value of y is restored. In this way, it is possible to define an EPTA_2 which simulates a two counter machine (clocks with increment and decrement can be used as counters). This allows to reduce the halting problem of two counter Minsky machine, which is known to be undecidable [5], to the reachability problem for an EPTA_2 (proofs can be found in the complete version of the paper available at address http://people.na.infn.it/∼minopoli/TRSM.pdf).

Theorem *2:* The reachability problem for EPTA_2 is undecidable.

As consequence of Theorem 1 we have the following corollary.

Corollary *1:* The reachability problem for TRSMs is undecidable.

We shall now prove that the reachability problem for TRSM_1 is decidable. The proof combines two techniques: the former is the standard regionalization technique, used to prove reachability in Timed Automata [2], while the latter is derived from the algorithm, based on a fix point construction, proposed in [1] to solve the reachability problem for RSMs.

Let us first recall the notion of clock region and region automaton of a Timed Automaton. Following the standard construction [6], we assume that constants occurring in the clock constraints of the automaton are integers. For any $t \in \mathcal{D}^{\geq 0}$, $\langle t\rangle$ denotes the fractional part of t, and $\lfloor t \rfloor$ denotes the integral part of t (i.e. t=$\lfloor t \rfloor$+$\langle t\rangle$). For each clock $x \in X$, let c_x be the largest integer constant c such that x is compared with c in some clock constraint appearing in a transition. The equivalence relation $\cong$, (*region equivalence*), is defined over the set of clock valuations for X. For two clock valuations $\mathbf{v}_1$ and $\mathbf{v}_2$, we write $\mathbf{v}_1 \cong \mathbf{v}_2$ iff the following conditions hold:

1) for all clocks $x \in X$, either $\lfloor v_1(x) \rfloor$ and $\lfloor v_2(x) \rfloor$ are the same, or both $v_1(x)$ and $v_2(x)$ exceed c_x;
2) for all clocks x, y with $v_1(x) \leq c_x$ and $v_1(y) \leq c_y$, $\langle v_1(x)\rangle \leq \langle v_1(y)\rangle$ iff $\langle v_2(x)\rangle \leq \langle v_2(y)\rangle$;
3) for all clocks $x \in X$ with $v_1(x) \leq c_x$, $\langle v_1(x)\rangle$ = 0 iff $\langle v_2(x)\rangle$ = 0.

A *clock region* is an equivalence class of clock valuations induced by $\cong$. If k is the number of the clocks, there are at most $k! \cdot 4^k \cdot \prod_{x \in X}(c_x + 1)$ regions (see [6]). We denote by Reg the set of all regions with respect to the set of clocks X and an indexed family $\{c_x\}_{x \in X}$.

A clock region reg' is a *time successor* of a clock region reg if and only if, for all clock valuations $\mathbf{v} \in reg$, there exists a $t \in \mathcal{D}^{\geq 0}$ such that $\mathbf{v}$+$t \in reg'$. Let $\varphi \in \mathcal{C}(X)$ be a clock constraint, we write $reg \in \varphi$ if and only if $\mathbf{v} \in \varphi$, for all $\mathbf{v} \in reg$. Note that for a clock constraint φ of a TA, if $\mathbf{v} \cong \mathbf{v}'$, then $\mathbf{v} \in \varphi$ iff $\mathbf{v}' \in \varphi$. Let $r \subseteq X$ a set of clocks, $reg \downarrow_r = \{\mathbf{v} \downarrow_r | \mathbf{v} \in reg\}$ denotes the region resulting from

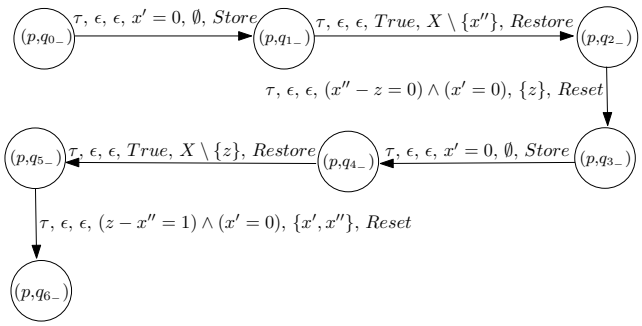

Figure 3. EPTA for the decrement of the clock z

reg by resetting the clocks in r.

It is well known that the reachability problem of a Timed Automaton can be reduced to the reachability problem over its region automaton, namely the automaton whose states are obtained by coupling control states and regions, and transitions are obtained by suitably coupling the transition relation of the timed automaton and the successor relation defined on regions.

Given a TRSM $\mathcal{T}$=$\langle A_1, \ldots, A_n, X\rangle$ belonging to TRSM_1, with $A_i = \langle N_i \cup B_i, Y_i, En_i, Ex_i, \delta_i\rangle$, in order to compute reachability in $\mathcal{T}$ we build a *region* RSM R for $\mathcal{T}$, over the alphabet Σ, whose components are the region automata obtained from the components of the original TRSM. More formally, $R = \langle A'_1, \ldots, A'_n\rangle$, where:

$$A'_i = \langle (N_i \times Reg) \cup B_i, Y_i, En_i \times Reg, Ex_i \times Reg, \delta'_i\rangle, \quad \text{with } 1 \leq i \leq n$$

and Reg is the finite set of regions for the set of clocks X and the indexed family $\{c_x\}_{x\in X}$, where c_x is the maximal constant c which is compared with clock x in some clock constraint φ decorating a transition of $\mathcal{T}$. $\mathcal{R}$ is computed as the fix point of an iterative process, which builds a chain of region RSMs $\mathcal{R}^{(0)}, \ldots, \mathcal{R}^{(k)}$. The idea is that, for the sake of reachability, given a run of a TRSM_1, it is possible to abstract away all the sub-runs bounded by matching pairs of call and return transitions. This abstraction can be performed by augmenting the region RSM $\mathcal{R}$ with *summary transitions*, which can be used, during the reachability analysis, to skip the invocation of and the return from the component, provided that the exit node of the component is actually reachable from the corresponding entry node. The process described below iteratively computes these summary transitions.

The initial RSM is $R^0 = \langle A'^{(0)}_1, \ldots, A'^{(0)}_n\rangle$, where $A'^{(0)}_i$=$\langle (N_i \times Reg) \cup B_i, Reg, Y_i, En'_i, Ex'_i, \delta^{(0)}_i\rangle$, with $1 \leq i \leq n$. The transitions in $\delta^{(0)}_i$ are:

1) $\langle (u_1,reg_1),\sigma,(u_1,reg_2)\rangle$, if reg_2 is a time successor of reg_1;
2) $\langle (u_1,reg_1),\sigma,(u_2,reg_2)\rangle$, if $\langle u_1, \sigma, \phi, r_1, \emptyset, u_2\rangle \in \delta_i$, $reg_1 \in \phi$ and $reg_2 = reg_1 \downarrow_{r_1}$;
3) $\langle (u_1,reg_1),\sigma,(b, (u_2,reg_2))\rangle$, if $\langle u_1, \sigma, \phi, r_1, \emptyset, (b, u_2)\rangle \in \delta_i$, $reg_1 \in \phi$ and $reg_2 = reg_1 \downarrow_{r_1}$;

Notice that $\delta^{(0)}_i$ does not contain return transitions. At the $(k+1)$-th iteration ($k \geq 0$), we compute $\delta^{(k+1)}_i$, by adding appropriate summary transitions. Suppose that there is a call transition in the i-th component from a node u_1 to a box b invoking the j-th component into the entry node en, and there is a return transition from the exit node ex of the j-th component to a node u_2 of the i-th component. In order to add a summary transition from (u_1, reg_1) to (u_2, reg_2), we need to know whether (ex, reg'') is locally reachable (i.e. without exploiting call or return transitions) from (en, reg') in the j-th component of $R^{(k)}$, where reg' is the region resulting from reg_1 after resetting the clocks according to the call transition, reg_2 is the region resulting from reg_1 after resetting the clocks according the return transition in the TRSM and reg'' satisfy the constraint in the return transition.

Formally, the relation transition $\delta^{(k+1)}_i$ is $\delta^{(k)}_i$ augmented with the following transitions. For each entry node (b,(en,reg')), where j=$Y_i(b)$, and for each exit node (ex,reg'') $\in A'^{(k)}_j$, reachable from (en, reg') in $A'^{(k)}_j$, if there is a transition $\langle (u_1, reg_1), \sigma_1, (b, (en, reg'))\rangle$ in $\delta^{(k)}_i$ we add to $\delta^{(k+1)}_i$ the transitions:

- $\langle (u_1, reg_1), \sigma_1, (u_2, reg_1\downarrow_r)\rangle$, whenever $\langle (b, ex), \sigma_2, \varphi, r, X, u_2\rangle \in \delta_i$ and $reg'' \in \varphi$.

The iterative construction terminates when it is not possible to add new transitions. Termination of the procedure is ensured since the number of the states of $\mathcal{R}$ and the number of the transitions in each δ_i is finite.

The region RSM R, resulting from the construction, can be proved equivalent, from the reachability viewpoint, to the original TRMS (see the complete work). Since the reachability problem for RMSs is decidable [1], this establishes decidability of reachability for the class TRSM_1.

Theorem *3:* The reachability problem for the class TRSM_1 is decidable.

As a consequence of Theorem 1, that establish an effective equivalence of TRSM and EPTA_2, and of Theorems 2 and 3, we can state the following expressiveness results.

Corollary *2:* TRSM_1 (resp. EPTA_1) is a proper subclass of TRSM (resp. EPTA_2).

We conclude the section by considering the complexity of the reachability problem, by showing that the problem is PSPACE-complete for the subclass TRSM_0. Notice that, the construction above shows that the reachability problem for TRSM_1 can be solved using exponential space, since the size of the region RSM R is exponential in the size of the TRSM $\mathcal{T}$. At the moment we have no results concerning a strict lower bound for TRSM_1.

The idea underlying the construction for reachability in a TRSM_0 $\mathcal{T}$, is similar to the construction given above for TRSM_1. In this case, however, we do not need to build the region graph for $\mathcal{T}$, as we can add summary transitions directly to $\mathcal{T}$. Indeed, any call transition in a TRSM_0 resets all the clocks when entering the invoked component. Hence, the only relevant local reachability problem from an enter to an exit node in the invoked component is the one which assumes the clock valuation set to zero at the entry node. This is the main difference with respect to TRSM_1, where different clock valuations for the entry node have to be considered, thus forcing to explicitly take into account regions in the definition of a summary transition.

Since the number of summary transitions for TRSM_0 is clearly polynomial in the number of transitions of $\mathcal{T}$, and local reachability in a component boils down to reach-

ability in a Timed Automaton, which is known to be a PSPACE-complete problem [2], we can construct, using only polynomial space, a TRSM_0 $\mathcal{T}'$ equivalent to $\mathcal{T}$ from the reachability viewpoint (see the complete work). This is done, again, using a fix point construction, which builds a chain of TRSM_0 $\mathcal{T}^{(0)}, \ldots, \mathcal{T}^{(m)}$, with $\mathcal{T}^{(0)} = \mathcal{T}$, and the transition relation $\delta_i^{(k+1)}$ $(k \geq 0)$ is $\delta_i^{(k)}$ augmented with summary transitions as follows.

For every box $b \in B_i$, with $Y_i(b) = j$, for every entry node $en \in En_j$ and for every return transition $t_{ex} = \langle (b, ex), \sigma_2, \varphi_2, r_2, X, u_2 \rangle \in \delta_i$, we first build the component machine $A_j^{(k)} \downarrow_{t_{ex}} = \langle N_j \cup B_j \cup \{u'\}, Y_j, En_j, \{u'\}, \delta'_j \rangle$, where:

- $\delta'_j = \delta_j^{(k)} \cup \{\langle ex, \sigma_2, \varphi_2, \emptyset, \emptyset, u' \rangle\}$, and $u' \notin N_j$.

Notice that, for any clock valuation $\mathbf{v}$, $\langle en, \mathbf{v}_0 \rangle \rightarrow^*_{A_j^{(k)} \downarrow_{t_{ex}}} \langle u', \mathbf{v} \rangle$ if and only if $\langle en, \mathbf{v}_0 \rangle \rightarrow^*_{A_j} \langle ex, \mathbf{v} \rangle$ and $\mathbf{v} \in \varphi_2$. Therefore, local reachability of u' in $A_j^{(k)} \downarrow_{t_{ex}}$ ensures both that ex is locally reachable in $A_j^{(k)}$ and that t_{ex} is enabled at the exit node, since the constraint φ_2, occurring in t_{ex}, is satisfied by the reached clock valuation.

Then, if u' is locally reachable from en in $A_j^{(k)} \downarrow_{t_{ex}}$, then for all call transitions $\langle u_1, \sigma_1, \varphi_1, X, \emptyset, (b, en) \rangle \in \delta_i$, the summary transition $\langle u_1, \sigma_1, \varphi_1, r_2, \emptyset, u_2 \rangle$ is added to $\delta_i^{(k+1)}$.

Once computed $\mathcal{T}'$, the reachability problem for an untimed global state $\langle b_1, \ldots, b_s, u \rangle$ in $\mathcal{T}$ can be solved by checking reachability of $\langle b_1, \ldots, b_s, u \rangle$ in $\mathcal{T}'$, according to the following procedure.

Let $A_{i_0}, A_{i_1}, \ldots, A_{i_s}$ be the sequence of the invoked components, with $i_0 = 1$ and $Y_{i_{j-1}}(b_j) = i_j$. We start by guessing a sequence of entry nodes $en_0, \ldots, en_s$, with $en_j \in En_{i_j}$. For each $j \geq 0$, we then check whether (b_{j+1}, en_{j+1}) is locally reachable from en_j in the component A_{i_j}, where, for the sake of local reachability, call transitions are treated as reset transitions. Finally, we check whether u is locally reachable from en_s in the component A_{i_s}. If all the checks are fulfilled, then reachability is ensured. Notice that each local reachability check is a reachability analysis in a Timed Automaton, which can be done using polynomial space. Hence, we can conclude the following theorem:

Theorem *4:* The reachability problem for TRSM_0 is PSPACE-complete.

VI. Conclusion

In this paper we have introduced TRSMs, a real time extension of RSMs, able to model real time recursive systems. We have shown that TRSMs allow to specify interesting context-free properties, both on the untimed and the timed dimensions. Though reachability of the general framework is undecidable, we have shown that the problem is still decidable for the meaningful classes TRSM_1 and TRSM_0, and that it is PSPACE-complete for the class TRSM_0, the same complexity of reachability in standard Timed Automata.

A number of interesting issues are still to be investigated. In particular, tight complexity results for the reachability problem in TRSM_1 are yet to be established.

The paper focuses on the relationship with the most related formalisms, namely PTAs and RSMs. On the other hand, a comparison with other related formalisms need to be settled. In particular, the undecidability proof for EPTAs shows that they are able to simulate clock updates (increment and decrement) similar to those provided by Updatable Timed Automata [3]. Also the formalisms of Stopwatch Automata [8] requires a comparison. It seems that Stopwatch Automata can be simulated by the class EPTA_2, while the opposite seems not to hold. Another relevant issue concerns the study of the class of context-free timed languages accepted by EPTAs and by their subclasses.

References

[1] Alur R., Benedikt, M., Etessami, K., Godefroid, P. Reps, T. and Yannakakis, M.: Analysis of Recursive State Machines. In: 13th International Conference on Computer-aided Verification (CAV'01), pp. 207–220.

[2] Alur R., and Dill D. L.: A Theory of Timed Automata. In: Theoretical Computer Science, Volume 126, pages 183–235, 1994.

[3] Bouyer P., Dufourd C., Fleury E. and Petit A.: Updatable timed automata, Theoretical Computer Science, vol. 321, number 2–3, pages 291–345, 2004.

[4] Dang Z.: Pushdown timed automata: a binary reachability characterization and safety verification. In: Theoretical Computer Science, Volume 302 , Issue 1-3, pages 93–121, 2003.

[5] Minsky M. L.: Computation: finite and infinite machines, Prentice-Hall Inc., Upper Saddle River, NJ, USA., 1967.

[6] Alur R., Parthasathy M.: Decision Problems for Timed Automata: A Survey. In: Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication and Software System: Real Time (SFM-RT 2004), pp. 1–24.

[7] Bouajjani A., Echahed R., Robbana R.: On the Automatic Verification of Systems with Continuous Variables and Unbounded Discrete Data Structures. In: Proc. Hybrid System II, LNCS 999, 1995.

[8] Cassez F. and Larsen K.: The Impressive Power of Stopwatches. In Proc. of CONCUR 2000: Concurrency Theory, pp. 138–152, LNCS 1877, 2000.