Nested Timed Automata

Guoqiang Li[1], Xiaojuan Cai[1], Mizuhito Ogawa[2], and Shoji Yuen[3]

[1] School of Software, Shanghai Jiao Tong University, China
{li.g, cxj}@sjtu.edu.cn
[2] Japan Advanced Institute of Science and Technology, Japan
mizuhito@jaist.ac.jp
[3] Graduate School of Information Science, Nagoya University, Japan
yuen@is.nagoya-u.ac.jp

Abstract. This paper proposes a new timed model named *nested timed automata (NeTAs)*. A NeTA is a pushdown system whose stack symbols are *timed automata (TAs)*. It either behaves as the top TA in the stack, or switches from one TA to another by pushing, popping, or changing the top TA of the stack. Different from existing component-based context-switch models such as *recursive timed automata* and *timed recursive state machines*, when time passage happens, all clocks of TAs in the stack elapse uniformly. We show that the safety property of NeTAs is decidable by encoding NeTAs to the *dense timed pushdown automata*. NeTAs provide a natural way to analyze the recursive behaviors of component-based timed systems with structure retained. We illustrate this advantage by the deadline analysis of nested interrupts.

1 Introduction

Due to the rapid development of large and complex timed systems, requirements to model and analyze complex real-time frameworks with recursive context switches have been stresses. Difficulty comes from two dimensions of infinity, a stack with unbounded number of symbols, and clocks recording dense time.

Timed automata (TAs) [1] are a finite automaton with a finite set of *clocks* that grow uniformly. A typical timed model with context switches is *timed pushdown automata (TPDAs)* [2], equipped with an unbounded stack, where clocks are not updated in the stack. This limitation is found unnatural in analyzing the timed behavior of programs since clock values should be updated in suspension. Recently, a new timed pushdown model, *dense timed pushdown automata (DTPDAs)* [3] has been proposed, where each symbol in the stack is equipped with local clocks named "ages", and all ages in the stack are updated uniformly for time passage. Reachability problem of DTPDAs is in EXPTIME [3].

This paper proposes a new timed model named *nested timed automata (NeTAs)*. A NeTA is a pushdown system whose stack symbols are TAs. It either behaves as the top TA in the stack, or switches from one TA to another following three kinds of transitions: pushing a new TA, popping the current TA when terminates, or replacing the top TA of the stack. This hierarchical design captures the dynamic feature of functionally independent component-based structure of timed systems. The existing models, such as *recursive timed automata* [4], and *timed recursive state machines* [5] do not update clocks in the stack when time passage happens, while in NeTAs, all clocks elapse uniformly. When a TA is pushed into the stack, a set of fresh local clocks is introduced to the system. In this respect, NeTAs may have to handle an unbounded number of local clocks. NeTAs are shown to be encoded to DTPDAs preserving the state reachability. All transitions of NeTAs are simulated by DTPDAs, and vice versa. We illustrate that NeTAs are adopted to analyze the timely deadline of nested interrupts.

The rest of the paper is organized as follows. Section 2 gives an introduction of TAs and DTPDAs. Section 3 gives the formal definition and semantics of NeTAs. Section 4 presents an encoding method from NeTAs to DTPDAs, and proves its correctness. Section 5 illustrates the usages of NeTAs by an application example. Section 6 gives the related work and Section 7 concludes the paper.

Due to the lack of space, we omit proofs of theorems, detailed explanations and nations, which can be found in its extended version [6].

2 Preliminaries

Let $\mathbb{R}^{\geq 0}$ and $\mathbb{N}$ denote the sets of non-negative real numbers and natural numbers, respectively. We define $\mathbb{N}^\omega := \mathbb{N} \cup \{\omega\}$, where ω is the first limit ordinal. Let $\mathcal{I}$ denote the set of *intervals*. An interval is a set of numbers, written as (a, b), $[a, b]$, $[a, b)$ or $(a, b]$, where $a \in \mathbb{N}$ and $b \in \mathbb{N}^\omega$. For a number $r \in \mathbb{R}^{\geq 0}$ and an interval $I \in \mathcal{I}$, we use $r \in I$ to denote that r belongs to I.

Let $X = \{x_1, \ldots, x_n\}$ be a finite set of *clocks*. A *clock valuation* $\nu : X \to \mathbb{R}^{\geq 0}$, assigns a value to each clock $x \in X$. $\nu_{\mathbf{0}}$ represents all clocks in X assigned to zero. Given a clock valuation ν and a time $t \in \mathbb{R}^{\geq 0}$, $(\nu + t)(x) = \nu(x) + t$, for $x \in X$. A clock assignment function $\nu[y \leftarrow b]$ is defined by $\nu[y \leftarrow b](x) = b$ if $x = y$, and $\nu(x)$ otherwise.

2.1 Timed Automata

A timed automaton is an automaton augmented with a finite set of clocks [1, 7]. Time can elapse in a location, while switches are instantaneous.

Since we focus on the *safety properties* (i.e., *emptiness* problem of a TA, or *reachability* problem of a timed transition system), we omit input symbols for all concerned automata, following the formalization in [3].

We adopt the TA definition style from that in [3], which looks different from the one in [1, 7]. The main difference is that we do not adopt *invariant*, a time constraint assigned to each control location. The reason lies that invariants cause time lock problems. When context switches back, it may occur that the system can neither stay in the current control location since the invariant is violated nor transit to other control location since all constraints on transitions are violated. Nondeterministic clock updates are also taken from [9] with interval tests as diagonal free time constraints where decidability results are not affected.

Definition 1 (Timed Automata). *A timed automaton is a tuple* $\mathcal{A} = (Q, q_0, F, X, \Delta) \in \mathscr{A}$*, where*

- Q *is a finite set of control locations, with the initial location* $q_0 \in Q$*,*

- *$F \subseteq Q$ is the set of final locations,*
- *X is a finite set of clocks,*
- *$\Delta \subseteq Q \times \mathcal{O} \times Q$, where $\mathcal{O}$ is a set of* operations. *A transition $\delta \in \Delta$ is a triplet (q_1, ϕ, q_2), written as $q_1 \xrightarrow{\phi} q_2$, in which ϕ is either of*
  Local *ϵ, an* empty *operation,*
  Test *$x \in I?$ where $x \in X$ is a clock and $I \in \mathcal{I}$ is an interval, and*
  Assignment *$x \leftarrow I$ where $x \in X$ and $I \in \mathcal{I}$.*

Given a TA $\mathcal{A} \in \mathscr{A}$, we use $Q(\mathcal{A})$, $q_0(\mathcal{A})$, $F(\mathcal{A})$, $X(\mathcal{A})$ and $\Delta(\mathcal{A})$ to represent its set of control locations, initial location, set of final locations, set of clocks and set of transitions, respectively. We will use similar notations for other models.

Definition 2 (Semantics of TAs). *Given a TA $\mathcal{A} = (Q, q_0, F, X, \Delta)$, a configuration is a pair (q, ν) of a control location $q \in Q$, and a clock valuation ν on X. The transition relation of the TA is represented as follows,*

- Progress transition*: $(q, \nu) \xrightarrow{t}_{\mathscr{A}} (q, \nu + t)$, where $t \in \mathbb{R}^{\geq 0}$.*
- Discrete transition*: $(q_1, \nu_1) \xrightarrow{\phi}_{\mathscr{A}} (q_2, \nu_2)$, if $q_1 \xrightarrow{\phi} q_2 \in \Delta$, and one of the following holds,*
  - **Local** *$\phi = \epsilon$, then $\nu_1 = \nu_2$.*
  - **Test** *$\phi = x \in I?$, $\nu_1 = \nu_2$ and $\nu_2(x) \in I$ holds.*
  - **Assignment** *$\phi = x \leftarrow I$, $\nu_2 = \nu_1[x \leftarrow r]$ where $r \in I$.*

The initial configuration is $(q_0, \nu_\mathbf{0})$. The transition relation is $\rightarrow$ and we define $\rightarrow = \xrightarrow{t}_{\mathscr{A}} \cup \xrightarrow{\phi}_{\mathscr{A}}$, and define $\rightarrow^*$ to be the reflexive and transitive closure of $\rightarrow$.

Remark 1 (Sound Simulation). The TA definition in Definition 1 follows the style in [3]. In [1], a TA allows a logical connection of several constraint tests, e.g. $x \leq 15 \wedge y > 20$, and several resets operations of different clocks during one discrete transition. Only one test or assignment (a generalization of the reset) is allowed during one discrete transition in the definition. Since a discrete transition is followed by a progress transition where time elapses, the main ambiguity of two definitions is whether a conjunction of two tests can be checked one by one, between which the time elapses. It is shown that TA with our definition soundly simulates the timed traces in the original definition, as follows,

For $\geq$ or $>$, $c \xrightarrow{x \in [a, +\infty)?}_{\mathscr{A}} c' \xrightarrow{t}_{\mathscr{A}} c'$ is safely converted to $c \xrightarrow{t}_{\mathscr{A}} c \xrightarrow{[a, +\infty)?}_{\mathscr{A}} c'$, for some configurations c and c'.

For $\leq$ or $<$, $c \xrightarrow{t}_{\mathscr{A}} c \xrightarrow{x \in [0, a]?}_{\mathscr{A}} p'$ is safely converted to $c \xrightarrow{x \in [0, a]?}_{\mathscr{A}} c' \xrightarrow{t}_{\mathscr{A}} c'$, for some configurations c and c'.

For test transitions, a general simulation strategy is, firstly, checking the $\geq$, and $>$ one by one, then check $\leq$ and $<$ later. If there exists a "=" constraint, decomposed it into $\geq \wedge \leq$. For example, a transition $p \xrightarrow{x \leq 15 \wedge y > 20} q$ in the original definition is simulated by two transitions $p \xrightarrow{y \in (20, +\infty)?} p' \xrightarrow{x \in [0, 15]?} q$ under the new definition, where p' is a fresh control location.

For reset transitions, a group of clocks are reset simultaneously can be simulated by resetting clocks one by one, with a zero test of the first reset clock on the tail. For example, a transition $p \xrightarrow{\{x, y\}} q$, resetting x and y simultaneously, in the original definition is simulated by $p \xrightarrow{x \leftarrow [0,0]} p' \xrightarrow{y \leftarrow [0,0]} p'' \xrightarrow{x \in [0,0]?} q$, where p', p'' are fresh control locations.

If a transition contains both test and reset operations, we firstly simulate test operation, then reset operation, following the above rules.

2.2 Dense Timed Pushdown Automata

Dense Timed Pushdown Automata (DTPDAs) [3] extend TPDAs with time update in the stack. Each symbol in the stack is equipped with a local clock named *age*, and all ages in the stack elapse uniformly.

Definition 3 (Dense Timed Pushdown Automata). *A dense timed pushdown automaton is a tuple* $\mathcal{D} = \langle S, s_0, \varGamma, C, \varDelta \rangle \in \mathscr{D}$, *where*

- S *is a finite set of states with the initial state* $s_0 \in S$,
- $\varGamma$ *is a finite stack alphabet,*
- C *is a finite set of clocks, and*
- $\varDelta \subseteq S \times \mathcal{O} \times S$ *is a finite set of transitions.*

A transition $\delta \in \varDelta$ *is a triplet* (s_1, ϕ, s_2), *written as* $s_1 \xrightarrow{\phi} s_2$, *in which* ϕ *is either of*

- **Local** ϵ, *an empty operation,*
- **Test** $x \in I?$, *where* $x \in X$ *is a clock and* $I \in \mathcal{I}$ *is an interval,*
- **Assignment** $x \leftarrow I$ *where* $x \in C$ *and* $I \in \mathcal{I}$,
- **Push** $push(\gamma, I)$, *where* $\gamma \in \varGamma$ *is a stack symbol and* $I \in \mathcal{I}$. *It pushes* γ *to the top of the stack, with the age in the interval* I.
- **Pop** $pop(\gamma, I)$, *where* $\gamma \in \varGamma$ *and* $I \in \mathcal{I}$. *It pops the top-most stack symbol provided that this symbol is* γ, *and its age belongs to* I.
- **Push**$_A$ $push(\gamma, x)$, *where* $\gamma \in \varGamma$ *is a stack symbol and* $x \in C$, *and*
- **Pop**$_A$ $pop(\gamma, x)$, *where* $\gamma \in \varGamma$ *is a stack symbol and* $x \in C$.

Definition 4 (Semantics of DTPDAs). *For a DTPDA* $\langle S, s_0, \varGamma, C, \varDelta \rangle$, *a configuration is a triplet* (s, w, ν) *with* $s \in S$, $w \in (\varGamma \times \mathbb{R}^{\geq 0})^*$, *and a clock valuation* ν *on* X. *Time passage of the stack* $w + t = (\gamma_1, t_1 + t). \cdots .(\gamma_n, t_n + t)$ *for* $w = (\gamma_1, t_1). \cdots .(\gamma_n, t_n)$.

The transition relation of the DTPDA is defined as follows:

- Progress transition*:* $(s, w, \nu) \xrightarrow{t}_{\mathscr{D}} (s, w + t, \nu + t)$, *where* $t \in \mathbb{R}^{\geq 0}$.
- Discrete transition*:* $(s_1, w_1, \nu_1) \xrightarrow{\phi}_{\mathscr{D}} (s_2, w_2, \nu_2)$, *if* $s_1 \xrightarrow{\phi} s_2$, *and one of the following holds,*
  - **Local** $\phi = \epsilon$, *then* $w_1 = w_2$, *and* $\nu_1 = \nu_2$.
  - **Test** $\phi = x \in I?$, *then* $w_1 = w_2$, $\nu_1 = \nu_2$ *and* $\nu_2(x) \in I$ *holds.*
  - **Assignment** $\phi = x \leftarrow I$, *then* $w_1 = w_2$, $\nu_2 = \nu_1[x \leftarrow r]$ *where* $r \in I$.
  - **Push** $\phi = push(\gamma, I)$, *then* $\nu_1 = \nu_2$, *and* $w_2 = (\gamma, r).w_1$ *for some* $r \in I$.
  - **Pop** $\phi = pop(\gamma, I)$, *then* $\nu_1 = \nu_2$, *and* $w_1 = (\gamma, r).w_2$ *for some* $r \in I$.
  - **Push**$_A$ $\phi = push(\gamma, x)$, *then* $\nu_1 = \nu_2$, *and* $w_2 = (\gamma, \nu_1(x))w_1$.
  - **Pop**$_A$ $\phi = pop(\gamma, x)$, *then* $\nu_2 = \nu_1[x \leftarrow t]$, *and* $w_1 = (\gamma, t)w_2$.

The initial configuration $\kappa_0 = (q_0, \epsilon, \nu_{\mathbf{0}})$. *We use* $\hookrightarrow$ *to range over these transitions, and* $\hookrightarrow^*$ *is the transitive closure of* $\hookrightarrow$, *conventionally.*

Example 1. Fig. 1 shows transitions between configurations of a DTPDA consisting of a singleton state set $S = \{\bullet\}$ (omitted in the figure), clocks $C = \{x_1, x_2, x_3\}$, and stack symbols $\Gamma = \{a, b, d\}$. From κ_1 to κ_2, a discrete transition $push(d, x_3)$ pushes $(d, 2.3)$ into the stack. A time transition from κ_2 to κ_3 elapses 2.6 time units, and each value grows older for 2.6. From κ_3 to κ_4, the value of x_2 is reset to 3.8, which lies in the interval $(2, 5]$, and the last transition pops (d, x_1) and resets x_1 to 4.9.

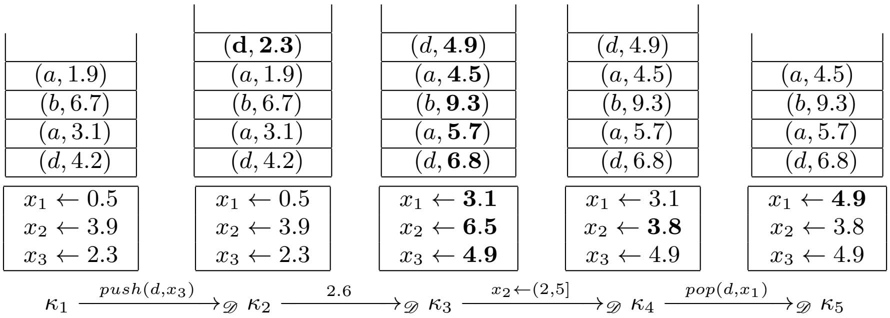

Fig. 1. An Example of DTPDA

Remark 2. Definition 3 extends the definition in [3] by adding $\mathbf{Push}_A$ and $\mathbf{Pop}_A$, which stores and recovers from clocks to ages and vice versa. This extension does not destroy decidability of state reachability of DTPDAs [8], since its symbolic encoding is easily modified to accept $\mathbf{Push}_A$ and $\mathbf{Pop}_A$. For instance, $\mathbf{Push}_A$ is encoded similar to **Push**, except for the definition on *Reset* [3]. $\mathit{Reset}(R)[a \leftarrow I]$ symbolically explores all possibility of the fraction of an instance in I. Instead, $\mathit{Reset}(R)[a \leftarrow x]$ will assign the same integer and fraction parts to x, which means an age is simply placed at the same position to x in the region.

3 Nested Timed Automata

Nested Timed Automata (NeTAs) aim to give an operation strategy to a group of TAs, in which a TA is able to preempt the other ones. All clocks in a NeTA are local clocks, with the scope of their respective TAs. These clocks in the stack elapse simultaneously. An unbounded number of clocks may be involved in one NeTA, due to recursive preemption loops.

Definition 5 (Nested Timed automata). *A nested timed automaton is a triplet* $\mathcal{N} = (T, \mathcal{A}_0, \Delta) \in \mathscr{N}$, *where*

- *T is a finite set of timed automata, with the initial timed automaton $\mathcal{A}_0 \in T$.*

– $\Delta \subseteq T \times \mathcal{P} \times (T \cup \{\varepsilon\})$*, where* $\mathcal{P} = \{push, pop, internal\}$*. A rule* $(\mathcal{A}_i, \Phi, \mathcal{A}_j) \in \Delta$ *is written as* $\mathcal{A}_i \xrightarrow{\Phi} \mathcal{A}_j$*, where*
  Push $\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j$*,*
  Pop $\mathcal{A}_i \xrightarrow{pop} \varepsilon$*, and*
  Internal $\mathcal{A}_i \xrightarrow{internal} \mathcal{A}_j$*.*

The initial state of NeTAs is the initial location in $\mathcal{A}_0$, s.t. $q_0(\mathcal{A}_0)$. We also assume that $X(\mathcal{A}_i) \cap X(\mathcal{A}_j) = \emptyset$, and $Q(\mathcal{A}_i) \cap Q(\mathcal{A}_j) = \emptyset$ for $\mathcal{A}_i, \mathcal{A}_j \in T$ and $i \neq j$.

The operational semantics of NeTAs is informally summarized as follows. It starts with a stack with the only symbol of the initial TA. The system has the following four behaviors: when there exists time passage, all clocks in the stack elapse simultaneously; it is able to behave like the top TA in the stack; when there exists a push transition from the top TA of the stack to the other TA, a new instance of the latter TA is pushed into the stack at any time and executed, while the suspended location of the former TA is recorded in the stack; when the top TA in the stack reaches the final location and a pop transition happens, it will be popped from the stack, and the system will run the next TA beginning with the suspended location; if an internal transition from the top TA to the other TA occurs, the top TA in the stack will be changed to a new instance of the latter TA when it reaches some final location.

Definition 6 (Semantics of NeTAs). *Given a NeTA* $(T, \mathcal{A}_0, \Delta)$*, a configuration is a stack, and the stack alphabet is a tuple* $\langle \mathcal{A}, q, \nu \rangle$*, where* $\mathcal{A} \in T$ *is a timed automaton,* q *is the current running control location where* $q \in Q(\mathcal{A})$*, and* ν *is the clock valuation of* $X(\mathcal{A})$*. For a stack content* $c = \langle \mathcal{A}_1, q_1, \nu_1 \rangle \langle \mathcal{A}_2, q_2, \nu_2 \rangle \ldots \langle \mathcal{A}_n, q_n, \nu_n \rangle$*, let* $c + t$ *be* $\langle \mathcal{A}_1, q_1, \nu_1 + t \rangle \langle \mathcal{A}_2, q_2, \nu_2 + t \rangle \ldots \langle \mathcal{A}_n, q_n, \nu_n + t \rangle$*.*

The transition of NeTAs is represented as follows:

– *Progress transitions:* $c \xrightarrow{t}_{\mathscr{N}} c + t$*.*
– *Discrete transitions:* $c \xrightarrow{\phi}_{\mathscr{N}} c'$ *is defined as a union of the following four kinds of transition relations,*
  - **Intra-action** $\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{\phi}_{\mathscr{N}} \langle \mathcal{A}, q', \nu' \rangle c$*, if* $q \xrightarrow{\phi} q' \in \Delta(\mathcal{A})$*, and one of the following holds,*
    - ∗ **Local** $\phi = \epsilon$*, then* $\nu = \nu'$*.*
    - ∗ **Test** $\phi = x \in I?$*,* $\nu = \nu'$ *and* $\nu'(x) \in I$ *holds.*
    - ∗ **Assignment** $\phi = x \leftarrow I$*,* $\nu' = \nu[x \leftarrow r]$ *where* $r \in I$*.*
  - **Push** $\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{push}_{\mathscr{N}} \langle \mathcal{A}', q_0(\mathcal{A}'), \nu'_{\mathbf{0}} \rangle \langle \mathcal{A}, q, \nu \rangle c$*, if* $\mathcal{A} \xrightarrow{push} \mathcal{A}'$*, and* $q \in Q(\mathcal{A})$*.*
  - **Pop** $\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{pop}_{\mathscr{N}} c$*, if* $\mathcal{A} \xrightarrow{pop} \varepsilon$*, and* $q \in F(\mathcal{A})$*.*
  - **Inter-action** $\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{internal}_{\mathscr{N}} \langle \mathcal{A}', q_0(\mathcal{A}'), \nu'_{\mathbf{0}} \rangle c$*, if* $\mathcal{A} \xrightarrow{internal} \mathcal{A}'$*, and* $q \in F(\mathcal{A})$*.*

The initial configuration $c_0 = \langle \mathcal{A}_0, q_0(\mathcal{A}_0), \nu_{\mathbf{0}} \rangle$*. We use* $\longrightarrow$ *to range over these transitions and* $\longrightarrow^*$ *is the transitive closure of* $\longrightarrow$*, conventionally.*

In followings, we focus on the state reachability that is regarded as the most important property in modelling software behavior.

Definition 7 (Safety Property). *A safety property of NeTAs is defined as the state reachability problem: Given a NeTA $\mathcal{N} = (T, \mathcal{A}_0, \Delta)$, and a control location $p_f \in Q(\mathcal{A})$ for some $\mathcal{A} \in T$, decide whether there exists a configuration c of $\mathcal{N}$ and a clock valuation ν, such that $c_0 \longrightarrow^* \langle \mathcal{A}, p_f, \nu \rangle c$.*

Example 2. We take a simple example to show the usage of NeTAs. Assume that two processes access a shared buffer. One is to read from the buffer periodically each 4 time units. It accomplishes after it reads one or more data. The other is to write to the buffer periodically. The execution time is between 3 and 5 time units. It will return after writes one or more data. The writing process may overtake the reading process which initially starts running. The NeTA is shown in Fig. 2, with three TAs. $\mathcal{A}_0$ is an empty TA for the idle state. $\mathcal{A}_1$ and $\mathcal{A}_2$ are for reading and writing processes, respectively. We have three transition rules: $\mathcal{A}_0 \xrightarrow{internal} \mathcal{A}_1$, $\mathcal{A}_1 \xrightarrow{push} \mathcal{A}_2$, and $\mathcal{A}_2 \xrightarrow{pop} \varepsilon$. The pop transition is not explicitly represented in the figure. We use dash-line frames to represent the border of TAs in the NeTA, double-line arrows to indicate the initial location/TA, and double-line circles to represent the final locations of TAs.

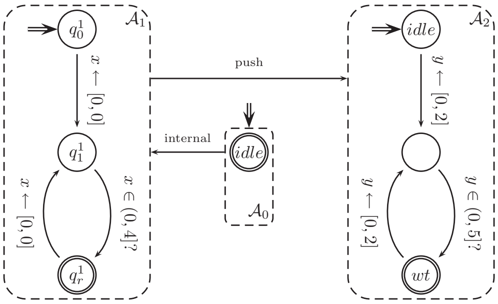

Fig. 2. An Example of NeTA

Remark 3 (Composition with timed automata). A NeTA is composed with a TA by synchronization with shared actions in Σ, where we are allowed to add input symbols as actions on transitions of NeTA. A composed TA presents behavioral properties independent of recursive context switches such as the environment. Although this extension does not increase the expressiveness of NeTAs, it is very useful to model and analyze the behavioral properties in the component-based manner [10, 11]. A formal definition of the parallel composition, between a NeTA $\mathcal{N}$ and a TA $\mathcal{A}$, written as $\mathcal{N} \| \mathcal{A}$, is formally defined in [6].

4 Decidability of Safety Property

In this section we prove the *safety property* problem of NeTAs is decidable by encoding it into DTPDAs, of which state reachability is decidable.

4.1 Encoding NeTA to DTPDA

The idea to encode a NeTA to a DTPDA is straightforward, dealing with multiple clocks at push and pop operations. We adopt extra fresh locations and transitions to check whether a group of push/pop actions happens instantly.

Given a NeTA $\mathcal{N} = (T, \mathcal{A}_0, \varDelta)$, we define $\mathcal{E}(\mathcal{N}) = \langle S, s_0, \varGamma, C, \nabla \rangle$ as the target of DTPDA encoding of $\mathcal{N}$. Each element is described as,

The set of states $S = S_{\mathcal{N}} \cup S_{inter}$, where $S_{\mathcal{N}} = \bigcup_{\mathcal{A}_i \in T} Q(\mathcal{A}_i)$ is the set of all locations of TAs in $T(\mathcal{N})$. S_{inter} is the set of *intermediate* states during the simulation of push, pop, and internal rules of NeTAs. We assume that $S_{\mathcal{N}}$ and S_{inter} are disjoint.

Let $n = |T|$ and $m_i = |X(\mathcal{A}_i)|$ for each $\mathcal{A}_i \in T$. S_{inter} is

$$S_{inter} = (\bigcup_{\mathcal{A}_i \in T} S^i_{reset}) \cup (\bigcup_{\mathcal{A}_i \overset{push}{\longrightarrow} \mathcal{A}_j \in \varDelta} S^{i,j}_{push}) \cup \{o\}$$

- For every $\mathcal{A}_i \in T$, we define $S^i_{reset} = (\bigcup_{k \in \{1 \cdots m_i+1\}} r^i_k) \cup t^i$. $r^i_k \in S^i_{reset}$ is the start state of a transition to initialize the k-th clock of $\mathcal{A}_i$ to 0. t^i is the start state of a testing transition to make sure that no time is elapsed during the sequence of initializing transitions.
- For every push rule $\mathcal{A}_i \overset{push}{\longrightarrow} \mathcal{A}_j$, we define $S^{i,j}_{push} = \bigcup_{k \in \{1 \cdots m_i+1\}} p^{i,j}_k$. $p^{i,j}_k$ is the start state of a push transition that push the pair of the k-th clock of $\mathcal{A}_i$ and its value. After all clock values are stored in the stack, the last destination is the initial state $q_0(\mathcal{A}_j)$ of $\mathcal{A}_j$.
- o is a special state for repeat popping.

The initial state $s_0 = q_0(\mathcal{A}_0)$ is the initial location of the initial TA of $\mathcal{N}$.

The set of clocks $C = \{d\} \cup \bigcup_{\mathcal{A} \in T} X(\mathcal{A})$ consists of all clocks of TA in $T(\mathcal{N})$ and the special dummy clock d only to fulfill the field of push and pop rules, like $push(q, d)$ and $pop(q, d)$. (The value of d does not matter.)

The stack alphabet $\varGamma = C \cup S_{\mathcal{N}}$.

The set of transitions ∇ is the union of $\bigcup_{\mathcal{A}_i \in T} \varDelta(\mathcal{A}_i)$ (as **Local** transitions of $\mathcal{E}(\mathcal{N})$) and the set of transitions described in Fig. 3. For indexes, we assume $0 \le i, j \le n-1$ and $1 \le k \le m_i$ (where i is specified in a context).

Local	$p^{i,j}_{m_i+1} \overset{\epsilon}{\rightarrow} r^j_1,\ r^i_{m_i+1} \overset{\epsilon}{\rightarrow} t^i,\ q_i \overset{\epsilon}{\rightarrow} r^j_1,\ q_i \overset{\epsilon}{\rightarrow} o$ for $q_i \in F(\mathcal{A}_i)$.
Test	$t^i \xrightarrow{x^i_1 \in [0,0]?} q_0(\mathcal{A}_i)$.
Assignment	$r^i_k \xrightarrow{x^i_k \leftarrow [0,0]} r^i_{k+1}$.
Push	$q_i \xrightarrow{push(q_i,d)} p^{i,j}_1,\ p^{i,j}_k \xrightarrow{push(x^i_k,x^i_k)} p^{i,j}_{k+1}\ if\ k \le m_i,$ for $q_i \in Q(\mathcal{A}_i)$.
Pop	$o \xrightarrow{pop(x,x)} o,\ o \xrightarrow{pop(q,d)} q$ forall $x \in X(\mathcal{A}_i)$. $q \in Q(\mathcal{A}_i)$.

Fig. 3. Transition Rules ∇ of $\mathcal{E}(\mathscr{C})$

Definition 8. *For a NeTA $\mathcal{N} = (T, \mathcal{A}_0, \Delta)$, its encoding into a DTPDA $\mathcal{E}(\mathcal{N})$ is as follows.*

$\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j$	$q_i \xrightarrow{push(q_i,d)} p_1^{i,j} \xrightarrow{push(x_1^i,x_1^i)} \cdots p_{m_i}^{i,j} \xrightarrow{push(x_{m_i}^i,x_{m_i}^i)} p_{m_i+1}^{i,j} \xrightarrow{\epsilon}$ $r_1^j \xrightarrow{x_1^j \leftarrow [0,0]} r_2^j \cdots r_{m_j+1}^j \xrightarrow{\epsilon} t^j \xrightarrow{x_1^j \in [0,0]?} q_0(\mathcal{A}_j)$
$\mathcal{A}_i \xrightarrow{pop} \epsilon$	$q_i \xrightarrow{\epsilon} o \xrightarrow{pop(x_{m_i+1}^i,x_{m_i+1}^i)} \cdots \xrightarrow{pop(x_1^i,x_1^i)} o \xrightarrow{pop(q,d)} q$
$\mathcal{A}_i \xrightarrow{internal} \mathcal{A}_j$	$q_i \xrightarrow{\epsilon} r_1^j \xrightarrow{x_1^j \leftarrow [0,0]} r_2^j \cdots r_{m_j+1}^j \xrightarrow{\epsilon} t^j \xrightarrow{x_1^j \in [0,0]?} q_0(\mathcal{A}_j)$

For a *push* transition $\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j$, $\mathcal{E}(\mathcal{N})$ simulates, by storing current state of $\mathcal{A}_i$ into the stack, pushing each clock with its current value as an age, and switching to the initial configuration of $\mathcal{A}_j$ (which consists of initializing each clock $x \in X(\mathcal{A}_j)$, testing that no timed transitions interleaved, and move to the initial state $q_0(\mathcal{A}_j)$).

For a *pop* transition $\mathcal{A}_i \xrightarrow{pop} \epsilon$, $\mathcal{A}_i$ has finished its run at a final state and restores the previous context. $\mathcal{E}(\mathcal{N})$ simulates, first popping and setting each clock (of $\mathcal{E}(\mathcal{N})$), and set a state to q being stored in the stack.

Note that clocks of $\mathcal{E}(\mathcal{N})$ are used for currently running TA (at the top of the stack), and ages are used to store values of clocks of suspended TAs.

Example 3. A NeTA is shown in Fig. 4, which includes two TAs $\mathcal{A}_1$ and $\mathcal{A}_2$. $p_1, p_2 \in Q(\mathcal{A}_1)$, $x_1, x_2 \in X(\mathcal{A}_1)$, $q_1, q_2 \in Q(\mathcal{A}_2)$, and $y_1, y_2 \in X(\mathcal{A}_2)$, respectively. A push transition from $\mathcal{A}_1$ to $\mathcal{A}_2$ occurs at the location p_2, and the value of x_1 and x_2 are 2.9 and 3.3, respectively. After pushing, y_1 and y_2 are reset to zero, and the system begins with q_1. The encoding DTPDA is shown in Fig. 5. p_2 is firstly pushed into the stack, and afterwards, x_1 and x_2 in $\mathcal{A}_1$ is pushed into the stack one by one, with the initial value of the age as their respective value. Then after y_1 and y_2 in $\mathcal{A}_2$ are reset to 0 through the states r_1^2, r_2^2, and r_3^2, the system moves to q_1 provided the value of y_1 is kept as 0.

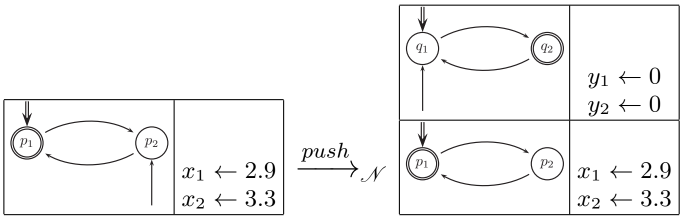

Fig. 4. A Push Transition on a Nested Timed Automaton

Example 4. The NeTA in Fig. 2 is encoded into a DTPDA in Fig. 6.

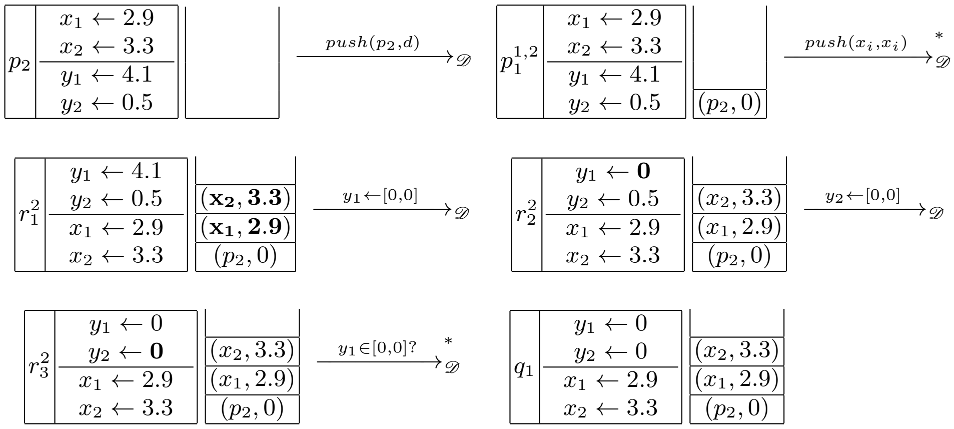
p_2
$x_1 \leftarrow 2.9$
$x_2 \leftarrow 3.3$
$y_1 \leftarrow 4.1$
$y_2 \leftarrow 0.5$
$\xrightarrow{push(p_2,d)}_{\mathscr{D}}$
$p_1^{1,2}$
$(p_2, 0)$
$\xrightarrow{push(x_i,x_i)}{}^{*}_{\mathscr{D}}$
r_1^2
$(\mathbf{x_2}, \mathbf{3.3})$
$(\mathbf{x_1}, \mathbf{2.9})$
$\xrightarrow{y_1 \leftarrow [0,0]}_{\mathscr{D}}$
r_2^2
$y_1 \leftarrow \mathbf{0}$
$(x_2, 3.3)$
$(x_1, 2.9)$
$\xrightarrow{y_2 \leftarrow [0,0]}_{\mathscr{D}}$
r_3^2
$y_2 \leftarrow \mathbf{0}$
$\xrightarrow{y_1 \in [0,0]?}{}^{*}_{\mathscr{D}}$
q_1
$y_1 \leftarrow 0$
$y_2 \leftarrow 0$

Fig. 5. Encoding the Push Transition in DTPDA

- The larger circles are the original states from the NeTA, while the smaller ones are intermediate states.
- Since $\mathcal{A}_0 \xrightarrow{internal} \mathcal{A}_1$, before q_0^0 connects to q_0^1, all clocks in $\mathcal{A}_1$ are reset to zero and kept uniformly. q_0^0 firstly is connected to r_1^1. r_1^1 and r_2^1 reset clocks and t^1 tests the uniformity of clocks.
- Since $\mathcal{A}_1 \xrightarrow{push} \mathcal{A}_2$, each state in $\mathcal{A}_1$ connects to $p_1^{1,2}$ by a transition to push itself. $p_1^{1,2}$ and $p_2^{1,2}$ push each clock in $\mathcal{A}_1$. Before connecting to $q_0^2 \in \mathcal{A}_2$, all clocks in $\mathcal{A}_2$ are similarly reset and tested, through r_1^2, r_2^2 and t^2.
- Since $\mathcal{A}_2 \xrightarrow{pop} \varepsilon$, after some final state of $\mathcal{A}_2$ is reached, each clock in the stack should be popped, through an extra state o. After that, o will connect each state in $\mathcal{A}_1$, by which the respective suspended state is popped.

4.2 Correctness of the Encoding

To reduce state reachability problem of NeTAs to that of DTPDAs, we show that transitions are preserved and reflected by the encoding.

Definition 9 (Encoded Configuration). *For a NeTA* $\mathcal{N} = (T, \mathcal{A}_0, \Delta)$*, its DTPDA encoding* $\mathcal{E}(\mathcal{N}) = \langle S, s_0, \Gamma, C, \nabla \rangle$*. and a NeTA configuration*

$$c = \langle \mathcal{A}_1, q_1, \nu_1 \rangle \langle \mathcal{A}_2, q_2, \nu_2 \rangle \ldots \langle \mathcal{A}_n, q_n, \nu_n \rangle$$

let $c_{hd} = \langle \mathcal{A}_1, q_1, \nu_1 \rangle$ *and* $c_{tl} = \langle \mathcal{A}_n, q_n, \nu_n \rangle$*. A clock valuation of* c*,* $\overline{\nu}(c) : C \to \mathbb{R}^{\geq 0}$ *is defined as* $\overline{\nu}(c)(x) = \nu_1(x)$ *if* $x \in X(\mathcal{A}_1)$*, and* any*, otherwise.* [1] *Let* $\overline{w}(c) = w_1 \cdots w_n$*, where* $w_i = (x^i_{m_i}, \nu_i(x^i_{m_i})) \cdots (x^i_1, \nu_i(x^i_1))(q_i, 0)$*.*

[1] any means any value, since except for a clock in the top stack frame of a nested timed automaton, its value does not matter.

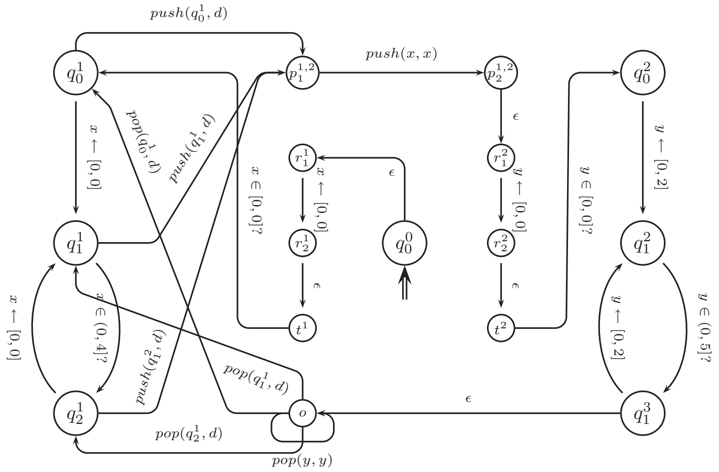

Fig. 6. Encoding the NeTA to DTPDA

We denote a configuration $(q_1, \overline{w}(c_{tl}), \overline{\nu}(c))$ of $\mathcal{E}(\mathcal{N})$ by $[\![c]\!]$. A configuration κ of DTPDA with some c and $\kappa = [\![c]\!]$ is called an encoded configuration*.*

Lemma 1. *Given a NeTA $\mathcal{N}$, its encoding $\mathcal{E}(\mathcal{N})$, and configurations c, c' of $\mathcal{N}$.*

- **(Preservation)** *if $c \longrightarrow c'$, then $[\![c]\!] \hookrightarrow^* [\![c']\!]$;*
- **(Reflection)** *if $[\![c]\!] \hookrightarrow^* \kappa$,*
  1. *there exists c' such that $\kappa = [\![c']\!]$ and $c \longrightarrow^* c'$, or*
  2. *κ is not an encoded configuration, and there exists c' such that $\kappa \hookrightarrow^* [\![c']\!]$ by discrete transitions (of $\mathcal{E}(\mathcal{N})$) and $c \longrightarrow^* c'$.*

From Lemma 1, we have the decidability of the safety property of NeTAs.

Theorem 1. *The state reachability problem of NeTAs is decidable.*

Remark 4 (Global clocks). We can assign a disjoint finite set of global clocks to a NeTA. These global clocks are tested and reassigned during push, pop and internal transitions, to control time conditions for push, pop and internal actions. Global clocks do not affect the safety property of a NeTA, since during the encoding to DTPDA, we just include these clocks to the set of clocks in DTPDA, keeping the copies of global clocks for all stack elements.

Fact 1 *A parallel composition of a NeTA and a TA can be encoded into a NeTA with global clocks by forgetting the synchronizing actions.*

Remark 5 (Encode DTPDAs into NeTAs). We can also encode a DTPDA into a NeTA with global clocks by regarding each state of the DTPDA as a TA with only one (local) clock. These TAs and their respective clocks can thus be used to represent pairs of stack symbols and ages.

5 Deadline Analysis for Nested Interrupts

Timely interrupt handling is part of correctness for real-timed, interrupt-driven system. It is vital for a *deadline analysis* [12, 13] in such systems to check that all specified deadlines for interrupt handling will be met. Such analysis is a daunting task because of large number of different possible interrupt arrival scenarios. An *interrupt signal* from outside transfers the control to an *interrupt handler* deferring the interrupted execution. When there are more than one interrupts, an *interrupt controller* provides priorities for them according to urgency of each interrupt. An interrupt handler is suspended by another handler with higher priority. After the high priority handler is done, the previous handler is resumed. In the followings NeTA combined with TA is shown to be useful for deadline analysis with such nested interrupt handling.

The time and frequency of interrupt signals can be represented by a TA, with input actions as events that trigger interrupt handlers. For instance, Fig. 7 gives an example of a TA that trigger three interrupt handlers, by $coming_P$, $coming_Q$, and $coming_R$, respectively.

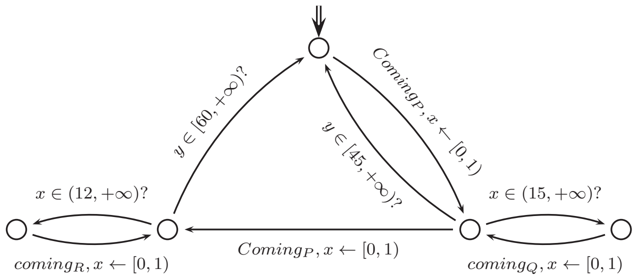
$y \in [60, +\infty)?$
$Coming_P, x \leftarrow [0,1)$
$y \in [45, +\infty)?$
$x \in (12, +\infty)?$
$x \in (15, +\infty)?$
$coming_R, x \leftarrow [0,1)$
$Coming_P, x \leftarrow [0,1)$
$coming_Q, x \leftarrow [0,1)$

Fig. 7. A Timed Automata as an Environment

Assume a finite set of interrupt handler specifications $\mathscr{H}$. Each handler is specified by $P(\mathcal{A}, D)$, where $\mathcal{A}$ is a TA to describe its behavior, and D is its *relative deadline.* A system should guarantee that each executed handler p of $P(\mathcal{A}, D)$ is executed as $\mathcal{A}$ and reached to some final location of $\mathcal{A}$ within D time units. If the handler misses the deadline, it raises an error.

An interrupt handler with relative deadline D is transformed from $\mathcal{A}$ to another TA with error location. $\texttt{Guarded} : \mathscr{H} \to \mathscr{A}$ is defined by $\texttt{Guarded}(P(\mathcal{A}, D)) = (Q^G, q_0^G, F^G, X^G, \Delta^G)$. Each element is shown as follows,

- $Q^G = Q(\mathcal{A}) \cup Q_\Delta \cup \{q_{err}\}$, where $Q_\Delta = \{q_\delta \mid \text{for each } \delta \in \Delta(\mathcal{A})\}$.
- $q_0^G = q_0(\mathcal{A})$, and $F^G = F(\mathcal{A})$.
- $X^G = X(\mathcal{A}) \cup \{x_{sch}\}$.
- $\Delta^G = \Delta_{sch} \cup \Delta_{err}$, where
  - $\Delta_{sch} = \{q \xrightarrow{o} q_\delta, q_\delta \xrightarrow{x_{sch} \in [0,D]?} q' \mid \delta = (q, o, q') \in \Delta(\mathcal{A})\}$.
  - $\Delta_{err} = \{q \xrightarrow{x_{sch} \in (D,+\infty)?} q_{err} \mid q \in Q(\mathcal{A}) \cup Q_\Delta\}$.

Given a finite set of handler specifications $\mathscr{H}$, a priority policy is described by a relation $\prec$ on $\mathscr{H}$. For instance, the most common policy is *fixed priority strategy (FPS)*, where $\prec$ is a strict partial order (irreflexive, asymmetric and transitive). An interrupt controller $\texttt{Sch}(\mathscr{H}, \prec)$ with $\prec$ as a FPS is defined by a nested timed automaton $(T, \mathcal{A}_0, \Delta)$ over a set of input symbols Σ where,

- $\Sigma = \{\texttt{Coming}_P \mid \text{for each } P \in \mathscr{H}\}$.
- $T = \{\texttt{Guarded}(P) \mid \text{for each } P \in \mathscr{H}\} \cup \{\mathcal{A}_{idle}\}$, where $\mathcal{A}_{idle}$ is a singleton timed automaton without any transitions.
- $\mathcal{A}_0 = \mathcal{A}_{idle}$.
- Δ is defined by $\Delta_{idle} \cup \Delta_{push} \cup \Delta_{pop} \cup \Delta_{internal}$, where
  - $\Delta_{idle} = \{\mathcal{A}_{idle} \xrightarrow{\texttt{Coming}_P, push} \mathcal{A} \mid \forall P \in \mathscr{H}, \mathcal{A} = \texttt{guarded}(P)\}$.
  - $\Delta_{push} = \{\mathcal{A} \xrightarrow{\texttt{Coming}_{P'}, push} \mathcal{A}' \mid \forall P, P' \in \mathscr{H}, P \prec P' \wedge \mathcal{A} = \texttt{guarded}(P) \wedge \mathcal{A}' = \texttt{guarded}(P')\}$.
  - $\Delta_{pop} = \{\mathcal{A} \xrightarrow{pop} \varepsilon \mid \forall P \in \mathscr{H}, \mathcal{A} = \texttt{guarded}(P)\}$.
  - $\Delta_{internal} = \{\mathcal{A} \xrightarrow{\texttt{Coming}_{P'}, internal} \mathcal{A}' \mid \forall P, P' \in \mathscr{H}, P \not\prec P' \wedge P \not\succ P' \wedge \mathcal{A} = \texttt{guarded}(P) \wedge \mathcal{A}' = \texttt{guarded}(P')\}$.

After performing parallel composition with a TA as an environment, we are allowed to check the deadline of each interrupt handler P_i through the reachability problem on the error location in $\texttt{Guarded}(P_i)$, considering the fact that a finite number of interrupt handlers are effectively invoked.

6 Related Work

After TAs [1] had been proposed, lots of researches were intended timed context switches. TPDAs were firstly proposed in [2], which enjoys decidability of reachability problem. Dang proved in [14] the decidability of binary reachability of TPDAs. All clocks in TPDAs were treated globally, which were not effected when the context switches.

Our model relied heavily on a recent significant result, named *dense timed pushdown automata (DTPDAs)* [3]. The difference between DTPDAs and NeTAs was the hierarchical feature. In NeTAs, a finite set of local clocks were pushed into the stack at the same time. When a pop action happens, the values of clocks belonging to popped TA were popped simultaneously and reused. This feature eased much for modelling the behavior of time-aware software. In DTPDAs, local clocks must be dealt within some proper bookkeeping process, which was not essential part of the analysis. In [15], a discrete version of DTPDAs, named *discrete timed pushdown automata* was introduced, where time was incremented in discrete steps and thus the ages of clocks and stack symbols are in the natural numbers. This made the reachability problem much simpler, and easier for efficient implementation.

Based on *recursive state machines* [16], two similar timed extensions, *timed recursive state machines (TRSMs)* [5] and *recursive timed automata (RTAs)* [4], were given independently. The main differences from NeTAs were, the two models had no stack time-update during progress transitions, and the number of clocks was essentially finite. The *hierarchical timed automata (HTAs)* [17] kept the similar structure of clocks, where only a bounded number of levels were treated, while NeTAs treated an unbounded number of levels.

The class of *extended timed pushdown automata (ETPDAs)* was proposed in [5]. An ETPDA was a pushdown automaton enriched with a set of clocks, with an additional stack used to store/restore clock valuations. Two stacks were independently. ETPDAs have the same expressiveness with TRTMs via weak timed bisimulation. The reachability problem of ETPDAs was undecidable, while the decidability held with a syntactic restriction on the stack.

Controller automata (CAs) [18, 11], was proposed to analyze interrupts. In a CA, a TA was assigned to each state. A TA at a state may be preempted by another state by a labeled transition. The number of clocks of CAs were finite, and thus when existing preemption loop, only the newest timed context were kept. Given a strict partial order over states, an ordered controller automaton was able to be faithfully encoded into a TA, and thus safety property of the restrictive version was preserved.

The *updatable timed automata (UTAs)* [9] proposed the possibility of updating the clocks in a more elaborate way, where the value of a clock could be reassigned to a basic arithmetic computation result of values of other clocks. UTAs raised up another way to accumulate time when timed context switches, and thus *updatable timed pushdown automata (UTPDAs)* could be an interesting extension.

7 Conclusion

This paper proposed a timed model called *nested timed automata (NeTAs)*. A NeTA was a pushdown system with a finite set of TAs as stack symbols. All clocks in the stack elapse uniformly, capable to model the timed behavior of the suspended components in the stack. The safety property of NeTAs was shown to be decidable by encoding NeTAs to DTPDAs. As an example of its application, behavior of multi-level interrupt handling is concisely modelled and its deadline analysis is encoded as a safety property.

We are planning to develop a tool based on NeTAs. Instead of general NeTAs, we will restrict a class such that a pop action occurs only with an integer-valued age. We expect this subclass of NeTAs can be encoded into updatable TPDAs (without local age), which would be more efficiently implemented.

Acknowledgements This work is supported by the NSFC-JSPS bilateral joint research project (61011140074), NSFC(61100052, 61003013, 61033002, 6126113 0589), and JSPS KAKENHI Grant-in-Aid for Scientific Research(B) (23300008 and 25280023).

References

1. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science **126** (1994) 183–235

2. Bouajjani, A., Echahed, R., Robbana, R.: On the Automatic Verification of Systems with Continuous Variables and Unbounded Discrete Data Structures. In: Proceedings of the International Conference on Hybrid Systems: Computation and Control. LNCS 999, Springer-Verlag (1994) 64–85
3. Abdulla, P.A., Atig, M.F., Stenman, J.: Dense-Timed Pushdown Automata. In: Proceedings of the LICS'12, IEEE Computer Society (2012) 35–44
4. Trivedi, A., Wojtczak, D.: Recursive Timed Automata. In: Proceedings of the ATVA'10. LNCS 6252, Springer-Verlag (2010) 306–324
5. Benerecetti, M., Minopoli, S., Peron, A.: Analysis of Timed Recursive State Machines. In: Proceedings of the TIME'10, IEEE Computer Society (2010) 61–68
6. Li, G., Cai, X., Ogawa, M., Yuen, S.: Nested Timed Automata. Technical Report IS-RR-2013-004, JAIST (2013)
7. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic Model Checking for Real-Time Systems. Information and Computation **111** (1994) 193–244
8. Ogawa, M., Cai, X.: On Reachability of Dense Timed Pushdown Automata. Technical Report IS-RR-2013-005, JAIST (2013)
9. Bouyer, P., Dufourd, C., Fleury, E., Petit, A.: Updatable Timed Automata. Theoretical Computer Science **321** (2004) 291–345
10. Bengtsson, J., Yi, W.: Timed Automata: Semantics, Algorithms and Tools. In: Lecture Notes on Concurrency and Petri Nets. LNCS 3098, Springer-Verlag (2004) 87–124
11. Li, G., Yuen, S., Adachi, M.: Environmental Simulation of Real-Time Systems with Nested Interrupts. In: Proceedings of the TASE'09, IEEE Computer Society (2009) 21–28
12. Brylow, D., Palsberg, J.: Deadline Analysis of Interrupt-Driven Software. IEEE Transactions on Software Engineering (TSE) **30** (2004) 634–655
13. Fersman, E., Krcal, P., Pettersson, P., Yi, W.: Task Automata: Schedulability, Decidability and Undecidability. Information and Computation **205** (2007) 1149–1172
14. Dang, Z.: Pushdown Timed Automata: a Binary Reachability Characterization and Safety Verification. Theoretical Computer Science **302** (2003) 93–121
15. Abdulla, P.A., Atig, M.F., Stenman, J.: The Minimal Cost Reachability Problem in Priced Timed Pushdown Systems. In: Proceedings of the LATA'12. LNCS 7183, Springer-Verlag (2012) 58–69
16. Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T.W., Yannakakis, M.: Analysis of Recursive State Machines. ACM Transactions on Programming Languages and Systems (TOPLAS) **27** (2005) 786–818
17. David, A., Möller, M.O.: From HUPPAAL to UPPAAL - A Translation from Hierarchical Timed Automata to Flat Timed Automata. Technical Report RS-01-11, BRICS (2001)
18. Li, G., Cai, X., Yuen, S.: Modeling and Analysis of Real-Time Systems with Mutex Components. International Journal of Foundations of Computer Science (IJFCS) **23** (2012) 831–851