Zenoness for Timed Pushdown Automata

Parosh Aziz Abdulla   Mohamed Faouzi Atig   Jari Stenman

Timed pushdown automata are pushdown automata extended with a finite set of real-valued clocks. Additionaly, each symbol in the stack is equipped with a value representing its age. The enabledness of a transition may depend on the values of the clocks and the age of the topmost symbol. Therefore, dense-timed pushdown automata subsume both pushdown automata and timed automata. We have previously shown that the reachability problem for this model is decidable. In this paper, we study the zenoness problem and show that it is EXPTIME-complete.

1 Introduction

Pushdown automata [9, 20, 16, 17] and timed automata [6, 11, 10] are two of the most widely used models in verification. Pushdown automata are used as models for (discrete) recursive systems, whereas timed automata model timed (nonrecursive) systems. Several models have been proposed that extend pushdown automata with timed behaviors [8, 14, 12, 13, 15].

We consider the model of (Dense-)Timed Pushdown Automata (TPDA), introduced in [1], that subsumes both pushdown automata and timed automata. As in the case of a pushdown automaton, a TPDA has a stack which can be modified by pushing and popping. A TPDA extends pushdown automata with time in the sense that the automaton (1) has a finite set of real-valued clocks, and (2) stores with each stack symbol its (real-valued) age. Pushing a symbol adds it on top of the stack with an initial age chosen nondeterministically from a given interval. A pop transition removes the topmost symbol from the stack provided that it matches the symbol specified by the transition, and that its age lies within a given interval. A TPDA can also perform timed transitions, which simulate the passing of time. A timed transition synchronously increases the values of all clocks and the ages of all stack symbols with some non-negative real number. The values of the clocks can be tested for inclusion in a given interval or non-deterministically reset to a value in a given interval. The model yields a transition system that is infinite in two dimensions; the stack contains an unbounded number of symbols, and each symbol is associated with a unique real-valued clock.

In [1], we showed that the reachability problem, i.e. the problem of deciding whether there exists a computation from the initial state to some target state, is decidable (specifically, EXPTIME-complete). In this paper, we address the zenoness problem for TPDA. The zenoness problem is the problem of deciding whether there is a computation that contains infinitely many discrete transitions (i.e. transitions that are not timed transitions) in *finite time* [5, 4, 21]. Zeno computations may represent specification errors, since these kinds of runs are not possible in real-world systems. We show that the zenoness problem for TPDA can be reduced to the problem of deciding whether a pushdown automaton has an infinite run with the labelling a^ω. The latter problem is polynomial in the size of the pushdown automaton, which is itself exponential in the size of the TPDA.

Related Work

The works in [8, 14, 12, 13, 15] consider pushdown automata extended with clocks. However, these models separate the timed part and the pushdown part of the automaton, which means that the stack

symbols are not equipped with clocks.

In [7], the authors define the class of *extended pushdown timed automata*. An extended pushdown timed automaton is a pushdown automaton enriched with a set of clocks, with an additional stack used to store/restore clock valuations. In our model, clocks are associated with stack symbols and store/restore operations are disallowed. The two models are quite different. This is illustrated, for instance, by the fact that the reachability problem is undecidable in their case.

In [22], the authors introduce *recursive timed automata*, a model where clocks are considered as variables. A recursive timed automaton allows passing the values of clocks using either *pass-by-value* or *pass-by-reference* mechanism. This feature is not supported in our model since we do not allow pass-by-value communication between procedures. Moreover, in the recursive timed automaton model, the local clocks of the caller procedure are stopped until the called procedure returns. The authors show decidability of the reachability problem when either all clocks are passed by reference or none is passed by reference. This is the model that is most similar to ours, since in both cases, the reachability problem reduces to the same problem for a pushdown automaton that is abstract-time bisimilar to the timed system.

In a recent work [2] we have shown decidability of the reachability problem for *discrete-timed* pushdown automata, where time is interpreted as being incremented in discrete steps and thus the ages of clocks and stack symbols are in the natural numbers. This makes the reachability problem much simpler to solve, and the method of [2] cannot be extended to the dense-time case.

Finally, the zenoness problem for different kinds of timed systems is well studied in the literature (see, e.g., [4, 18] for timed automata and [3] for dense-timed Petri nets).

2   Preliminaries

We use $\mathbb{N}$ and $\mathbb{R}^{\geq 0}$ to denote the set of natural numbers and non-negative reals, respectively. For values $n, m \in \mathbb{N}$, we denote by the intervals $[n : m]$, $(n : m)$ $[n : m)$, $(n : m]$, $[n : \infty)$ and $(n : \infty)$ the sets of values $r \in \mathbb{R}^{\geq 0}$ satisfying the constraints $n \leq r \leq m$, $n < r < m$, $n \leq r < m$, $n < r \leq m$, $n \leq r$, and $n < r$, respectively. We let $\mathcal{I}$ denote the set of all such intervals.

For a non-negative real number $r \in \mathbb{R}^{\geq 0}$, with $r = n + r'$ $n \in \mathbb{N}$, and $r' \in [0 : 1)$, we let $\lfloor r \rfloor = n$ denote the *integral part*, and $frac(r) = r'$ denote the *fractional part* of r. Given a set S, we use 2^S for the powerset of S. For sets A and B, $f : A \to B$ denotes a (possibly partial) function from A to B. We write $f(a) = \bot$ when f is undefined at $a \in A$. We use $dom(f)$ and $range(f)$ to denote the domain and range of f. We write $f[a \leftarrow b]$ to denote the function f' such that $f'(a) = b$ and $f'(x) = f(x)$ for $x \neq a$. The set of partial functions from A to B is written as $[A \to B]$.

Let A be an alphabet. We denote by A^*, (resp. A^+) the set of all *words* (resp. non-empty words) over A. The empty word is denoted by ϵ. For a word w, $|w|$ denotes the length of w (we have $|\epsilon| = 0$). For words w_1, w_2, we use $w_1 \cdot w_2$ for the concatenation of w_1 and w_2. We extend the operation $\cdot$ to sets W_1, W_2 of words by defining $W_1 \cdot W_2 = \{w_1 \cdot w_2 \,|\, w_1 \in W_1, w_2 \in W_2\}$. We denote by $w[i]$ the ith element a_i of $w = a_1 \ldots a_n$.

We use A^ω to denote the set of all infinite words over the alphabet A. We let a^ω denote the infinite word $aaa\ldots$ and write $|w| = \infty$ for any infinite word w over A.

We define a binary *shuffle operation* $\otimes$ inductively: For $w \in (2^A)^*$, define $w \otimes \epsilon = \epsilon \otimes w = \{w\}$. For sets $r_1, r_2 \in 2^A$ and words $w_1, w_2 \in (2^A)^*$, define $(r_1 \cdot w_1) \otimes (r_2 \cdot w_2) = (r_1 \cdot (w_1 \otimes (r_2 \cdot w_2))) \cup (r_2 \cdot ((r_1 \cdot w_1) \otimes w_2))) \cup ((r_1 \cup r_2) \cdot (w_1 \otimes w_2))$.

Let $w = a_1 \ldots a_m$ and $w' = b_1 \ldots b_n$ be words in A^*. An *injection* from w to w' is a partial function $h : \{1, \ldots, m\} \to \{1, \ldots, n\}$ that is *strictly monotonic*, i.e. for all $i, j \in \{1, \ldots, m\}$, if $i < j$ and $h(i), h(j) \neq \bot$, then $h(i) < h(j)$. The *fragmentation* w/h of w w.r.t. h is the sequence $\langle w_0 \rangle a_{i_1} \langle w_1 \rangle a_{i_2} \ldots \langle w_{k-1} \rangle a_{i_k} \langle w_k \rangle$, where $dom(h) = \{i_1, \ldots, i_k\}$ and $w = w_0 \cdot a_{i_1} \cdot w_1 \cdot \ldots \cdot a_{i_k} \cdot w_k$. The fragmentation w'/h is the sequence $\langle w'_0 \rangle b_{j_1} \langle w'_1 \rangle \ldots \langle w'_{l-1} \rangle b_{j_l} \langle w'_l \rangle$, where $range(h) = \{j_1, \ldots, j_l\}$ and $w' = w'_0 \cdot b_{j_1} \cdot \ldots \cdot b_{i_l} \cdot w'_l$.

Pushdown Automata

A pushdown automaton is a tuple $(Q, q_{init}, \Sigma, \Gamma, \Delta)$, where Q is a finite set of states, q_{init} is an initial state, Σ is a finite input alphabet, Γ is a finite stack alphabet and Δ is a set of transition rules of the form $\langle q, \sigma, \mathbf{nop}, q' \rangle$, $\langle q, \sigma, \mathbf{pop}(a), q' \rangle$ or $\langle q, \sigma, \mathbf{push}(a), q' \rangle$, where $q, q' \in Q$, $a \in \Gamma$ and $\sigma \in \Sigma \cup \{\epsilon\}$.

A configuration is a pair (q, w), where $q \in Q$ and $w \in \Gamma^*$. We define $\gamma_{init} = (q_{init}, \epsilon)$ to be the *initial configuration*, meaning that the automaton starts in the initial state and with an empty stack. We define a transition relation $\to$ on the set of configurations in the following way: Given two configurations $\gamma_1 = (q_1, w_1)$, $\gamma_2 = (q_2, w_2)$ and a transition rule $t = \langle q_1, \sigma, \mathbf{op}, q_2 \rangle \in \Delta$, we write $\gamma_1 \xrightarrow{t} \gamma_2$ if one of the following conditions is satisfied:

- $\mathbf{op} = \mathbf{nop}$ and $w_2 = w_1$,

- $\mathbf{op} = \mathbf{push}(a)$ and $w_2 = a \cdot w_1$,

- $\mathbf{op} = \mathbf{pop}(a)$ and $w_1 = a \cdot w_2$.

For any transition rule $t = \langle q_1, \sigma, \mathbf{op}, q_2 \rangle \in \Delta$, define $\Sigma(t) = \sigma$. We define $\longrightarrow = \cup_{t \in \Delta} \xrightarrow{t}$ and let $\longrightarrow^*$ be the reflexive transitive closure of $\longrightarrow$. We say that an infinite word $\sigma_1 \sigma_2 \sigma_3 \cdots \in \Sigma^\omega$ is a *trace* of $\mathcal{P}$ if there exists configurations $\gamma_1, \gamma_2, \gamma_3, \ldots$ such that $\gamma_1 = \gamma_{init}$, $\gamma_1 \xrightarrow{t_1} \gamma_2 \xrightarrow{t_2} \gamma_3 \xrightarrow{t_3} \ldots$, and $\Sigma(t_i) = \sigma_i$ for all $i \in \mathbb{N}$. We denote by $Traces(\mathcal{P})$ the set of all traces of $\mathcal{P}$.

3 Timed Pushdown Automata

Syntax

A *Timed Pushdown Automaton* (TPDA) is a tuple $\mathcal{T} = (Q^{\mathcal{T}}, q_{init}^{\mathcal{T}}, X^{\mathcal{T}}, \Gamma^{\mathcal{T}}, \Delta^{\mathcal{T}})$. Here, $Q^{\mathcal{T}}$ is a finite set of *states*, $q_{init}^{\mathcal{T}} \in Q^{\mathcal{T}}$ is an initial state, $X^{\mathcal{T}}$ is a finite set of *clocks*, $\Gamma^{\mathcal{T}}$ is a finite *stack alphabet* and $\Delta^{\mathcal{T}}$ is finite set of *transition rules* of the form $(q, \mathbf{op}, q')$, where $q, q' \in Q^{\mathcal{T}}$ and $\mathbf{op}$ is one of the following:

nop An "empty" operation that does not modify the clocks or the stack,

push(a, I) Pushes $a \in \Gamma^{\mathcal{T}}$ to the stack with a (nondeterministic) initial age in $I \in \mathcal{I}$,

pop(a, I) Pops the topmost symbol if it is a and its age is in $I \in \mathcal{I}$,

test(x, I) Tests if the value of $x \in X^{\mathcal{T}}$ is within $I \in \mathcal{I}$,

reset(x, I) Sets the value of $x \in X^{\mathcal{T}}$ (nondeterministically) to some value in $I \in \mathcal{I}$.

Intuitively, a transition rule $\langle q, \mathbf{op}, q' \rangle$ means that the automaton is allowed to move from state q to state q' while performing the operation $\mathbf{op}$. The **nop** operation can be used to switch states without changing the stack or the values of clocks.

Semantics

The semantics of TPDA is defined by a transition relation over the set of *configurations*. A configuration is a tuple $(q, \mathtt{X}, w)$, where $q \in Q^{\mathcal{T}}$ is a state, $\mathtt{X} : X^{\mathcal{T}} \to \mathbb{R}^{\geq 0}$ is a *clock valuation* which assigns concrete values to clocks, and $w = (a_1, y_1) \ldots (a_n, y_n) \in (\Gamma^{\mathcal{T}} \times \mathbb{R}^{\geq 0})^*$ is a *stack content*. In other words, the stack content is a sequence of pairs, each pair consisting of a symbol and its age. Here, (a_1, y_1) is on the top and (a_n, y_n) is on the bottom of the stack. Given a TPDA $\mathcal{T}$, we denote by $Conf(\mathcal{T})$ the set of all configurations of $\mathcal{T}$.

The transition relation consists of two types of transitions; *discrete* transitions, which correspond to applications of the transition rules, and *timed* transitions, which simulate the passing of time.

Discrete Transitions. Let $t = (q, \mathbf{op}, q') \in \Delta^{\mathcal{T}}$ be a transition rule and let $\gamma = (q, \mathtt{X}, w)$ and $\gamma' = (q', \mathtt{X}', w')$ be configurations. We have $\gamma \xrightarrow{t} \gamma'$ if one of the following conditions is satisfied:

- $\mathbf{op} = \mathbf{nop}$, $w' = w$ and $\mathtt{X}' = \mathtt{X}$,

- $\mathbf{op} = \mathbf{push}(a, I)$, $w' = (a, v)w$ for some $v \in I$, and $\mathtt{X}' = \mathtt{X}$,

- $\mathbf{op} = \mathbf{pop}(a, I)$, $w = (a, v)w'$ for some $v \in I$, and $\mathtt{X}' = \mathtt{X}$,

- $\mathbf{op} = \mathbf{test}(x, I)$, $w' = w$, $\mathtt{X}' = \mathtt{X}$ and $\mathtt{X}(x) \in I$,

- $\mathbf{op} = \mathbf{reset}(x, I)$, $w' = w$, and $\mathtt{X}' = \mathtt{X}[x \leftarrow v]$ for some $v \in I$.

Timed Transitions. Let $r \in \mathbb{R}^{\geq 0}$ be a real number. Given a clock valuation $\mathtt{X}$, let $\mathtt{X}^{+r}$ be the function defined by $\mathtt{X}^{+r}(x) = \mathtt{X}(x) + r$ for all $x \in X$. For any stack content $w = (a_1, y_1) \ldots (a_n, y_n)$, let w^{+r} be the stack content $(a_1, y_1 + r) \ldots (a_n, y_n + r)$. Let $\gamma = (q, \mathtt{X}, w)$ and $\gamma' = (q', \mathtt{X}', w')$ be configurations. Then $\gamma \xrightarrow{r} \gamma'$ if and only if $q' = q$, $\mathtt{X}' = \mathtt{X}^{+r}$ and $w' = w^{+r}$.

Computations. A *computation* (or *run*) π is a (finite or infinite) sequence of the form $(\gamma_1, \tau_1, \gamma_2)(\gamma_2, \tau_2, \gamma_3) \cdots$ (written as $\gamma_1 \xrightarrow{\tau_1} \gamma_2 \xrightarrow{\tau_2} \gamma_3 \cdots$) such that $\gamma_i \xrightarrow{\tau_i} \gamma_{i+1}$ for all $1 \leq i \leq |\pi|$. For $\tau \in (\Delta^{\mathcal{T}} \cup \mathbb{R}^{\geq 0})$, we define $Disc(\tau) = 1$ if $\tau \in \Delta^{\mathcal{T}}$ and $Disc(\tau) = 0$ if $\tau \in \mathbb{R}^{\geq 0}$. Then, the number of discrete transitions in π is defined as $|\pi|_{disc} = \sum_{i=1}^{|\pi|} Disc(\tau_i)$. Note that if $|\pi| = \infty$, then it may be the case that $|\pi|_{disc} = \infty$.

In this paper, we will consider the *duration* of transitions. Given a $\tau \in (\Delta^{\mathcal{T}} \cup \mathbb{R}^{\geq 0})$, the duration $\delta(\tau)$ is defined in the following way:

- $\delta(\tau) = 0$ if $\tau \in \Delta^{\mathcal{T}}$. Discrete transitions have no duration.

- $\delta(\tau) = \tau$ if $\tau \in \mathbb{R}^{\geq 0}$.

For a computation π, we define the duration $\delta(\pi)$ to be $\sum_{i=1}^{|\pi|} \delta(\tau_i)$. If the automaton can perform infinitely many discrete transitions in finite time, it exhibits a behavior called *zenoness*.

Definition 1 (Zenoness). *A computation π is* zeno *if it contains infinitely many discrete transitions and has a finite duration, i.e. if $|\pi|_{disc} = \infty$ and $\delta(\pi) \leq c$ for some $c \in \mathbb{N}$. π is* non-zeno *if it is not zeno.*

The *zenoness problem* is the question whether a given TPDA contains a zeno run starting from the initial configuration:

Definition 2 (The Zenoness Problem). *Given a TPDA $\mathcal{T}$, decide if there exists a computation $\pi = \gamma_{init} \longrightarrow \gamma_1 \longrightarrow \gamma_2 \longrightarrow \ldots$ from the initial configuration of $\mathcal{T}$ such that π is zeno.*

Given two computations $\pi = \gamma_1 \xrightarrow{\tau_1} \gamma_2 \xrightarrow{\tau_2} \gamma_3 \xrightarrow{\tau_3} \cdots$ and π', we say that π' is a *prefix* of π if $\pi = \pi'$ or $\pi' = \gamma_1 \xrightarrow{\tau_1} \gamma_2 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_{n-1}} \gamma_n$ for some $1 \le n$. We say that π' is a *suffix* of π if either $\pi' = \pi$ or $\pi' = \gamma_n \xrightarrow{\tau_n} \gamma_{n+1} \xrightarrow{\tau_{n+1}} \cdots$ for some $n \in \mathbb{N}$. We define the *concatenation* of a finite computation $\pi = \gamma_1 \xrightarrow{\tau_1} \gamma_2 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_{n-1}} \gamma_n$ with a (finite or infinite) computation $\pi' = \gamma_1' \xrightarrow{\tau_1'} \gamma_2' \xrightarrow{\tau_2'} \cdots$, where $\gamma_n = \gamma_1'$, as $\pi \cdot \pi' = \gamma_1 \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_{n-1}} \gamma_n \xrightarrow{\tau_1'} \gamma_2' \xrightarrow{\tau_2'} \cdots$.

Let $\pi = \pi_1 \cdot \pi_2$ be a computation. We call the suffix π_2 a *unit suffix* if $\delta(\pi) < 1$. The question whether a TPDA $\mathcal{T}$ has a zeno run starting from the initial configuration can be reduced to the question whether there exists a run from the initial configuration which contains a zeno unit suffix:

Lemma 1. *A TPDA $\mathcal{T}$ contains a zeno run iff $\mathcal{T}$ contains a run $\pi = \pi_1 \cdot \pi_2$ such that π_2 is zeno and $\delta(\pi_2) < 1$.*

Proof. We prove both directions:

If: By the definition of zenoness.

Only if: Assume π is a zeno run of $\mathcal{T}$. Then there exists a smallest $n \in \mathbb{N}$ such that $\delta(\pi) \le n$. Call it c. This means that the longest prefix π' of π for which $\delta(\pi') \le c - 1$ contains finitely many discrete transitions. We have that after π', the next transition in π will be a timed transition $\gamma \xrightarrow{r} \gamma'$ for some $r \in \mathbb{R}^{\ge 0}$, and $\delta(\pi') + r > c - 1$. Now, let $\pi_1 = \pi' \cdot \gamma \xrightarrow{r} \gamma'$, and let π_2 be the remaining suffix in π. We can conclude that $\delta(\pi_2) = c - \delta(\pi_1) < c - (c - 1) = 1$. $\qquad\square$

In the rest of the paper, we will show how to decide whether $\mathcal{T}$ contains a run that has a zeno unit suffix. Intuitively, given a TPDA $\mathcal{T}$, we will construct a pushdown automaton $\mathcal{P}$ which simulates the behavior of $\mathcal{T}$. The pushdown automaton $\mathcal{P}$ operates in two modes.

Initially, $\mathcal{P}$ runs in the first mode, in which it simulates the behavior of $\mathcal{T}$ exactly as described in [1]. While $\mathcal{P}$ runs in the first mode, all transitions are labelled with ϵ. At any time, $\mathcal{P}$ may guess that it can simulate a unit suffix. In this case, $\mathcal{P}$ switches to the second mode, in which it reads symbols from a unary alphabet (say $\{a\}$) while simulating discrete transitions of $\mathcal{T}$. The question whether $\mathcal{T}$ contains a unit suffix then reduces to the question whether $Traces(\mathcal{P})$ includes a^ω.

4 Symbolic Encoding

In this section, we show how to construct a symbolic PDA $\mathcal{P}$ that simulates the behavior of a TPDA $\mathcal{T}$. The PDA uses a symbolic *region* encoding to represent the infinitely many clock valuations of $\mathcal{T}$ in a finite way. The notion of regions was introduced in the classical paper on timed automata [6], in which a timed automaton is simulated by a region automaton (a finite-state automaton that encodes the regions in its states). This abstraction relies on the set of clocks being fixed and finite. Since a TPDA may in general operate on unboundedly many clocks (the stack is unbounded, and each symbol has an age), we cannot rely on this abstraction. Instead, we use regions of a special form as stack symbols in $\mathcal{P}$. For each symbol in the stack of $\mathcal{T}$, the stack of $\mathcal{P}$ contains, at the same position, a region that relates the stack symbol with all clocks. A problem with this approach is that we might need to record relations between clocks and stack symbols that lie arbitrarily far apart in the stack. However, in [1], we show that it is enough to enrich the regions in finite way (by recording the relationship between clocks and adjacent stack symbols), thus keeping the stack alphabet of $\mathcal{P}$ finite.

R_2 $\begin{array}{|c|} (\vdash^\bullet,0) \\ (\vdash,0) \end{array}$ $\boxed{(a^\bullet,1)}$ $\begin{array}{|c|} (x_1^\bullet,4) \\ (x_1,4) \end{array}$ $\begin{array}{|c|} (x_2^\bullet,3) \\ (x_2,3) \end{array}$ $\boxed{(c,0)}$

R_1 $\boxed{(\vdash,0)}$ $\begin{array}{|c|} (a,1) \\ (\vdash^\bullet,1) \end{array}$ $\begin{array}{|c|} (b^\bullet,2) \\ (x_1,4) \end{array}$ $\begin{array}{|c|} (x_2,3) \\ (x_1^\bullet,5) \end{array}$ $\boxed{(x_2^\bullet,3)}$
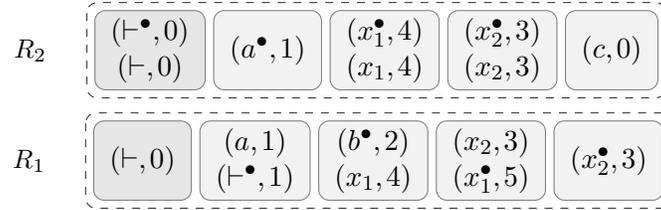
Figure 1: Two examples of regions

Regions

A region is a word over sets, where each set consists of a number of *items*. There are *plain items*, which represent the values of clocks and the topmost stack symbols. In addition, this set includes a reference clock $\vdash$, which is always 0 except when simulating a pop transition. Furthermore, we have *shadow items* which record the values of the corresponding plain items in the region below. Shadow items are used to remember the time that elapses while the plain symbols they represent are not on the top of the stack.

To illustrate this, assume that the region R_1 in Figure 1 is the topmost region in the stack. R_1 records the integral values and the relationships between the clocks x_1, x_2, the topmost stack symbol a and the reference clock $\vdash$. It also relates these symbols to the values of x_1, x_2, b and $\vdash$ in the previous topmost region. Now, if we simulate the pushing of c with inital age in $[0:1]$, one of the possible resulting regions is R_2. The region R_2 uses $x_1^\bullet$, $x_2^\bullet$ and $\vdash^\bullet$ to record the previous values of the clocks (initially, their values are identical to those of their plain counterparts). The value of the previous topmost symbol a is recorded in $a^\bullet$. Finally, the region relates the new topmost stack symbol c with all the previously mentioned symbols.

We define the set $Y = X \cup \Gamma \cup \{\vdash\}$ of plain items and a corresponding set $Y^\bullet = X^\bullet \cup \Gamma^\bullet \cup \{\vdash^\bullet\}$ of shadow items. We then define the set of *items* $Z = Y \cup Y^\bullet$.

Let c_{max} be the largest constant in the definition of $\mathcal{T}$. We denote by Max the set $\{0, 1, \ldots, c_{max}, \infty\}$. A *region* R is a word $r_1 \ldots r_n \in (2^{Z \times Max})^+$ such that the following holds:

- $\sum_{i=1}^n |(\Gamma \times Max) \cap r_i| = 1$ and $\sum_{i=1}^n |(\Gamma^\bullet \times Max) \cap r_i| = 1$. There is exactly one occurrence of a stack symbol and one occurrence of a shadow stack symbol.

- $\sum_{i=1}^n |(\{\vdash\} \times Max) \cap r_i| = 1$ and $\sum_{i=1}^n |(\{\vdash^\bullet\} \times Max) \cap r_i| = 1$. There is exactly one occurrence of $\vdash$ and one occurrence of $\vdash^\bullet$.

- For all clocks $x \in X$, $\sum_{i=1}^n |(\{x\} \times Max) \cap r_i| = 1$ and $\sum_{i=1}^n |(\{x^\bullet\} \times Max) \cap r_i| = 1$. Each plain clock symbol and shadow clock symbol occurs exactly once.

- $r_i \neq \varnothing$ for all $2 \leq i \leq n$. Only the first set may be empty.

For items $z \in Z$, if we have $(z, k) \in r_i$ for some $i \in \{1, \ldots, n\}$ and some (unique) $k \in Max$, then define $Val(R, z) = k$ and $Index(R, z) = i$. Otherwise, define $Val(R, z) = \bot$ and $Index(R, z) = \bot$ (this may be the case for stack symbols). We define $R^\top = \{z \in Z \mid Index(R, z) \neq \bot\}$.

Operations on Regions

In order to define the transition rules of the symbolic PDA, we need a number of operations on regions:

Testing Satisfiability

When we construct new regions, we need to limit the values of the items to certain intervals. To do this, we define what it means for a region to *satisfy* a membership predicate. Given an item $z \in Z$, an interval $I \in \mathcal{I}$ and a region R such that $z \in R^\top$, we write $R \models (z \in I)$ if and only if one of the following conditions is satisfied:

- $Index(R, z) = 1$, $Val(R, z) \neq \infty$ and $Val(R, z) \in I$,

- $Index(R, z) > 1$, $Val(R, z) \neq \infty$ and $Val(R, z) + v \in I$ for all $v \in \mathbb{R}^{\geq 0}$ such that $0 < v < 1$,

- $Val(R, z) = \infty$ and I is of the form $(m : \infty)$ or the form $[m : \infty)$ for some $m \in \mathbb{N}$.

Adding and Removing Items

In the following, we define operations that describe how items are added and deleted from regions. We also define, in terms of these operations, an operation that assigns a new value to an item.

For a region $R = r_1 \ldots r_n$, an item $z \in Z$ and an $k \in Max$, we define $R \oplus (z, k)$ to be the set of regions R' satisfying the following conditions:

- $R = r_1 \ldots r_{i-1} (r_i \cup \{(z, k)\}) r_{i+1} \ldots r_n$, where $1 \leq i \leq n$

- $R = r_1 \ldots r_i \{(z, k)\} r_{i+1} \ldots r_n$, where $1 \leq i \leq n$

We extend the definition of $\oplus$ by letting $R \oplus a$ denote the set $\bigcup_{m \in Max} R \oplus (a, m)$, i.e. the set of regions where we have added all possible values of a.

We define $R \ominus z$ to be the region $R' = r'_1 \ldots r'_n$, where, for $1 \leq i \leq n$, we have $r'_i = r_i \setminus \{\{z\} \times Max\}$ if $r_i \setminus \{\{z\} \times Max\} \neq \varnothing$, and $r'_i = \epsilon$ otherwise. We extend the definition of $\ominus$ to sets of items in the following way: $R \ominus \varnothing = R$ and $R \ominus \{z_1, \ldots z_n\} = (R \ominus z_1) \ominus \{z_2, \ldots z_n\}$.

Given a region R, an item $z \in Z$ and an interval $I \in \mathcal{I}$, we define an *assignment* operation. We write $R[z \leftarrow I]$ to mean the set of regions R' such that $R' \in (R \ominus z) \oplus z$ and $R' \models (z \in I)$. For any number $n \in \mathbb{N}$, we write $R[z \leftarrow n]$ to mean $R[z \leftarrow [n : n]]$.

Creating New Regions

When we push a new stack symbol, we need to record the values of clocks and the value of the current top-most stack symbol. The operation $Make$ takes as arguments a region, a stack symbol, and an interval, It constructs the set of regions in which the shadow items record the values of the plain items in the old topmost region, and the value of the stack symbol is in the given interval.

Given a region R, a stack symbol $a \in \Gamma$ and an interval $I \in \mathcal{I}$, we define $Make(R, a \in I)$ to be the set of regions R' such that there are R_1, R_2, R_3 satisfying the following:

- $R_1 = R \ominus (R^\top \cap Y^\bullet)$,

- If $R_1 = r_1 \ldots r_n$, then $R_2 = r'_1 \ldots r'_n$, where $r'_i = r_i \cup \{(y^\bullet, k) \,|\, (y, k) \in r_i\}$ for $i \in \{1, \ldots, n\}$,

- $R_3 = R_2 \ominus (R^\top \cap \Gamma)$,

- $R' \in R_3 \oplus a$ and $R' \models (a \in I)$.

Passage of Time

We implement the passage of time by *rotating* the region. A rotation describes the effect of the smallest timed transition that changes the region. If the leftmost set (i.e. the set which represents items with fractional part 0) is nonempty, a timed transition, no matter how small, will "push" those items out. If the leftmost set is empty, the smallest timed transition that changes the regions is one that makes the fractional parts of those items 0.

Given a pair $(z, k) \in Z \times Max$, define $(z, k)^+ = (z, k')$, where $k' = k + 1$ if $k < c_{max}$ and $k' = \infty$ otherwise. For a set $r \in 2^{Z \times Max}$, define $r^+ = \{(z, k)^+ \mid (z, k) \in r\}$. For a region $R = r_1 \ldots r_n$, we define $R^+ = R'$ such that one of the following conditions is satisfied:

- $r_1 \neq \varnothing$ and $R' = \varnothing r_1 \ldots r_n$,

- $r_1 = \varnothing$ and $R' = r_n^+ r_1 \ldots r_{n-1}$.

We denote by R^{++} the set $\{R, R^+, (R^+)^+, ((R^+)^+)^+, \ldots\}$. Note that this set is finite.

Product

When we simulate a pop transition, the region that we pop contains the most recent values of all clocks. On the other hand, the region below it contains shadow items that record relationships between items further down the stack. We need to keep all of this information. To do this, we define a product operation $\odot$ that merges the information contained in two regions. For regions $P = p_1 \ldots p_{|P|}$ and $Q = q_1 \ldots q_{|Q|}$, and an injection h from $\{1, \ldots, |P|\}$ to $\{1, \ldots, |Q|\}$, we write $P \preceq_h Q$ iff the following conditions are satisfied:

- $Val(P, y^\bullet) = Val(Q, y)$ for all $y \in P^\top \cap Y$,

- For every $i > 1$, $h(i) \neq \perp$ iff there exists a $y \in Y$ such that $Index(P, y) = i$,

- $h(1) = 1$,

- For all $y \in Y$, $i \in \{1, \ldots, |P|\}$ and $j \in \{1, \ldots, |Q|\}$, if $Index(P, y) = i$ and $Index(Q, y^\bullet) = j$, then $h(i) = j$.

We say that P *supports* Q, written $P \preceq Q$, if $P \preceq_h Q$ for some h. Let $P/h = p_{i_1} \langle P_1 \rangle p_{i_2} \ldots p_{i_m} \langle P_m \rangle$ and $Q/h = q_{j_1} \langle Q_1 \rangle q_{j_2} \ldots q_{j_m} \langle Q_m \rangle$. We define $p'_k = p_{i_k} \cap (Y^\bullet \cup \Gamma)$ and $q'_k = q_{j_k} \cap (X \cup \{\vdash\})$. Finally, define $r_1 = p'_1 \cup q'_1$ and, for $k \in \{2, \ldots, m\}$, define $r_k = p'_k \cup q'_k$ if $p_k \cup q'_k \neq \varnothing$ and $r_k = \epsilon$ if $p_k \cup q'_k = \varnothing$. Then, $R \in P \odot Q$ if $R = r_1 \cdot R_1 \cdot r_2 \ldots r_m \cdot R_m$ and $R_k \in P_k \otimes Q_k$ for $k \in \{1, \ldots, m\}$.

5   An EXPTIME **Upper Bound for the Zenoness Problem**

In this section, we prove our main result:

Theorem 2. *The Zenoness problem for TPDA is in* EXPTIME.

The rest of this section will be devoted to the proof of Theorem 2. Given a TPDA $\mathcal{T} = (Q^\mathcal{T}, q^\mathcal{T}_{init}, \Gamma^\mathcal{T}, X^\mathcal{T}, \Delta^\mathcal{T})$, we construct an (untimed) PDA $\mathcal{P} = (Q^\mathcal{P}, q^\mathcal{P}_{init}, \Sigma^\mathcal{P}, \Gamma^\mathcal{P}, \Delta^\mathcal{P})$ such that $\mathcal{P}$ simulates zeno runs of $\mathcal{T}$. More specifically, $\mathcal{P}$ simulates a zeno run of $\mathcal{T}$ by first simulating the prefix, and then simulating the unit suffix. In order to do this, $\mathcal{P}$ runs in two modes. In the first mode, it simulates the prefix. In the second mode, it simulates the suffix while keeping track of the fact that the value of a special *control clock* $x_{control}$ is smaller than 1. We now describe the components of $\mathcal{P}$.

The states of $\mathcal{P}$ are composed of two disjoint sets; the *genuine* states $\{\mathbf{0}, \mathbf{1}\} \times Q^{\mathcal{T}}$ and some *temporary* states $\mathtt{Tmp}$. Each genuine state (m, q) contains a state q from $Q^{\mathcal{T}}$ and a symbol m indicating the current simulation mode. If $m = \mathbf{0}$, $\mathcal{P}$ is currently simulating the prefix of a run. Conversely, if $m = \mathbf{1}$, $\mathcal{P}$ is simulating the suffix. The temporary states are used for intermediate transitions between configurations containing genuine states. We assume that we have functions $\mathtt{tmp}$, $\mathtt{tmp}_1$ and $\mathtt{tmp}_2$ that input arguments and map them to a unique element in $\mathtt{Tmp}$. The initial state $q_{init}^{\mathcal{P}}$ of $\mathcal{P}$ is the state $(\mathbf{0}, q_{init}^{\mathcal{T}})$. The input alphabet $\Sigma^{\mathcal{P}}$ is the unary alphabet $\{a\}$. The automaton reads an a when (and only when) it simulates a discrete transition in the suffix. When it simulates any other transition, it reads ϵ. Let $x_{control} \notin X^{\mathcal{T}}$ be a special control clock. The stack alphabet $\Gamma^{\mathcal{P}}$ contains all possible regions over the items $Z \cup \{x_{control}, x_{control}^{\bullet}\}$. The purpose of the control clock is to limit the duration of the suffix. We will now describe the set $\Delta^{\mathcal{T}}$ of transition rules:

nop For each transition rule $\langle q_1, \mathbf{nop}, q_2 \rangle \in \Delta^{\mathcal{T}}$, the set $\Delta^{\mathcal{P}}$ contains the transition rules $\langle (\mathbf{0}, q_1), \epsilon, \mathbf{nop}, (\mathbf{0}, q_2) \rangle$ and $\langle (\mathbf{1}, q_1), a, \mathbf{nop}, (\mathbf{1}, q_2) \rangle$. Nop transitions are used for switching states without modifying the clocks or the stack.

test$(x \in I)$ We simulate a test transition in $\mathcal{T}$ with two transition in $\mathcal{P}$. If the topmost region satisfies the constraint, we pop it and move to a temporary state. Since a test transition is not supposed to modify the stack, we push back the same region we popped, while moving to the second genuine state. Formally, for each transition rule $\tau = \langle q_1, \mathbf{test}(x \in I), q_2 \rangle \in \Delta^{\mathcal{T}}$, and region R such that $R \vDash (x \in I)$, the set $\Delta^{\mathcal{T}}$ contains the transition rules:

- $\langle (\mathbf{0}, q_1), \epsilon, \mathbf{pop}(R), \mathtt{tmp}(\tau, R, \mathbf{0}) \rangle$,
- $\langle \mathtt{tmp}(\tau, R, \mathbf{0}), \epsilon, \mathbf{push}(R), (\mathbf{0}, q_2) \rangle$ (for simulating the prefix),
- $\langle (\mathbf{1}, q_1), a, \mathbf{pop}(R), \mathtt{tmp}(\tau, R, \mathbf{1}) \rangle$,
- $\langle \mathtt{tmp}(\tau, R, \mathbf{1}), \epsilon, \mathbf{push}(R), (\mathbf{1}, q_2) \rangle$ (for simulating the suffix).

reset$(x \leftarrow I)$ We simulate reset transitions by popping the topmost region and pushing it back, in a similar way to test transitions, except that the given clock is nondeterministically set to some value in the given interval. Formally, for each transition rule $\tau = \langle q_1, \mathbf{reset}(x \leftarrow I), q_2 \rangle \in \Delta^{\mathcal{T}}$, and each pair of regions R, R' such that $R' \in R[x \leftarrow I]$, the set $\Delta^{\mathcal{T}}$ contains the transition rules:

- $\langle (\mathbf{0}, q_1), \epsilon, \mathbf{pop}(R), \mathtt{tmp}(\tau, R, \mathbf{0}) \rangle$,
- $\langle \mathtt{tmp}(\tau, R, \mathbf{0}), \epsilon, \mathbf{push}(R'), (\mathbf{0}, q_2) \rangle$ (for simulating the prefix),
- $\langle (\mathbf{1}, q_1), a, \mathbf{pop}(R), \mathtt{tmp}(\tau, R, \mathbf{1}) \rangle$,
- $\langle \mathtt{tmp}(\tau, R, \mathbf{1}), \epsilon, \mathbf{push}(R'), (\mathbf{1}, q_2) \rangle$ (for simulating the suffix).

push(a, I) We will need two temporary states to simulate a push. First, we move to a temporary state while popping the topmost region. This is done in order to remember its content. Then, we push back that region unmodified. Finally, we push a region containing the given symbol, constructed from the previous topmost region such that the initial age of the symbol is in the given interval. Formally, for each transition rule $\tau = \langle q_1, \mathbf{push}(a, I), q_2 \rangle \in \Delta^{\mathcal{T}}$, and each pair of regions R, R' such that $R' \in Make(R, a \in I)$, the set $\Delta^{\mathcal{T}}$ contains the transition rules:

- $\langle (\mathbf{0}, q_1), \epsilon, \mathbf{pop}(R), \mathtt{tmp}_1(\tau, R, \mathbf{0}) \rangle$,

- $\langle \mathtt{tmp}_1(\tau, R, \mathbf{0}), \epsilon, \mathbf{push}(R), \mathtt{tmp}_2(\tau, R, \mathbf{0}) \rangle$,

- $\langle \mathtt{tmp}_2(\tau, R, \mathbf{0}), \epsilon, \mathbf{push}(R'), (\mathbf{0}, q_2) \rangle$ (for simulating the prefix),

- $\langle (\mathbf{1}, q_1), a, \mathbf{pop}(R), \mathtt{tmp}(\tau, R, \mathbf{1}) \rangle$,

- $\langle \mathtt{tmp}_1(\tau, R, \mathbf{1}), \epsilon, \mathbf{push}(R), \mathtt{tmp}_2(\tau, R, \mathbf{1}) \rangle$,

- $\langle \mathtt{tmp}(\tau, R, \mathbf{1}), \epsilon, \mathbf{push}(R'), (\mathbf{1}, q_2) \rangle$ (for simulating the suffix).

$\mathbf{pop}(a, I)$ The simulation of pop transitions also requires two temporary states. First, we pop the topmost region and move to a temporary state. Then, in order to update the new topmost region, we need to first pop it, then rotate and merge it with the first region we popped, and finally push back the result. Formally, for each transition rule $\tau = \langle q_1, \mathbf{pop}(a, I), q_2 \rangle \in \Delta^{\mathcal{T}}$, and all regions R_1, R_1', R_2, such that $R_2 \vDash (a \in I)$ and $R_1' \in \bigcup \{ R_2 \odot R' \mid R' \in R_1^{++} \text{ and } R' \preceq R_2 \}$, the set $\Delta^{\mathcal{T}}$ contains the transition rules:

- $\langle (\mathbf{0}, q_1), \epsilon, \mathbf{pop}(R_2), \mathtt{tmp}_1(\tau, R_2, \mathbf{0}) \rangle$,

- $\langle \mathtt{tmp}_1(\tau, R_2, \mathbf{0}), \epsilon, \mathbf{pop}(R_1), \mathtt{tmp}_2(\tau, R_2, \mathbf{0}) \rangle$,

- $\langle \mathtt{tmp}_2(\tau, R_2, \mathbf{0}), \epsilon, \mathbf{push}(R'), (\mathbf{0}, q_2)) \rangle$ (for simulating the prefix),

- $\langle (\mathbf{1}, q_1), a, \mathbf{pop}(R_2), \mathtt{tmp}_1(\tau, R_2, \mathbf{1}) \rangle$,

- $\langle \mathtt{tmp}_1(\tau, R_2, \mathbf{1}), \epsilon, \mathbf{pop}(R_1), \mathtt{tmp}_2(\tau, R_2, \mathbf{1}) \rangle$,

- $\langle \mathtt{tmp}_2(\tau, R_2, \mathbf{1}), \epsilon, \mathbf{push}(R'), (\mathbf{1}, q_2)) \rangle$ (for simulating the suffix).

Timed Transitions

For every state $q \in Q^{\mathcal{T}}$ and every pair of regions R, R' such that $R' \in R^+[\vdash \leftarrow [0:0]]$ (this is a singleton set), the set $\Delta^{\mathcal{P}}$ contains the transition rules:

- $\langle (\mathbf{0}, q), \epsilon, \mathbf{pop}(R), \mathtt{tmp}(timed, q, R, \mathbf{0}) \rangle$,

- $\langle \mathtt{tmp}(timed, q, R, \mathbf{0}), \epsilon, \mathbf{push}(R'), (\mathbf{0}, q) \rangle$.

Additionally, if $R' \vDash (x_{control} \in [0:1))$, then $\Delta^{\mathcal{P}}$ also contains the transitions

- $\langle (\mathbf{1}, q), \epsilon, \mathbf{pop}(R), \mathtt{tmp}(timed, R, \mathbf{1}) \rangle$,

- $\langle \mathtt{tmp}(timed, R, \mathbf{1}), \epsilon, \mathbf{push}(R'), (\mathbf{1}, q) \rangle$.

Switching Modes

In addition to the transitions described so far, $\mathcal{P}$ must also be able to switch from mode $\mathbf{0}$ to mode $\mathbf{1}$. This is done nondeterministically at any point in the simulation of the prefix. When the automaton changes mode, it resets the control clock $x_{control}$. For each state $q \in Q^{\mathcal{T}}$ and region R, the set $\Delta^{\mathcal{P}}$ contains the transition rules $\langle (\mathbf{0}, q), \epsilon, \mathbf{pop}(R), \mathtt{tmp}(switch, q, R) \rangle$ and $\langle \mathtt{tmp}(switch, q, R), \epsilon, \mathbf{push}(R'), (\mathbf{1}, q) \rangle$, where R' is the region in the singleton set $R[x_{control} \leftarrow 0]$.

Correctness. The simulation of the prefix (mode **0**) works exactly like the simulation in [1]. The simulation of the suffix (mode **1**) only imposes a restriction on the duration of the remaining run, namely that the value of the control clock $x_{control}$ may not reach 1. In other words, the automaton may simulate any unit suffix. Additionally, it reads an a each time it simulates a discrete transition. This, together with Lemma 1 implies the following result:

Lemma 3. *There exists a zeno run in $\mathcal{T}$ if and only if for the corresponding symbolic automaton $\mathcal{P}$, we have $a^\omega \in Traces(\mathcal{P})$.*

Using our construction, the size of $\mathcal{P}$ is exponential in the size of $\mathcal{T}$. The problem of checking $a^\omega \in Traces(\mathcal{P})$ is polynomial in the size of $\mathcal{P}$ [9]. This gives membership in EXPTIME for Theorem 2.

6 An EXPTIME **Lower Bound for the Zenoness Problem**

The following theorem gives EXPTIME-hardness for the zenoness problem for TPDA (matching its upper bound).

Theorem 4. *The zenoness problem for TPDA is EXPTIME-hard.*

Proof. The following problem is EXPTIME-complete [19]: Given a labelled pushdown automaton $\mathcal{P}$ recognizing the language L and n finite automata $A_1, \ldots, A_n$ recognizing languages $L_1, \ldots, L_n$, is the intersection $L \cap \bigcap_{i=1}^n L_i$ empty? This problem can be reduced, in polynomial time, to the zenoness problem for a TPDA $\mathcal{T}$. The pushdown part of $\mathcal{T}$ simulates $\mathcal{P}$, while a clock x_i encodes the state of the finite automaton A_i. We can use an additional control clock to ensure that no time passes during the simulation. We may assume w.l.o.g. that the finite automata are free of ϵ-transitions. An ϵ-transition of $\mathcal{P}$ is simulated by the pushdown part of $\mathcal{T}$. A labelled transition of $\mathcal{P}$ is first simulated by the pushdown part of $\mathcal{T}$ and then followed by a sequence of transitions that checks and updates the clocks in order to ensure that each finite automaton A_i is able to match the transition.

From a final state of $\mathcal{P}$, we introduce a series of transitions that checks if all finite-state automata A_i are also in their final states. If they are, we move to a special state of $\mathcal{T}$ from which there exists a zeno run. In this special state, we remove the restriction that time cannot pass and we add a self-loop performing a **nop** operation. Thus, the intersection $L \cap \bigcap_{i=1}^n L_i$ is empty if and only if $\mathcal{T}$ does not contain a zeno run. □

7 Conclusion and Future Work

In this paper, we have considered the problem of detecting zeno runs in TPDA. We showed that the zenoness problem for TPDA is EXPTIME-complete. The proof uses a reduction from the zenoness problem for TPDA to the problem of deciding whether a^ω is contained in the set of traces of a PDA. More specifically, given a TPDA $\mathcal{T}$, we construct a PDA $\mathcal{P}$ which simulates zeno runs of $\mathcal{T}$ and whose size is exponential in the size of $\mathcal{T}$.

We are currently considering the problem of computing the minimal (or infimal, if it does not exist) *reachability cost* in the model of *priced* TPDA, in which discrete transitions have *firing costs* and stack contents have *storage costs*, meaning that the cost of taking a timed transition depends on the stack content.

Another interesting question is whether there are fragments of some suitable metric logic for which model checking TPDA is decidable.

References

[1] P. A. Abdulla, M. F. Atig & J. Stenman (2012): *Dense-timed pushdown automata*. In: *Logic in Computer Science (LICS), 2012 27th Annual IEEE Symposium on*, IEEE, doi:10.1109/LICS.2012.15.

[2] P. A. Abdulla, M. F. Atig & J. Stenman (2012): *The Minimal Cost Reachability Problem in Priced Timed Pushdown Systems*. In: *LATA*, doi:10.1007/978-3-642-28332-1_6.

[3] P. A. Abdulla, P. Mahata & R. Mayr (2005): *Decidability of Zenoness, syntactic boundedness and token-liveness for dense-timed petri nets*. In: *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science*, Springer, pp. 58–70, doi:10.1007/978-3-540-30538-5_6.

[4] R. Alur (1991): *Techniques for Automatic Verification of Real-Time Systems*. Ph.D. thesis, Dept. of Computer Sciences, Stanford University.

[5] R. Alur & D. L. Dill (1990): *Automata For Modeling Real-Time Systems*. In: *ICALP*, *LNCS* 443, Springer, pp. 322–335, doi:10.1007/BFb0032042.

[6] R. Alur & D. L. Dill (1994): *A Theory of Timed Automata*. Theor. Comput. Sci. 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[7] M. Benerecetti, S. Minopoli & A. Peron (2010): *Analysis of Timed Recursive State Machines*. In: *TIME*, IEEE Computer Society, pp. 61–68, doi:10.1109/TIME.2010.10.

[8] A. Bouajjani, R. Echahed & R. Robbana (1994): *On the Automatic Verification of Systems with Continuous Variables and Unbounded Discrete Data Structures*. In: *Hybrid Systems*, LNCS 999, Springer, pp. 64–85, doi:10.1007/3-540-60472-3_4.

[9] A. Bouajjani, J. Esparza & O. Maler (1997): *Reachability Analysis of Pushdown Automata: Application to Model-Checking*. In: *CONCUR*, LNCS 1243, Springer, pp. 135–150, doi:10.1007/3-540-63141-0_10.

[10] P. Bouyer, F. Cassez, E. Fleury & K. G. Larsen (2004): *Optimal Strategies in Priced Timed Game Automata*. In: *FSTTCS*, LNCS 3328, Springer, pp. 148–160, doi:10.1007/978-3-540-30538-5_13.

[11] P. Bouyer & F. Laroussinie (2008): *Model Checking Timed Automata*. In Stephan Merz & Nicolas Navet, editors: *Modeling and Verification of Real-Time Systems*, ISTE Ltd. – John Wiley & Sons, Ltd., pp. 111–140, doi:10.1002/9780470611012.ch4.

[12] Z. Dang (2003): *Pushdown timed automata: a binary reachability characterization and safety verification*. Theor. Comput. Sci. 302(1-3), pp. 93–121, doi:10.1016/S0304-3975(02)00743-0.

[13] Z. Dang, T. Bultan, O. H. Ibarra & R. A. Kemmerer (2004): *Past pushdown timed automata and safety verification*. Theor. Comput. Sci. 313(1), pp. 57–71, doi:10.1016/j.tcs.2003.10.004.

[14] Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer & J. Su (2000): *Binary Reachability Analysis of Discrete Pushdown Timed Automata*. In: *CAV*, LNCS 1855, Springer, pp. 69–84, doi:10.1007/10722167_9.

[15] M. Emmi & R. Majumdar (2006): *Decision Problems for the Verification of Real-Time Software*. In: *HSCC*, LNCS 3927, Springer, pp. 200–211, doi:10.1007/11730637_17.

[16] J. Esparza, D. Hansel, P. Rossmanith & S. Schwoon (2000): *Efficient Algorithms for Model Checking Pushdown Systems*. In: *CAV*, *LNCS* 1855, Springer, doi:10.1007/10722167_20.

[17] J. Esparza & S. Schwoon (2001): *A BDD-Based Model Checker for Recursive Programs*. In: *CAV*, *LNCS* 2102, Springer, pp. 324–336, doi:10.1007/3-540-44585-4_30.

[18] F. Herbreteau, B. Srivathsan & I. Walukiewicz (2012): *Efficient emptiness check for timed büchi automata*. Formal Methods in System Design 40(2), pp. 122–146, doi:10.1007/s10703-011-0133-1.

[19] A. Heußner, J. Leroux, A. Muscholl & G. Sutre (2012): *Reachability Analysis of Communicating Pushdown Systems*. Logical Methods in Computer Science 8(3:23), pp. 1–20, doi:10.2168/LMCS-8(3:23)2012.

[20] S. Schwoon (2002): *Model-Checking Pushdown Systems*. Ph.D. thesis, Technische Universität München.

[21] S. Tripakis (1999): *Verifying Progress in Timed Systems*. In: *ARTS*, *LNCS* 1601, Springer, pp. 299–314, doi:10.1007/3-540-48778-6_18.

[22] A. Trivedi & D. Wojtczak (2010): *Recursive timed automata*. In: *Proceedings of the 8th international conference on Automated technology for verification and analysis*, ATVA, pp. 306–324, doi:10.1007/978-3-642-15643-4_23.