Dense-Timed Pushdown Automata

Parosh Aziz Abdulla Mohamed Faouzi Atig Jari Stenman
Department of Information Technology
Uppsala University
Uppsala, Sweden

***Abstract*—We propose a model that captures the behavior of real-time recursive systems. To that end, we introduce *dense-timed pushdown automata* that extend the classical models of pushdown automata and timed automata, in the sense that the automaton operates on a finite set of real-valued clocks, and each symbol in the stack is equipped with a real-valued clock representing its "age". The model induces a transition system that is infinite in two dimensions, namely it gives rise to a stack with an unbounded number of symbols each of which with a real-valued clock. The main contribution of the paper is an EXPTIME-complete algorithm for solving the reachability problem for dense-timed pushdown automata.**

***Index Terms*—Formal verification, Automata.**

I. Introduction

During the last two decades there has been a large amount of work devoted to the verification of *discrete* program models that have *infinite* state spaces such as Petri nets, pushdown systems, counter automata, and channel machines. In particular, pushdown systems have been studied extensively as a model for the analysis of recursive programs (e.g., [7], [17], [14], [15]). In parallel, *timed automata* [3], [9], [8] are the most widely used model for the analysis of systems with *timed* behaviors. Recently, several works have augmented discrete infinite-state models with timed behaviors. For instance, many different formalisms have been proposed for extending Petri nets with clocks and timed constraints, leading to various definitions of *Timed Petri Nets* (e.g., [5], [2]).

In this paper, we consider *(Dense-)Timed Push-Down Automata* (or *TPDA* for short). A TPDA combines the classical models of pushdown automata and timed automata in the sense that the automaton is equipped with a finite set of real-valued clocks, and each symbol in the stack is equipped with a real-valued clock representing its "age". The types of transitions performed by a TPDA include the usual ones by a pushdown automaton, namely pushing and popping symbols to/from the stack. However, in a similar manner to timed automata, the transitions are now conditioned by the values of the clocks in the automaton. Furthermore, transitions are labeled by intervals that constrain the ages of the symbols that are pushed or popped from/to the stack. Thus, when a transition t is fired, we (i) check that the values of the clocks satisfy the conditions stated by t, (ii) update the clock values as specified by t, and (iii) perform a stack operation. The latter may either be a *pop* operation that removes the topmost symbol in the stack provided its has the correct label and age, or a *push* operation that adds a symbol whose age belongs to a given interval. Finally, a TPDA may perform a *timed transition* in which the clock values and the ages of the symbols are all increased at the same rate. The TPDA model thus subsumes both the model of pushdown automata and timed automata. More precisely, we obtain the former if we prevent the TPDA from using the timed information (all the timing constraints are trivially valid); and obtain the latter if we prevent the TPDA from using the stack (no symbols are pushed or popped from the stack). Notice that a TPDA induces a system that is infinite in two dimensions, namely it gives rise to a stack containing an unbounded number of symbols each of which is equipped with a real-valued clock.

In this paper, we show decidability of the reachability problem for TPDA. We show the decidability through a reduction to the corresponding problem for (untimed) pushdown automata. Then, we prove that the reachability problem for TPDA is EXPTIME-complete.

Related Work: The works in [6], [12], [10], [11], [13] consider timed pushdown automata. However, the models in these works consider only global clocks which means that the stack symbols are not equipped with clocks.

In [18], 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. This makes the semantics of the models incomparable with ours, since all the clocks in our model evolve synchronously. In fact, the authors show decidability of the reachability problem only in the special cases where either all clocks are passed by reference or none is passed by reference.

In [4], 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 a recent work [1] we have shown decidability of the reachability problem for *discrete-timed* pushdown automata, where time is interpreted as being incremented in discrete steps

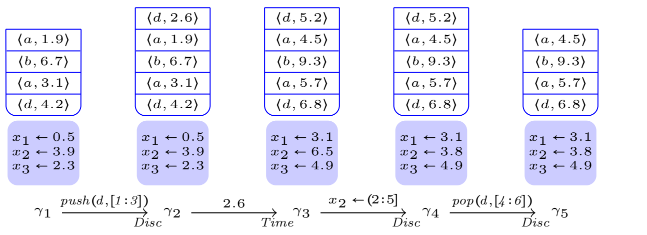

$\langle a,1.9\rangle$ $\langle b,6.7\rangle$ $\langle a,3.1\rangle$ $\langle d,4.2\rangle$
$x_1 \leftarrow 0.5$ $x_2 \leftarrow 3.9$ $x_3 \leftarrow 2.3$
$\langle d,2.6\rangle$ $\langle a,1.9\rangle$ $\langle b,6.7\rangle$ $\langle a,3.1\rangle$ $\langle d,4.2\rangle$
$x_1 \leftarrow 0.5$ $x_2 \leftarrow 3.9$ $x_3 \leftarrow 2.3$
$\langle d,5.2\rangle$ $\langle a,4.5\rangle$ $\langle b,9.3\rangle$ $\langle a,5.7\rangle$ $\langle d,6.8\rangle$
$x_1 \leftarrow 3.1$ $x_2 \leftarrow 6.5$ $x_3 \leftarrow 4.9$
$\langle d,5.2\rangle$ $\langle a,4.5\rangle$ $\langle b,9.3\rangle$ $\langle a,5.7\rangle$ $\langle d,6.8\rangle$
$x_1 \leftarrow 3.1$ $x_2 \leftarrow 3.8$ $x_3 \leftarrow 4.9$
$\langle a,4.5\rangle$ $\langle b,9.3\rangle$ $\langle a,5.7\rangle$ $\langle d,6.8\rangle$
$x_1 \leftarrow 3.1$ $x_2 \leftarrow 3.8$ $x_3 \leftarrow 4.9$
$\gamma_1 \xrightarrow[Disc]{push(d,[1:3])} \gamma_2 \xrightarrow[Time]{2.6} \gamma_3 \xrightarrow[Disc]{x_2 \leftarrow (2:5]} \gamma_4 \xrightarrow[Disc]{pop(d,[4:6])} \gamma_5$

Fig. 1. Configurations and transition in a TPDA.

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 [1] cannot be extended to the dense-time case.

II. Overview

In this section, we give an informal but detailed overview of the paper. We introduce Timed PushDown Automata (TPDA), together with its reachability problem. We describe a symbolic representation that allows us to translate a TPDA into an (untimed) PDA. We also describe how such a PDA can simulate the TPDA while preserving reachability properties.

TPDA: A TPDA is an automaton that operates on a finite set of real-valued clocks and where the symbols (taken from a finite alphabet) inside the stack are equipped with real numbers that indicate their ages. Fig. 1 gives examples of typical configurations in a TPDA $\mathcal{T}$ that has three clocks x_1, x_2, x_3 and that has a stack alphabet with three symbols $\{a, b, d\}$. A configuration of a TPDA consists of three components. The first component defines the local (control) state of the automaton (to simplify the illustration, this part is not shown in the figure). The second component defines the clock values, while the third component defines the content of the stack. For instance, in γ_1, the clock values are given by $[x_1 \leftarrow 0.5, x_2 \leftarrow 3.9, x_3 \leftarrow 2.3]$, while the stack contains four symbols, namely (from top to bottom): a, b, a, and d, with ages 1.9, 6.7, 3.1 and 4.2 respectively. Fig. 1 also illustrates different types of transitions that can be performed by a TPDA. From γ_1, $\mathcal{T}$ performs a *discrete transition* in which the symbol d is pushed to the stack. The transition requires that the age of the newly pushed symbol lies in the interval $[1:3]$ (indeed, the age of the new symbol is $2.6 \in [1:3]$). From the new configuration γ_2, $\mathcal{T}$ performs a *timed transition* to γ_3 in which the values of all clocks, and the ages of all symbols inside the stack are increased by the same real number 2.6. From γ_3, $\mathcal{T}$ moves to γ_4 by assigning a new value to the clock x_2. The new value assigned to x_2 should lie in the interval $(2:5]$ (the chosen value is 3.8). From γ_4, $\mathcal{T}$ pops the top-most symbol from the stack. The transition may only be performed if the age of the popped symbol lies in the interval $[4:6]$ (which is the case here).

Reachability Problem: An instance of the *reachability problem* for TPDA is defined by an initial configuration γ_{init} and a final (target) state s_F of the automaton. The task is to check whether there exists a sequence of transitions leading from γ_{init} to some configuration whose state is s_F. In this paper, we show decidability of the problem by reducing it to the corresponding problem for (untimed) pushdown automata (which is known to be decidable). The main ingredient of our proof is a symbolic representation for infinite sets of configurations in TPDA. Given a TPDA $\mathcal{T}$, we use the representation to extract a pushdown automaton $\mathcal{P}$, called the *symbolic* automaton, such that $\mathcal{P}$ can simulate $\mathcal{T}$ wrt. reachability properties in an exact manner.

Symbolic Encoding: A symbolic representation is needed even in the (simpler) case of timed automata, since they operate on real-valued clocks and hence induce infinite (in fact uncountable) state spaces. There, the classical *regions* encoding has been used to produce a finite-state abstraction that is exact wrt. many properties including reachability [3]. However, the region-based abstraction relies heavily on the fact that a timed automaton operates on a *finite* set of clocks. In particular, this means that it is not applicable in the case of TPDA, since the latter operates on an unbounded number of clocks (the stack is unbounded, and each symbol has an age). A difficult feature in the behavior of TPDA is that the ages of the symbols inside the stack, and their relations with the clock values, change continuously during the run of the automaton (due to timed transitions and clock resettings). In fact, sometimes it is crucial to record relations that arise between clocks and symbols that lie arbitrarily deep inside the stack. Simulating the behavior of a TPDA by an (untimed) pushdown automaton is not trivial, since in the latter, the symbols inside the stack do not change, and furthermore, the system can only access the top-most stack symbol. This makes it difficult to capture the evolving relations between the clocks and the stack symbols. The symbolic automaton, that we derive from $\mathcal{T}$, uses a stack alphabet in which each symbol corresponds to a region of a special form. The region relates, among other things, the top-most stack symbol with the clocks of the automaton. Furthermore, each region is enriched with information that is sufficient to capture the above mentioned dependencies between clocks and symbols that lie arbitrarily deep inside the stack. A key idea of our proof is to show that it is enough to enrich the regions in a finite way in order to capture all such dependencies. Roughly speaking, we add a copy of each clock and a copy of an extra stack symbol to the region representation, where the additional items carry (partial information) about the history of the current run of the system. This makes it possible to maintain a finite number of regions, and hence the symbolic automaton only uses a finite stack alphabet.

Below, we will describe some aspects of the problems and the solutions we provide, based on the example of Fig. 1. A typical example of a region (in the sense of timed automata) is R_1 in Fig. 2. Here, we represent a region as a word of sets, where each set contains a number of *items*. There are three types of items in a region: (i) the *plain items* x_1, x_2, x_3, a represent the three clocks and the top-most stack symbol, (ii) the special item $\vdash$ (introduced for technical reasons) is used as a reference clock whose value is 0 unless we are performing

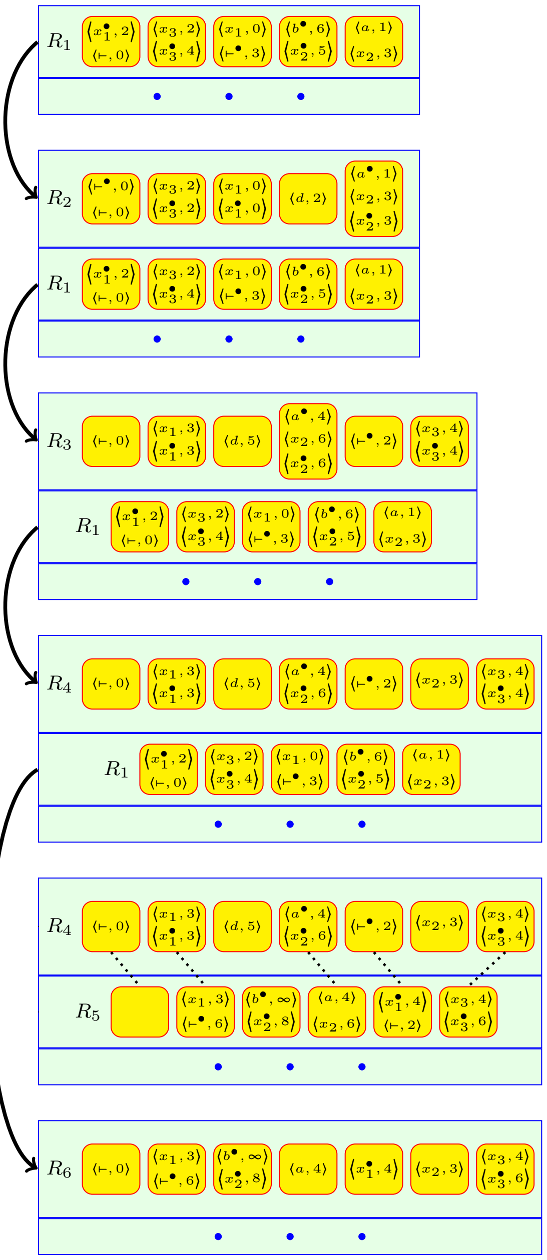


Fig. 2. A run in the symbolic automaton.

a *pop* operation (see below), and (iii) the *shadow items* $x_1^\bullet, x_2^\bullet, x_3^\bullet, b^\bullet, \vdash^\bullet$ are used for enriching the region and will be explained later. The items impose a number of conditions on any configuration satisfying R_1. An example of such a configuration is γ_1 in Fig. 1. Each item in R_1 is paired with a natural number that specifies the integral part of its value. Thus, R_1 requires for instance that the integral part of the value of x_1 should be 0; and that the top-most stack symbol should be a and the integral part of its value should be 1. Notice that these conditions are satisfied by γ_1, since the integral part the value of x_1 is 0, and that of a is 1. Items belonging to the same set should have identical fractional parts, and items belonging to successive sets should have strictly increasing fractional parts, so the fractional parts of a and x_2 should be identical and larger than that of x_1. All these conditions on the fractional parts are satisfied by γ_1. Finally, the left-most set plays a special role, in the sense that the fractional of the items in the left-most set ($x_1^\bullet$ and $\vdash$ here) should equal to 0.

Our aim is to derive a PDA $\mathcal{P}$ that is able to simulate $\mathcal{T}$. The stack alphabet of $\mathcal{P}$ is a set of regions of a special form. Next, we will use the run of $\mathcal{T}$, depicted in Fig. 1, to explain why we need and how we enrich the region representation to capture dependencies between clocks and stack symbols. We observe that in γ_1 the fractional parts of the clock x_2 and the (top-most) stack symbol a are equal (both of them are equal to 0.9). The fractional parts remain identical in γ_2, but a is no longer the top-most symbol and therefore its value is "not available" any more to the system. The equality remains in γ_3 although a has now obtained a new value inside the stack. However, the new value assigned to x_2 means that fractional parts of x_2 and a are no longer identical in γ_4. This information may become relevant in γ_5 where a has again become the top-most stack symbol. For instance, there may exist transitions whose enabledness may depend on whether the fractional parts of x_2 and a are different or identical (it is easy to encode such dependencies even in the case of timed automata). In fact, we can create examples showing that dependencies may exist among clocks and symbols that lie arbitrarily deep inside the stack. For instance, the clock x_2 may be assigned a new value after an arbitrary number of *push* operations rather than only one (as was the case above).

Simulation: We consider the computation of $\mathcal{T}$ from Fig. 1. In Fig. 2 we simulate it in the symbolic automaton $\mathcal{P}$ (whose stack alphabet consists of the set of regions as described above). Let us assume that we have already simulated the initial part of a computation leading to γ_1 and that the top-most stack symbol in $\mathcal{P}$ is R_1 (in the example we neglect the part of the stack below R_1). Intuitively the shadow $x_1^\bullet$ of a clock x_1 in the region R_1 represents the value of x_1 at the point of time when R_1 was pushed to the stack in $\mathcal{P}$. The shadow symbol $\vdash^\bullet$ represents the time that has elapsed since R_1 was pushed to the stack in $\mathcal{P}$. The shadow stack symbol $b^\bullet$ represents the current value of the next-top-most stack symbol.

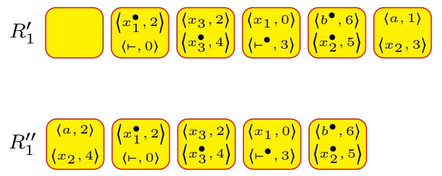
The *push* transition leading to γ_2 is simulated in $\mathcal{P}$ by pushing a new region R_2 that we derive from R_1 as follows. We identify each shadow clock with its plain counter-part (assign it the same integer value) and place it in the same set (e.g., x_2 and $x_2^\bullet$ have identical integral parts and are placed in the same set). This maintains the property that shadow clocks record the values of the plain clocks when the current region was pushed to the stack. For instance, $x_2^\bullet$ records the value of x_2 when R_2 is pushed to the stack in $\mathcal{P}$. From now on, the values of these shadow items are only updated through passage of time, and their values are not affected by discrete transitions that assign new values to the clocks. We also make a shadow copy $a^\bullet$ of the previous top-most stack symbol a. In other words, $a^\bullet$ records the value of a which is now the next top-most stack symbol. Finally, we add the new top-most stack symbol d into the region.

The timed transition from γ_2 to γ_3 is simulated in $\mathcal{P}$ as follows. A timed transition affects all the clocks and stack symbols. However, since we are dealing with a stack we can access only the top-most symbol. Therefore, we simulate the effect only on the top-most region (i.e., R_2) while we "freeze" the other regions inside the stack (those below R_2). This means that the items in R_1 no longer reflect the actual values (as we will see below, these values will later be recovered through the use of the shadow symbols). The effect of a timed transition on R_2 is simulated in $\mathcal{P}$ by popping R_2 and pushing a new region R_3. We derive R_3 from R_2 by (i) increasing the integral parts of the items and (ii) "rotating" the region left to right in order to obtain the correct ordering on the fractional parts. The item $\vdash$ is not affected in order to maintain the invariant that its value is zero. Since, we allow arbitrarily long time delays, the amount of rotation performed when simulating a timed transition is arbitrary. The rotation operation will be explained more in the simulation of the *pop* transition below.

The discrete transition from γ_3 to γ_4 is again simulated by updating the top-most region R_3 while freezing the regions below (including R_1). More precisely, we pop R_3 and push R_4 in which x_2 has been assigned a new integer and moved to a new position in the region in order to reflect its new value.

Simulating *pop* transitions is the most interesting step. First, we describe the *rotation* operation on regions (depicted in Fig. 3) that describes the manner in which a region changes due to the passage of time. The operation represents the next "interesting" event that occurs in R_1 when time elapses. There are two possible cases. The first case (which applies to R_1) is when there are some items with zero fractional parts (i.e., the left-most set in the region is not empty). The next event then is that the fractional values of these items become positive. We can obviously always choose the amount of time to be sufficiently small so that the value of none of the other items passes the next integer. For instance, in R_1, the items $x_2^\bullet$ and $\vdash$ leave the left-most set, meaning that there are no more items in the region with zero fractional parts. The result corresponds to R_1'. The second case (which applies to R_1') is when there are no items with integer values, and hence the operation corresponds to letting time pass by an amount that is exactly enough to make the values of the items with the highest fractional parts increase so that they reach the next integer (the integral parts of these items have now been incremented by one). In the case of R_1', this lead to R_1'' where the items a and x_2 have jumped to the left-most set, and their integral parts have been incremented by one.

The simulation of the *pop* transition leading from γ_4 to γ_5 is now performed in two steps. First, the next-top-most region R_1 is "refreshed", by repeatedly rotating it until its items are updated in a manner that reflects their current values (recall that these items were frozen while R_1 was not the top-most region). Concretely, we rotate R_1 sufficiently many times so that its information is consistent with that in R_4. The result is R_5. The plain items in R_5 should match their shadow



R_1'
$\langle x_1^\bullet, 2\rangle$
$\langle \vdash, 0\rangle$
$\langle x_3, 2\rangle$
$\langle x_3^\bullet, 4\rangle$
$\langle x_1, 0\rangle$
$\langle \vdash^\bullet, 3\rangle$
$\langle b^\bullet, 6\rangle$
$\langle x_2^\bullet, 5\rangle$
$\langle a, 1\rangle$
$\langle x_2, 3\rangle$
R_1''
$\langle a, 2\rangle$
$\langle x_2, 4\rangle$
$\langle x_1^\bullet, 2\rangle$
$\langle \vdash, 0\rangle$
$\langle x_3, 2\rangle$
$\langle x_3^\bullet, 4\rangle$
$\langle x_1, 0\rangle$
$\langle \vdash^\bullet, 3\rangle$
$\langle b^\bullet, 6\rangle$
$\langle x_2^\bullet, 5\rangle$

Fig. 3. Two successive rotations of R_1.

counter-parts in R_4 (shown by the dotted lines between the sets containing such items in R_4 and R_5). This means that the integral part of each item in R_5 is identical to its shadow counter-part in R_4 (for instance, this value is equal to 6 in the case of x_2 in R_5 and $x_2^\bullet$ in R_4). Also, the ordering of the plain items in R_5 should match the ordering of their shadow counter-parts in R_4 (for instance, $x_1^\bullet$ occurs before $x_2^\bullet$ in R_4 and hence x_1 should occur before x_2 in R_5). Notice that a dotted line connects the two left-most sets in order to take into considerations their special roles (collecting all items with zero fractional parts). In the second step, we pop both R_4 and R_5 and push a new region R_6 that we obtain by merging R_4 and R_5 as follows. The plain clock symbols in R_4 represent the current values of the clocks, and hence they are copied from R_4 to R_6 (these values are not affected by the *pop* operation). The age of the plain stack symbol a in R_5 represents the age of the next-top-most symbol (after popping, a will be at the top of the stack), and hence its value is copied from R_5 to R_6. All the shadow items are copied from R_5 to R_6 (again, the ages of these items are not affected by the *pop* operation). Since the ordering of fractional parts among the plain items in R_5 is identical to that of the shadow items in R_4, the ordering can be used to relate the fractional parts of the items copied from R_4 and R_5 to R_6. For instance, x_2 occurs before $x_3^\bullet$ in R_5 and x_1 occurs before $x_2^\bullet$ in R_4, and hence x_1 should occur before $x_3^\bullet$ in R_6 (items are not allowed to cross the dotted lines between the sets R_4 and R_5). Notice, for instance, that R_6 indicates correctly that x_2 and a have now different fractional parts. This relation was temporarily lost in the simulation, but has now been retrieved using the shadow items.

A detail taken from the standard representation of regions is that we can define an integer that is larger than all the constants that occur syntactically in the automaton. Item values larger than this constant behave equivalently and hence their integral parts can all be represented by a single symbolic value ∞ (e.g., $b^\bullet$ in R_5). In our particular example, we have assumed that this constant is equal to 9.

III. Preliminaries

We use $\mathbb{N}$ and $\mathbb{R}^{\geq 0}$ to denote the sets of natural numbers and non-negative reals respectively. We define $\mathbb{N}^\omega := \mathbb{N} \cup \{\omega\}$, where ω is the first limit ordinal. We use a set $\mathcal{I}$ of intervals. An open interval is written as $(a:b)$ where $a \in \mathbb{N}$ and $b \in \mathbb{N}^\omega$. Intervals can also be closed in one or both directions, e.g. $[a:b]$ is closed in both directions and $[a:b)$ is closed to the left and open to the right. For a number $v \in \mathbb{R}^{\geq 0}$ and interval $I \in \mathcal{I}$, we use $v \in I$ to indicate that v belongs to I. For a number $v \in \mathbb{R}^{\geq 0}$, we write $\lfloor v \rfloor$ and $fract(v)$ to denote the integral resp. fractional part of v. For $k \in \mathbb{N}$, we use $k^{(0)}$ and $k^{(1)}$ for the sets $\{0, 1, \ldots, k\}$ and $\{1, 2, \ldots, k\}$ respectively. The relation $\leq_{lex}$ is the standard lexicographic ordering on $\mathbb{N}^2$, i.e., $\langle k_1, \ell_1 \rangle \leq_{lex} \langle k_2, \ell_2 \rangle$ if either $k_1 < k_2$ or both $k_1 = k_2$ and $\ell_1 \leq \ell_2$ We write $\langle k_1, \ell_1 \rangle <_{lex} \langle k_2, \ell_2 \rangle$ to indicate that $\langle k_1, \ell_1 \rangle \leq_{lex} \langle k_2, \ell_2 \rangle$ and $\langle k_1, \ell_1 \rangle \neq \langle k_2, \ell_2 \rangle$.

For sets A and B, we use $f : A \to B$ to denote that f is a (possibly partial) function that maps A to B. We let $dom(f)$ and $range(f)$ denote the domain resp. range of f. For $a \in A$, we write $f(a) = \bot$ to indicate that f is not defined for a. By $f[a \leftarrow b]$ we mean the function f' such that $f'(x) = f(x)$ if $x \neq a$ and $f'(a) = b$. For a function f, with a finite domain, we sometimes write $f = [x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n]$ to denote that $f(x_i) = a_i$ for $i : 1 \leq i \leq n$, and that $f(y) = \bot$ if $y \notin \{x_1, \ldots, x_n\}$. For a set A, we write $|A|$ for the size of A, and write A^* for the set of finite words over A. For a word $w \in A^*$, we let $|w|$ denote the length of w, and let $w[i]$ denote the i^{th} element of w where $i : 1 \leq i \leq |w|$. The empty word is written as ϵ. For words $w_1, w_2 \in A^*$, we write $w_1 \cdot w_2$ to denote the concatenation of w_1 and w_2. For sets W_1, W_2 of words, we define $W_1 \cdot W_2 := \{w_1 \cdot w_2 | (w_1 \in W_1) \land (w_2 \in W_2)\}$.

In this paper, we will often use a number of operations on words. First, we consider words over sets over an alphabet A, i.e., members of the set $\left(2^A\right)^*$ and define a shuffle operator $\otimes$ inductively as follows. For a word $w \in \left(2^A\right)^*$, we define $w \otimes \epsilon := \epsilon \otimes w := w$. Furthermore, for sets $r_1, r_2 \in 2^A$ and words $w_1, w_2 \in \left(2^A\right)^*$ we define $(r_1 \cdot w_1) \otimes (r_2 \cdot w_2) := (r_1 \cdot (w_1 \otimes (r_2 \cdot w_2))) \cup (r_2 \cdot (r_1 \cdot w_1 \otimes w_2)) \cup ((r_1 \cup r_2) \cdot (w_1 \otimes w_2))$. *Example:* Given the words $w_1 = \{a, b\} \{c\} \{d, e\} \{f\}$ and $w_2 = \{u, v\} \{v\} \{y, z\}$, we have that $\{a, b, u, v\} \{c\} \{d, e, v\} \{y, z\} \{f\} \in w_1 \otimes w_2$.

Consider words $w = a_1 \cdots a_m$ and $w' = b_1 \cdots b_n$ in A^*. An *injection* from w to w' is a partial function $h : m^{(1)} \to n^{(1)}$ that is strictly monotonic, i.e., if $i < j$ and $h(i), h(j) \neq \bot$ then $h(i) < h(j)$. The *fragmentation* w/h of w wrt. h is the sequence $\langle w_0 \rangle a_{i_1} \langle w_1 \rangle a_{i_2} \cdots \langle w_{k-1} \rangle a_{i_k} \langle w_k \rangle$ where $dom(h) = \{i_1, i_2, \ldots, i_k\}$ and $w = w_0 \cdot a_{i_1} \cdot w_1 \cdot a_{i_2} \cdot \cdots \cdot w_{k-1} \cdot a_{i_k} \cdot w_k$. Similarly, the *fragmentation* w'/h is the sequence $\langle w'_0 \rangle b_{j_1} \langle w'_1 \rangle b_{j_2} \cdots \langle w'_{\ell-1} \rangle b_{j_\ell} \langle w'_\ell \rangle$ where $range(h) = \{j_0, j_1, \ldots, j_\ell\}$ and $w' = w'_0 \cdot b_{j_1} \cdot w'_1 \cdot b_{j_2} \cdot \cdots \cdot w'_{\ell-1} \cdot b_{j_\ell} \cdot w'_\ell$. *Example:* If $w = abcdefgh$, $w' = rstuvxyz$, $h(3) = 2$, $h(6) = 5$ and $h(i) = \bot$ for $i \in \{1, 2, 4, 5, 7, 8\}$ then $w/h = \langle ab \rangle c \langle de \rangle f \langle gh \rangle$ and $w'/h = \langle r \rangle s \langle tu \rangle v \langle xyz \rangle$.

IV. Model

We recall the standard model of pushdown automata, and then describe its timed extension. We give the operational semantics by defining the induced transition system, i.e., the set of configurations, and the transition relation on the set of configurations. Then, we describe the reachability problem in which we ask whether a given local state of the automaton is reachable from the initial configuration of the system.

PDA: We recall the classical model of *PushDown Automata* (*PDA* for short) . A *PDA* $\mathcal{P}$ is a tuple $\langle S, s_{init}, \Gamma, \Delta \rangle$, where S is a finite set of *states*, $s_{init} \in S$ is the *initial state*, Γ is a (finite) set of *stack symbols*, and Δ is a finite of *transitions*. A transition $t \in \Delta$ is a triple $\langle s, op, s' \rangle$ where $s, s' \in S$ are the source and target states of the transition, and op is a *stack operation* of one of three forms: (i) nop is an *empty* operation that does not change the content of the stack, (ii) $pop(a)$, where $a \in \Gamma$, is a *pop* operation that removes the top-most stack symbol if this symbol is equal to a, and (iii) $push(a)$, where $a \in \Gamma$, is a *push* operation that adds a to the top of the stack. A *configuration* β is a pair $\langle s, w \rangle$ where $s \in S$ is the local (control) state of the automaton and $w \in \Gamma^*$ is the content of the stack. We define the transition relation $\longrightarrow$ on the set of configurations as follows. For configurations $\beta = \langle s, w \rangle$ and $\beta' = \langle s', w' \rangle$ and a transition $t = \langle s, op, s' \rangle \in \Delta$, we write $\beta \xrightarrow{t} \beta'$ to denote that one of the following properties is satisfied: (i) $op = nop$ and $w' = w$, (ii) $op = push(a)$ and $w' = w \cdot a$, or (iii) $op = pop(a)$ and $w = w' \cdot a$. We define $\longrightarrow := \cup_{t \in \Delta} \xrightarrow{t}$ and define $\xrightarrow{*}$ to be the reflexive transitive closure of $\longrightarrow$. The *initial configuration* is defined by $\beta_{init} := \langle s_{init}, \epsilon \rangle$, i.e., the system starts from the initial state and with an empty stack. A configuration β is said to be *reachable* if $\beta_{init} \xrightarrow{*} \beta$. A local state $s \in S$ is said to be *reachable* if there is a stack content w such that the configuration $\langle s, w \rangle$ is reachable. An instance of the *reachability problem* is defined by a (target) local state s_F. The task is to check whether s_F is reachable, i.e., to check whether we can reach a configuration where the local state of the automaton is equal to s_F (regardless of the stack content).

TPDA: Assume a finite set X of *clocks*. A *Timed PushDown Automaton* (*TPDA* for short) is a tuple $\mathcal{T} = \langle S, s_{init}, \Gamma, \Delta \rangle$, where S is a finite set of *states*, $s_{init} \in S$ is the *initial state*, Γ is a (finite) set of *stack symbols*, and Δ is a finite set of *transitions*. A transition $t \in \Delta$ is a tuple $\langle s, op, s' \rangle$ where $s, s' \in S$, and op is a *stack operation* of one of five forms: (i) nop is an *empty* operation that does not change the contents of the stack, (ii) $x \in I?$, where $x \in X$ is a clock and $I \in \mathcal{I}$ is an interval, is a *test* operation where the transition may be fired only if the value of x belongs to I, (iii) $x \leftarrow I$, where $x \in X$ is a clock and $I \in \mathcal{I}$ is an interval, is an *assignment* operation where the clock x is non-deterministically assigned an arbitrary value in I, (iv) $pop(a, I)$, where $a \in \Gamma$ is a stack symbol and $I \in \mathcal{I}$ is an interval, is a *pop* operation that removes the top-most stack symbol provided that this symbol is a and its age belongs to the interval I, and (v) $push(a, I)$, where $a \in \Gamma$ is a stack symbol and $I \in \mathcal{I}$ is an interval, is a *push* operation that adds a to the top of the stack, such that the age of the newly added symbol is in the interval I.

Configurations: A *clock valuation* is a function $\mathtt{X} : X \to \mathbb{R}^{\geq 0}$ that assigns a real number to each clock. A *stack content* is a word $w \in \left(\Gamma \times \mathbb{R}^{\geq 0}\right)^*$ of pairs each defining a symbol and its age inside the stack. A configuration γ is of the form $\langle s, \mathtt{X}, w \rangle$ where $s \in S$, $\mathtt{X}$ is a clock valuation, and w is a *stack content*.

Transition Relation: We first define timed transitions. Consider a real number $v \in \mathbb{R}^{\geq 0}$. For a clock valuation $\mathtt{X}$, we define $\mathtt{X}^{+v}$ to be the clock valuation $\mathtt{X}'$ such that

$\mathtt{X}'(x) = \mathtt{X}(x) + v$ for all clocks $x \in X$. For a stack content $w = \langle a_1, k_1\rangle \langle a_2, k_2\rangle \cdots \langle a_n, k_n\rangle$, we define w^{+v} to be the stack content $w' = \langle a_1, k_1 + v\rangle \langle a_2, k_2 + v\rangle \cdots \langle a_n, k_n + v\rangle$. For configurations $\gamma = \langle s, \mathtt{X}, w\rangle$ and $\gamma' = \langle s', \mathtt{X}', w'\rangle$, we have $\gamma \overset{v}{\rightsquigarrow}_{Time} \gamma'$ if $s' = s$, $\mathtt{X} = \mathtt{X}^{+v}$, and $w' = w^{+v}$. The system makes a timed transition (of length v) to γ'. The local state of the automaton and the symbols inside the stack are not changed. The values of the clocks and the ages of the stack symbols are all increased by v. We write $\gamma \rightsquigarrow_{Time} \gamma'$ to denote that $\gamma \overset{v}{\rightsquigarrow}_{Time} \gamma'$ for some $v \in \mathbb{R}^{\geq 0}$.

Now, we define *discrete transitions*. Let $t = \langle s, op, s'\rangle \in \Delta$ be a transition. For configurations $\gamma = \langle s, \mathtt{X}, w\rangle$ and $\gamma' = \langle s', \mathtt{X}', w'\rangle$, we write $\gamma \overset{t}{\rightsquigarrow}_{Disc} \gamma'$ if one of the following conditions is satisfied:

- $op = nop$, $w' = w$, $\mathtt{X}' = \mathtt{X}$. The empty operation does not modify the clock values or the stack content.
- $op = x \in I?$, $w' = w$, $\mathtt{X}' = \mathtt{X}$, and $\mathtt{X}(x) \in I$ holds. The transition can be performed only if the value of clock x lies in the interval I. The clock values and the stack content are not changed.
- $op = x \leftarrow I$, $w' = w$, and $\mathtt{X}' = \mathtt{X}[x \leftarrow v]$ where $v \in I$. The clock x is assigned an arbitrary value in I. The values of the rest of clocks and the stack content are not changed.
- $op = pop(a, I)$, $\mathtt{X}' = \mathtt{X}$, and $w = w' \cdot \langle a, v\rangle$ for some $v \in I$. The operation checks whether the top-most symbol in the stack is a and whether its age value is in I. In such a case it removes the top-most stack symbol. The clock values are not changed.
- $op = push(a, I)$, $\mathtt{X}' = \mathtt{X}$, and $w' = w \cdot \langle a, v\rangle$ for some $v \in I$. The operation places a at the top of the stack and defines its age to be some (arbitrary) value v in I. The clock values are not changed.

Notice that the local states of γ and γ' agree with the source and target local states in t. We define $\rightsquigarrow_{Disc} := \cup_{t \in \Delta} \overset{t}{\rightsquigarrow}_{Disc}$. We define the transition relation $\rightsquigarrow := \rightsquigarrow_{Time} \cup \rightsquigarrow_{Disc}$ and define $\overset{*}{\rightsquigarrow}$ to be the reflexive transitive closure of $\rightsquigarrow$.

Reachability: We fix a clock valuation $\mathtt{X}_{init}$ defined by $\mathtt{X}_{init}(x) := 0$ for all $x \in X$. We define the *initial configuration* $\gamma_{init} := \langle s_{init}, \mathtt{X}_{init}, \epsilon\rangle$. In other words, the system starts running from a configuration where the automaton is in its initial local state, where all clocks have values 0, and where the stack is empty. A configuration γ is said to be *reachable* if $\gamma_{init} \overset{*}{\rightsquigarrow} \gamma$. A (target) local state s_F is said to be *reachable* if there is a clock valuation $\mathtt{X}$ and a stack content w such that the configuration $\langle s_F, \mathtt{X}, w\rangle$ is reachable. An instance of the *Reachability Problem* is defined by a final (target) state $s_F \in S$. The task is to check whether s_F is reachable.

V. Symbolic Encoding

Given a TPDA $\mathcal{T} = \left\langle S^{\mathcal{T}}, s^{\mathcal{T}}_{init}, \Gamma^{\mathcal{T}}, \Delta^{\mathcal{T}}\right\rangle$, we will show how we can simulate $\mathcal{T}$ by an (untimed) PDA $\mathcal{P} = \left\langle S^{\mathcal{P}}, s^{\mathcal{P}}_{init}, \Gamma^{\mathcal{P}}, \Delta^{\mathcal{P}}\right\rangle$. Sometimes, we refer to the latter as the *symbolic automaton*. The simulation relies on a symbolic encoding that $\mathcal{P}$ uses for representing, in a finite way, the (infinite set of) configurations in $\mathcal{T}$.

A. Regions

Each symbol in the stack of $\mathcal{P}$ is a *region*. We define a set of *items* that we use to build regions, then define their syntax and semantics.

Items: A region contains two sets of items, namely *plain items* and *shadow items*. The plain items include (i) one copy of each clock in $\mathcal{T}$, used to describe the value of the clock, (ii) exactly one symbol from the stack alphabet of $\mathcal{T}$, used to describe the name and the age of the top-most symbol in the stack of $\mathcal{T}$, and (iii) a fresh symbol $\vdash$ that is used as a reference clock whose value is 0 unless we are simulating a *pop* operation. Similarly, the shadow symbols include a copy of each clock together with a copy of exactly one stack symbol, and one copy $\vdash^\bullet$of $\vdash$. The shadow clocks record the values of the clocks at the point where a *push* operation is performed. Their values are then updated only through timed transitions. The same applies for $\vdash^\bullet$ The shadow stack symbol reflects the value of the next-top-most symbol in the stack. As in the classical definition of regions, items are abstracted by storing their integral parts, and the relative ordering of the fractional parts. This is done up to a certain constant (defined below).

Let $\Gamma := \Gamma^{\mathcal{T}} \cup \{\mathtt{bottom}\}$ where $\mathtt{bottom} \notin \Gamma^{\mathcal{T}}$ is a special symbol indicating the bottom of the stack. Define the set $Y := X \cup \Gamma \cup \{\vdash\}$ of *plain items*, and define the sets of *shadow clocks* $X^\bullet := \{x^\bullet | \, x \in X\}$, *shadow stack symbols* $\Gamma^\bullet := \{a^\bullet | \, a \in \Gamma\}$, and *shadow items* $Y^\bullet := X^\bullet \cup \Gamma^\bullet \cup \{\vdash^\bullet\}$. Define the set of *items* $Z := Y \cup Y^\bullet$.

Syntax: Let c_{max} be the largest natural number that occurs syntactically in the definition of $\mathcal{T}$, i.e., c_{max} is the largest natural number that appears in an interval that is used in the definition of a *testing*, *assignment*, *pop*, or *push* transition in $\Delta^{\mathcal{T}}$. Define the set $Max = \{0, 1, \ldots, c_{max}, \infty\}$, i.e., Max contains all natural numbers up to c_{max} together with a special constant ∞. A *region* R is a word $r_1 r_2 \cdots r_n \in (2^{Z \times Max})^+$ such that the following conditions are satisfied (below, let $r := \cup_{1 \leq i \leq n} r_i$):

- $|(\Gamma \times Max) \cap r| = 1$ and $|(\Gamma^\bullet \times Max) \cap r| = 1$, i.e., there is exactly one occurrence of a stack symbol and one occurrence of a shadow stack symbol in the region.
- $|(\{\vdash\} \times Max) \cap r| = 1$ and $|(\{\vdash^\bullet\} \times Max) \cap r| = 1$, i.e., there is exactly one occurrence of $\vdash$ and one occurrence of the shadow symbol $\vdash^\bullet$ in the region.
- $|(\{x\} \times Max) \cap r| = 1$ and $|(\{x^\bullet\} \times Max) \cap r| = 1$, for all clocks $x \in X$, i.e., each clock and each shadow clock occurs exactly once in the region.
- $r_i \neq \varnothing$ for all $i : 2 \leq i \leq n$, i.e., the sets (except possibly r_1) are not empty.

For $x \in X \cup X^\bullet \cup \{\vdash, \vdash^\bullet\}$, consider the unique $k \in Max$ and $i : 1 \leq i \leq n$ such that $\langle x, k\rangle \in r_i$. We define $Val(R)(x) := k$ and define $Index(R)(x) := i$. For $a \in \Gamma \cup \Gamma^\bullet$, we define $Val(R)(a)$ and $Index(R)(a)$ in a similar manner except that it may be the case that $Val(R)(a) = \bot$ and $Index(R)(a) = \bot$ (in case a is missing from R). We define $R^\top := \{z \in Z | \, Index(R)(z) \neq \bot\}$. In other words, it gives the set of items that occur in R. Notice that $X \cup X^\bullet \cup \{\vdash, \vdash^\bullet\} \subseteq R^\top$,

$|R^\top \cap \Gamma| = 1$ and $|R^\top \cap \Gamma^\bullet| = 1$. Sometimes, abusing notation, we write $z \in r_i$ to indicate that there is a $k \in Max$ such that $\langle z,k\rangle \in r_i$; and, for a set A of items we write $r_i \cap A$ for the set $\{\langle z,k\rangle \mid \langle z,k\rangle \in r_i \wedge z \in A\}$.

Semantics: A *valuation* of a region R is a total function $\theta : R^\top \to \mathbb{R}^{\geq 0}$. Consider a region $R = r_1 r_2 \cdots r_n$ and a valuation θ of R. We write $\theta \models R$ to denote that the following conditions are satisfied for all $z, z_1, z_2 \in R^\top$:

- $\theta(z) \geq c_{max} + 1$ iff $Val(R)(z) = \infty$. Values larger than or equal to $c_{max}+1$ are all abstracted to ∞ in the region.
- If $\theta(z) < c_{max} + 1$ then $\lfloor \theta(z) \rfloor = Val(R)(z)$. The region stores only the integral parts of the item values.
- $fract(\theta(z_1)) \leq fract(\theta(z_2))$ iff $Index(R)(z_1) \leq Index(R)(z_2)$. The ordering of the sets in the region reflects the ordering of the fractional parts of the items in these sets.
- $fract(\theta(z)) = 0$ iff $Index(R)(z) = 1$. The items in the first set are those with integer values.

We define $[\![R]\!] := \{\theta \mid \theta \models R\}$.
Example. Consider R_1 in Figure 2. We have $|R_1| = 5$, $x_3 \in r_2$, $Index(R_1)(x_3) = 2$, $Val(R_1)(x_2^\bullet) = 5$, $R_1^\top \cap \Gamma = \{a\}$, $R_1^\top \cap \Gamma^\bullet = \{b^\bullet\}$. Furthermore, we have $\theta \models R_1$ where $\theta = [x_1 \leftarrow 0.5, x_2 \leftarrow 3.9, x_3 \leftarrow 2.3, x_1^\bullet \leftarrow 2.0, x_2^\bullet \leftarrow 5.7, x_3^\bullet \leftarrow 4.3, a \leftarrow 1.9, b^\bullet \leftarrow 6.7, \vdash \leftarrow 0.0, \vdash^\bullet \leftarrow 3.5]$.

B. Operations on Regions

We define a number of operations on regions that we use to describe how we perform the simulation.

Satisfiability: For an item $z \in Z$, an interval $I \in \mathcal{I}$, and a region R with $z \in R^\top$, the operation checks that the value of z in R lies in I. More precisely, we write $R \models (z \in I)$ iff one of the following three conditions is satisfied: (i) $Index(R)(z) = 1$, $Val(R)(z) \neq \infty$, and $Val(R)(z) \in I$. (ii) $Index(R)(z) > 1$, $Val(R)(z) \neq \infty$, and $(Val(R)(z) + v) \in I$ for any $v \in \mathbb{R}^{\geq 0} : 0 < v < 1$. (iii) $Val(R)(z) = \infty$ and I is of the form $(k : \infty)$ or of the form $[k : \infty)$. If the fractional part of z is zero then the test is equivalent to whether the integral part of z lies in I. Otherwise, the test is equivalent to whether the integral part of z, increased by some arbitrary real number $v : 0 < v < 1$, lies in the interval.
Example. In the example of Figure 2, we have that $R_1 \models x_3 \in (2,5)$, $R_1 \not\models x_3 \in (4,5)$, $R_1 \models b^\bullet \in [2,\infty)$, $R_1 \not\models b^\bullet \in [2,5]$.

Assignment: The following operation describes the effect of assigning a new value to an item z in a region. The operation is used in simulating the assignment of a new value to a clock. We define the operation in two steps, namely by first deleting the item from the region, and then re-introducing it with its new value. First, we define an operation that deletes an item from a region. Consider a region $R = r_1 \cdots r_n$ and an item $z \in R^\top$ where $Index(R)(z) = i$. We define $R \ominus z$ to be the (unique) word $R' \in (2^{Z \times Max})^+$ satisfying one of the following conditions:

- $i > 1$, $|r_i| = 1$, and $R' = r_1 \cdots r_{i-1} r_{i+1} \cdots r_n$. If the set r_i is not the left-most set, and it becomes empty after removing the item then we delete r_i. This is done in order to maintain the invariant that all sets except (possibly) the left-most one are non-empty.
- $R' = r_1 \cdots r_{i-1} (r_i - \{\langle z,k\rangle\}) r_{i+1} \cdots r_n$, otherwise. Here $k = Val(R)(z)$

Next, we define an operation that adds an item. For a word $R = r_1 \cdots r_n \in (2^{Z \times Max})^+$, item $z \in Z$, and $k \in Max$, we define $R \oplus \langle z,k\rangle$ to be the set of words R' either of the form $r_1 \cdots r_{i-1} \{\langle z,k\rangle\} r_i \cdots r_n$ where $i : 2 \leq i \leq n+1$, or of the form $r_1 \cdots (r_i \cup \{\langle z,k\rangle\}) \cdots r_n$ where $i : 1 \leq i \leq n$. In other words, we insert the pair $\langle z,k\rangle$ somewhere in the word, either by inserting it between two sets, or by adding it to one set. For a region R_1, and an item $z \in Z$ with $z \in R_1^\top$, we define $R_1[z \leftarrow I]$ to be the set of regions R_2 such that:

- There is an R_3 and a $k \in Max$ such that $R_3 = R_1 \ominus z$, and $R_2 \in R_3 \oplus \langle z,k\rangle$, i.e., we get R_3 by first deleting z and then re-introducing it with a new value (possibly in a different position inside the region).
- $R_2 \models (z \in I)$. The new value of z should belong to I.

The above conditions imply that $R_2^\top = R_1^\top$. The operation, as defined, amounts to keeping the values of all the items, except z which is assigned a new value in I.
Example. Consider the region R_3 in Figure 2. Then $R_4 \in R_3[x_2 \leftarrow (2:5]]$

Passage of Time: To simulate timed transitions, we define an operation that describes the effect of the passage of time on regions. For this, we need a number of definitions. For a pair $\langle z,k\rangle \in (Z \times Max)$, we define $\langle z,k\rangle^+ := \langle z,k'\rangle$ where $k' = k+1$ if $k < c_{max}$ and $k' = \infty$ otherwise. For a set $r \in 2^{Z \times Max}$, we define $r^+ := \{\langle z,k\rangle^+ \mid \langle z,k\rangle \in r\}$. The operation increases the integral parts of the clock values by one up to c_{max}. Consider a region $R = r_1 r_2 \cdots r_n$. We define $R^+ := R'$ where R' satisfies one of the following two conditions:

- $r_1 \neq \emptyset$ and $R' = \emptyset r_1 r_2 \cdots r_n$.
- $r_1 = \emptyset$ and $R' = r_n^+ r_1 \cdots r_{n-1}$.

We write $R' \in R^{++}$, to denote that there are regions $R_0, \ldots, R_n$ such that $R_0 = R$, $R_n = R'$, and $R_{i+1} = R_i^+$ for all $i : 0 \leq i < n$. We also define $R_\vdash^+$ to be the region R' such that there are R_1 and R_2 satisfying the following properties:

- $R_1 = R \ominus \vdash$, i.e., we get R_1 from R by deleting the symbol $\vdash$.
- $R_2 = R_1^+$, i.e., we obtain R_2 by letting time elapse.
- $R' \in R_2 \oplus \langle \vdash, 0\rangle$ and $R' \models (\vdash \in [0,0])$, i.e., we re-introduce the symbol $\vdash$ s.t. its value in R' is 0 and such that it is placed in the left-most set of R'. Notice that R' is unique.

We extend $R_\vdash^+$ to $R_\vdash^{++}$ in a similar manner to above.
Example. In Fig. 2 and Fig. 3, we have that $R_1' = R_1^+$, $R_5 \in R_1^{++}$, and $R_3 \in (R_2)_\vdash^{++}$.

Product: The product operation, denoted $\odot$, "merges" the information in two regions. The operation is used in the simulation of *pop* transitions in which the top-most stack symbol is removed and its information merged with the next symbol in the stack. The operation can be performed only under the assumption that the two regions are consistent in the sense that each plain item in P should "match" its shadow counter-part in Q. More precisely, for regions $P = p_1 p_2 \cdots p_{n_P}$

and $Q = q_1 q_2 \cdots q_{n_Q}$, and an injection h from P to Q (recall the definition of an *injection* from Section III), we write $P \preceq_h Q$ if the following conditions are satisfied:

- $Val(Q)(y^\bullet) = Val(P)(y)$ for all $y \in P^\top \cap Y$.
- For every $i > 1$, $h(i) \neq \perp$ iff there is a $y \in Y$ such that $Index(P)(y) = i$.
- $h(1) = 1$.
- If $Index(P)(y) = i$ and $Index(Q)(y^\bullet) = j$ then $h(i) = j$.

We say that P *supports* Q, denoted $P \preceq Q$, if $P \preceq_h Q$ for some h. Notice that, for any P, Q, there is at most one h for which $P \preceq_h Q$. Let $P/h = p_{i_1} \langle P_1 \rangle p_{i_2} \cdots p_{i_m} \langle P_m \rangle$, and let $Q/h = q_{j_1} \langle Q_1 \rangle q_{j_2} \cdots q_{j_m} \langle Q_m \rangle$. Define $p'_k := p_{i_k} \cap (Y^\bullet \cup \Gamma)$, and $q'_k := q_{j_k} \cap (X \cup \{\vdash\})$. Define $r_1 := p'_1 \cup q'_1$, and for $k : 2 \leq k \leq m$, define $r_k := p'_k \cup q'_k$ if $p'_k \cup q'_k \neq \emptyset$, and $r_k := \epsilon$ if $p'_k \cup q'_k = \emptyset$, Then, $R \in P \odot Q$ if $R = r_1 \cdot R_1 \cdot r_2 \cdots r_m \cdot R_m$ and $R_k \in P_k \otimes Q_k$ for $k : 1 \leq k \leq m$.

For regions P, Q, we define $P * Q := \{P' \odot Q | \, P' \in P^{++} \wedge P' \preceq Q\}$. In other words, we let time pass on P until it supports Q after which we compute their product.

Example. In Fig. 2, we have that $R_5 \preceq R_4$, $R_6 \in R_5 \odot R_4$, and $R_6 \in R_1 * R_4$.

Resetting: The operation is used when describing the simulation of *push* operations. In $\mathcal{P}$ we add a new region to the top of the stack. The operation resets the shadow clocks and $\vdash^\bullet$ in the sense that it forgets their previous values and instead makes their values identical to the corresponding plain clocks. In other words, the value of each $x^\bullet$ will now be made equal to the value of x. The new shadow clocks (which record the values of the clocks when the push operation was made) should therefore be equal to their plain counter-parts. Furthermore, we add a new plain symbol to the stack whose age should be in the interval specified by the *push* operation. We first extend the operation $\ominus$ that deletes items (see the text on variable assignment) to sets of items as follows. For a region R and a set $A = \{z_1, \ldots, z_n\} \subseteq R^\top$ we define $R \ominus A := (\cdots((R \ominus z_1) \ominus z_2) \cdots) \ominus z_n$, i.e., it is the region we get by deleting from R all the items in A. For a region R_1, a stack symbol $a \in \Gamma$, and an interval $I \in \mathcal{I}$, we define $Reset(R_1)[a \leftarrow I]$ to be the set of regions R_2 such that there are $R_3 = r_1 \ldots r_n$, R_4, and R_5 satisfying the following properties:

- $R_3 = R_1 \ominus (R_1^\top \cap Y^\bullet)$, i.e., we get R_3 by deleting all the shadow symbols from R_1.
- $R_4 = r'_1 \cdots r'_n$ where $r'_i = r_i \cup \{\langle y^\bullet, k \rangle | \, \langle y, k \rangle \in r_i\}$ for $i : 1 \leq i \leq n$. In other words, for each plain item we add the shadow counter-part with an identical value and index.
- $R_5 = R_4 \ominus b$ where $b \in R_1^\top \cap \Gamma$. We remove the (only) plain stack symbol in R_1 (its shadow has already been copied in the previous step).
- $R_2 \in R_5 \oplus \langle a, k \rangle$ and $R_2 \models (a \in I)$, i.e., we add the new plain stack symbol such that its value and index reflect that its age belongs to I.

Example. $R_2 \in Reset(R_1)[d \leftarrow [1:3]]$ in Figure 2,

VI. Simulation

Fix a TPDA $\mathcal{T} = \langle S^\mathcal{T}, s^\mathcal{T}_{init}, \Gamma^\mathcal{T}, \Delta^\mathcal{T} \rangle$ with a set X of clocks. We will show how we can simulate $\mathcal{T}$ by an (untimed) PDA $\mathcal{P} = \langle S^\mathcal{P}, s^\mathcal{P}_{init}, \Gamma^\mathcal{P}, \Delta^\mathcal{P} \rangle$. We describe the construction of $\mathcal{P}$ in different steps. First, we define the set of states $S^\mathcal{P}$. Then, we describe transitions that carry out an initialization phase in $\mathcal{P}$. Finally, we give sets of transitions in $\mathcal{P}$ that are used to simulate both timed transitions and different types of discrete transitions (depending on the involved operation).

States: For each state $s \in S^\mathcal{T}$ there is a copy of s in $S^\mathcal{P}$. These states are called *genuine* states. Furthermore, the set $S^\mathcal{P}$ contains a number of *temporary* states that are used in the simulation. Each transition of $\mathcal{T}$ is simulated in $\mathcal{P}$ in a number of steps. To simplify the notation, we write a temporary state in the form $\texttt{tmp}(\cdot, \cdot)$ where the arguments indicate the transition in $\Delta^\mathcal{T}$ we are currently simulating and the number of steps we have performed in this simulation. A configuration $\beta = \langle s, w \rangle$ in $\mathcal{P}$ is said to be *genuine* resp. *temporary* if s is *genuine* resp. *temporary*. The simulation starts form the distinguished initial state $s^\mathcal{P}_{init}$ (which is considered to be a temporary state).

Initialization: We define the *initial region* $R_{init} := \{\langle x, 0 \rangle | \, x \in X \cup X^\bullet \cup \{\vdash, \vdash^\bullet\}\} \cup \{\langle \texttt{bottom}, 0 \rangle, \langle \texttt{bottom}^\bullet, 0 \rangle\}$. In other words, all plain/shadow clocks and the symbols $\{\vdash, \vdash^\bullet\}$ have initial values equal to 0. Furthermore, the region contains $\texttt{bottom}$ indicating the bottom of the stack. In fact, the value of $\texttt{bottom}$ is never used in the simulation, so taking its initial value to be 0 is an arbitrary decision. The same applies to the values of the shadow clocks and $\vdash^\bullet$. Also, the shadow $\texttt{bottom}^\bullet$ of the bottom symbol is not used in the simulation (we include only to preserve the invariant that a region contains a shadow stack symbol). Notice that R_{init} is a word of length one (it contains a single set). Now, the set $\Delta^\mathcal{P}$ contains a transition $\langle s^\mathcal{P}_{init}, push(R_{init}), s^\mathcal{T}_{init} \rangle$. This transition pushes the region indicating the bottom of the stack, and moves from the initial state $s^\mathcal{P}_{init}$ of $\mathcal{P}$ to the state $s^\mathcal{T}_{init}$ from which the simulation of $\mathcal{T}$ is started.

Timed Transitions: For each region R, and state $s \in S^\mathcal{T}$, the set $S^\mathcal{P}$ contains a state $\texttt{tmp}(timed, s, R)$ and $\Delta^\mathcal{P}$ contains the transitions $\langle s, pop(R), \texttt{tmp}(timed, s, R) \rangle$ and $\langle \texttt{tmp}(timed, s, R), push(R^+_\vdash), s \rangle$. In other words, we let time pass on the top-most region R in the stack by popping it and replacing by the region R^+. We also keep $\vdash$ in the left-most position of the top-most region. We can simulate the passage of an arbitrary amount of time by repeatedly firing the above two transitions. Notice that we simulate the effect of timed transition only on the top-most region in the stack.

nop: For each transition $\langle s, nop, s' \rangle \in \Delta^\mathcal{T}$, the set $\Delta^\mathcal{P}$ contains the transition $\langle s, nop, s' \rangle$. Since the empty operation only changes the local state of $\mathcal{T}$ it is simulated in a straightforward manner in $\mathcal{P}$.

$x \in I?$: For each transition $\langle s, x \in I?, s' \rangle \in \Delta^\mathcal{T}$ and region R such that $R \models (x \in I)$, the set $S^\mathcal{P}$ contains the state $\texttt{tmp}(t, R)$, and $\Delta^\mathcal{P}$ contains the transitions

$\langle s, pop(R), \mathtt{tmp}(t,R)\rangle$ and $\langle \mathtt{tmp}(t,R), push(R), s'\rangle$. The enabledness of the transition is checked by first popping the region (to check that the condition is satisfied), and then pushing it back to the stack. Since neither the clock values nor the stack content is affected in $\mathcal{T}$, the stack content in $\mathcal{P}$ is not affected.

$x \leftarrow I$: For each transition $\langle s, x \leftarrow I, s'\rangle \in \Delta^{\mathcal{T}}$ and region R, the set $S^{\mathcal{P}}$ contains the state $\mathtt{tmp}(t,R)$, and $\Delta^{\mathcal{P}}$ contains the transition $\langle s, pop(R), \mathtt{tmp}(t,R)\rangle$. Furthermore, for each $R' \in R[x \leftarrow I]$, $\Delta^{\mathcal{P}}$ contains the transition $\langle \mathtt{tmp}(t,R), push(R'), s'\rangle$. In other words, $\mathcal{P}$ first moves to $\mathtt{tmp}(t,R)$ to indicate that it is about to assign a new value to x. From $\mathtt{tmp}(t,R)$ there are several outgoing transitions each corresponding to the assignment of one particular value that belongs in I. Each of these transitions leads to the state s'.

$pop(a,I)$: We remove the top-most region Q in the stack in case its symbol is a and its age lies in the interval I. The main difficulty is to update the information in the next region P in the stack. Recall that when performing timed transitions, the items of P are not changed to reflect the passage of time. The update operation is performed in two steps. First we let time pass on P until it is transformed to a region that *supports* Q after which we compute the product R and use it to replace both Q and P. Formally, for each transition $t = \langle s, pop(a,I), s'\rangle \in \Delta^{\mathcal{T}}$, we add the following states and transitions. For regions P,Q with $Q \models (a \in I)$, $S^{\mathcal{P}}$ contains the states $\mathtt{tmp}(t,Q)$ and $\mathtt{tmp}(t,Q,P)$, and $\Delta^{\mathcal{P}}$ contains the transitions $\langle s, pop(Q), \mathtt{tmp}(t,Q)\rangle$ and $\langle \mathtt{tmp}(t,Q), pop(P), \mathtt{tmp}(t,Q,P)\rangle$. Furthermore, for each region $R \in P * Q$, $\Delta^{\mathcal{P}}$ contains the transition $\langle \mathtt{tmp}(t,Q,P), push(R), s'\rangle$. Observe that the plain symbol $\vdash$ is in the left-most position of the newly pushed region R.

$push(a,I)$: We create a new top-most region in the stack. In the new region, the values of the plain clocks are copied, and the shadow items are all reset making them equal to the values of their plain counter-parts. Finally, the new stack symbol a is assigned an age in the interval I. Formally, for each transition $t = \langle s, push(a,I), s'\rangle \in \Delta^{\mathcal{T}}$ and region R, the set $S^{\mathcal{P}}$ contains the states $\mathtt{tmp}_1(t,R)$ and $\mathtt{tmp}_2(t,R)$. The set $\Delta^{\mathcal{P}}$ contains the transitions $\langle s, pop(R), \mathtt{tmp}_1(t,R)\rangle$ and $\langle \mathtt{tmp}_1(t,R), push(R), \mathtt{tmp}_2(t,R)\rangle$. Furthermore, for each $R' \in Reset(R)[a \leftarrow I]$, $\Delta^{\mathcal{P}}$ contains the transition $\langle \mathtt{tmp}_2(t,R), push(R'), s'\rangle$. Observe that the plain symbol $\vdash$ is in the left-most position of the newly pushed region R'.

VII. Correctness

In this section we show correctness of the construction described in Section VI. Given a TPDA $\mathcal{T} = \left\langle S^{\mathcal{T}}, s^{\mathcal{T}}_{init}, \Gamma^{\mathcal{T}}, \Delta^{\mathcal{T}}\right\rangle$ consider the PDA $\mathcal{P} = \left\langle S^{\mathcal{P}}, s^{\mathcal{P}}_{init}, \Gamma^{\mathcal{P}}, \Delta^{\mathcal{P}}\right\rangle$ derived from $\mathcal{T}$ as described in Section VI. Consider a state $s_F \in S^{\mathcal{T}}$. Then:

Theorem 1. *s_F is reachable in $\mathcal{T}$ iff s_F is reachable in $\mathcal{P}$.*

Let A be a non-empty set of symbols. An *extended region* R_A over A is a word $r_1 r_2 \cdots r_n \in (2^{A \times Max})^+$ such that the following conditions are satisfied (below, let $r := \cup_{1 \le i \le n} r_i$): $|(A \times Max) \cap r| \le 1$ and $r_i \ne \emptyset$ for all $i : 2 \le i \le n$. Observe that a region is an extended region over the set Z. We extend all notations and operations on regions (when meaningful) to extended regions in the natural manner.

Below, we give the proof of Theorem 1 in both direction.

From $\mathcal{T}$ to $\mathcal{P}$: A *stack region*, or simply an *s-region*, is a word $\mathcal{R} = R_0 R_1 \cdots R_n$ over the set of regions. Notice that the stack content of $\mathcal{P}$ is always an s-region. For regions P,Q, we say that P *weakly supports* Q, denoted $P \lessapprox Q$, if there is a region $P' \in P^{++}$ such that $P' \preceq Q$. An s-region $\mathcal{R} = R_0 R_1 \cdots R_n$ is said to be *weakly coherent* if $R_i \lessapprox R_{i+1}$ for all $i : 0 \le i < n$; and is said to be *coherent* if $R_i \preceq R_{i+1}$ for all $i : 0 \le i < n$. A configuration $\langle s, \mathcal{R}\rangle$ in $\mathcal{P}$ is said to be (weakly) coherent if $\mathcal{R}$ is (weakly) coherent. We show the following property for reachable configurations in $\mathcal{P}$.

Lemma 2. *All reachable genuine configurations are weakly coherent.*

Consider a weakly coherent s-region $\mathcal{R} = R_0 R_1 \cdots R_n$, we say that $\mathcal{Q} = Q_0 Q_1 \cdots Q_n$ is a *strengthening* of $\mathcal{R}$ if $Q_n = R_n$, and $Q_i \in R_i^{++}$ and $Q_i \preceq Q_{i+1}$ for all $i : 0 \le i < n$. Notice that this operation is well-defined by the following lemma.

Lemma 3. *If $R_1 \lessapprox R_2$ and $R_3 \in R_2^{++}$ then $R_1 \lessapprox R_3$*

Consider a coherent s-region $\mathcal{R} = R_0 R_1 \cdots R_n$. Define $Z_{\mathcal{R}}$ to be the set of symbols $Z \times n^{(0)}$. A *collapsing* C of $\mathcal{R}$ is an extended region over $Z_{\mathcal{R}}$ such that the following conditions are satisfied:

- $C^{\top} = \bigcup_{i=0}^{n} (R_i^{\top} \times \{i\})$.
- $Val(R_i)(z) = Val(C)(z,i)$ for all $i : 0 \le i \le n$ and $z \in R_i^{\top}$.
- $Index(R_i)(z) = 1$ iff $Index(C)(z,i) = 1$ for all $i : 0 \le i \le n$ and $z \in R_i^{\top}$.
- $Index(R_i)(z_1) \le Index(R_i)(z_2)$ if and only if $Index(C)(z_1,i) \le Index(C)(z_2,i)$ for all $i : 0 \le i \le n$ and $z_1, z_2 \in R_i^{\top}$.
- $Val(C)(y^{\bullet}, i) = Val(C)(y, i-1)$ and $Index(C)(y^{\bullet}, i) = Index(C)(y, i-1)$ for all $y \in \left(R_{i-1}^{\top} \cap Y\right)$ and $i : 1 \le i \le n$.

Consider a coherent configuration $\beta = \langle s, \mathcal{R}\rangle$ in $\mathcal{P}$, a collapsing C of $\mathcal{R}$, and a configuration $\gamma = \langle s', \mathtt{X}, w\rangle$ in $\mathcal{T}$. Let $w = \langle a_1, v_1\rangle \cdots \langle a_n, v_n\rangle$ and let $\mathcal{R} = R_0 R_1 \cdots R_n$. We write $\gamma \models_C \beta$ if there is a valuation θ of C such that $\theta \models C$ and the following conditions are satisfied:

- $s' = s$.
- $\mathtt{X}(x) = \theta(x,n)$ for all $x \in X$.
- $a_i \in R_i^{\top}$ for all $i : 1 \le i \le n$.
- $v_i = \theta(a_i, i)$ for all $i : 1 \le i \le n$.

Lemma 4. *For any genuine reachable configuration β in $\mathcal{P}$, strengthening $\beta' = \langle s, \mathcal{R}\rangle$ of β, and collapsing C of $\mathcal{R}$, there is a configuration γ in $\mathcal{T}$ such that $\gamma \models_C \beta$ and $\gamma_{init} \overset{*}{\rightsquigarrow} \gamma$.*

Now, if the final state s_F is reachable in $\mathcal{P}$ then we know that there a genuine reachable configuration β whose state is precisely s_F. Notice that Lemma 2 implies that β is weakly coherent (and hence we can speak about the existence of at least one strengthening $\beta' = \langle s, \mathcal{R} \rangle$). Moreover, it is easy to show that for the coherent s-region $\mathcal{R}$, there exists at least one collapsing C of R. Thus, we can apply Lemma 4 to the genuine reachable configuration β in $\mathcal{P}$, the strengthening β', and the collapsing C, to show the existence of a reachable configuration γ in $\mathcal{T}$ whose state is s_F (since $\gamma \vDash_C \beta$).

From $\mathcal{P}$ to $\mathcal{T}$: For the opposite direction, we need to prove the following lemma:

Lemma 5. *For any reachable configuration γ in $\mathcal{T}$, there is a configuration β in $\mathcal{P}$, a strengthening $\beta' = \langle s, \mathcal{R} \rangle$ of β, and collapsing C of $\mathcal{R}$, such that $\gamma \vDash_C \beta$ and $\beta_{init} \xrightarrow{*} \beta$.*

As an immediate consequence of Lemma 5, we get that if a state s_F is reachable in $\mathcal{T}$, then s_F is reachable in $\mathcal{P}$.

VIII. Complexity of the Reachability Problem

In this section, we show:

Theorem 6. *The reachability problem for TPDA is* EXPTIME-*complete.*

The rest of this section is devoted to the proof of this theorem. Let us first prove the EXPTIME lower bound of Theorem 6.

Lemma 7. *The reachability problem for TPDA is* EXPTIME-*hard.*

Proof: It is known that the following problem is EXPTIME-complete [16]: Given a labeled pushdown automaton $\mathcal{P}$ recognizing a language L, and n finite state automata $\mathcal{A}_i$ recognizing languages L_i, check the non-emptiness of $L \cap \bigcap_{i=1}^{n} L_i$. We can show that this problem can be reduced, in polynomial time, to the reachability problem for a TPDA $\mathcal{T}$. The pushdown part of $\mathcal{T}$ simulates the labeled pushdown automaton $\mathcal{P}$, while each clock x_i is used to simulate the automaton $\mathcal{A}_i$. The valuation of the clock x_i gives the current state of $\mathcal{A}_i$. (We assume here that the automaton $\mathcal{A}_i$ does not contain epsilon-transitions.) Moreover, we use an auxiliary clock to ensure that no time elapses during the whole simulation.

The simulation proceeds as follows: An ϵ-transition of $\mathcal{P}$ is simulated by a transition of the pushdown part of $\mathcal{T}$ while the clocks remain unchanged. A labeled transition of $\mathcal{P}$ with an input symbol a, is simulated by a transition of the pushdown part of $\mathcal{T}$, followed by a sequence of transitions in which the clocks are checked and then updated, one after the other, to ensure each automaton $\mathcal{A}_i$ is able to perform a transition labeled by a. ■

The following lemma shows the EXPTIME upper bound of Theorem 6.

Lemma 8. *The reachability problem for TPDA is in* EXPTIME.

Proof: It is well-known that the reachability problem for (untimed) pushdown automata can be solved in polynomial time (see for instance [7]). On the other hand, in Sections VI and VII, we showed that it is possible to construct a PDA $\mathcal{P}$, whose size is exponential in the given TPDA $\mathcal{T}$, such that the reachability problem for $\mathcal{T}$ is reducible to its corresponding one for $\mathcal{P}$. This implies that the reachability problem for TPDA is in EXPTIME. ■

IX. Conclusions and Future Work

We have considered TPDA, an extension of two classical models, namely those of pushdown automata and timed automata. We have shown the decidability of the reachability problem for TPDA through a reduction to the corresponding problem for pushdown automata. The reduction relies on a non-trivial extension of the classical *region* encoding in a manner that allows to reason about unbounded sets of clocks. Interesting directions for future research include considering more general verification problems such as the model checking problem wrt. temporal logics such as LTL, decision problems over the game-based semantics, and the verification of priced TPDA models.

References

[1] P. A. Abdulla, M. F. Atig, and J. Stenman. The minimal cost reachability problem in priced timed pushdown systems. In *LATA*, 2012.

[2] P. A. Abdulla and A. Nylén. Timed Petri nets and BQOs. In *ICATPN*, 2001.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[4] M. Benerecetti, S. Minopoli, and A. Peron. Analysis of timed recursive state machines. In *TIME*, pages 61–68. IEEE Computer Society, 2010.

[5] B. Bérard, F. Cassez, S. Haddad, O. Roux, and D. Lime. Comparison of different semantics for time Petri nets. In *ATVA 2005*, 2005.

[6] A. Bouajjani, R. Echahed, and R. Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Hybrid Systems*, LNCS 999, pages 64–85. Springer, 1994.

[7] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, LNCS 1243, pages 135–150. Springer, 1997.

[8] P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In *FSTTCS*, LNCS 3328, pages 148–160. Springer, 2004.

[9] P. Bouyer and F. Laroussinie. Model checking timed automata. In Stephan Merz and Nicolas Navet, editors, *Modeling and Verification of Real-Time Systems*, pages 111–140. ISTE Ltd. – John Wiley & Sons, Ltd., January 2008.

[10] Z. Dang. Pushdown timed automata: a binary reachability characterization and safety verification. *Theor. Comput. Sci.*, 302(1-3):93–121, 2003.

[11] Z. Dang, T. Bultan, O. H. Ibarra, and R. A. Kemmerer. Past pushdown timed automata and safety verification. *Theor. Comput. Sci.*, 313(1):57–71, 2004.

[12] Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer, and J. Su. Binary reachability analysis of discrete pushdown timed automata. In *CAV*, LNCS 1855, pages 69–84. Springer, 2000.

[13] M. Emmi and R. Majumdar. Decision problems for the verification of real-time software. In *HSCC*, LNCS 3927, pages 200–211. Springer, 2006.

[14] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *CAV*, volume 1855 of *LNCS*. Springer, 2000.

[15] J. Esparza and S. Schwoon. A bdd-based model checker for recursive programs. In *CAV*, volume 2102 of *LNCS*, pages 324–336. Springer, 2001.

[16] A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In *FoSSaCS*, 2010.

[17] S. Schwoon. *Model-Checking Pushdown Systems*. PhD thesis, Technische Universität München, 2002.

[18] A. Trivedi and D. Wojtczak. Recursive timed automata. In *Proceedings of the 8th international conference on Automated technology for verification and analysis*, ATVA, pages 306–324, 2010.