Adding Time to Pushdown Automata

(Tutorial)

Parosh Aziz Abdulla
Department of Information Technology
Uppsala University
Sweden
parosh@it.uu.se

Mohamed Faouzi Atig
Department of Information Technology
Uppsala University
Sweden
mohamed_faouzi.atig@it.uu.se

Jari Stenman
Department of Information Technology
Uppsala University
Sweden
jari.stenman@it.uu.se

In this tutorial, we illustrate through examples how we can combine two classical models, namely those of *pushdown automata* (PDA) and *timed automata*, in order to obtain *timed pushdown automata* (TPDA) [2, 1]. Furthermore, we describe how the reachability problem for TPDAs can be reduced to the reachability problem for PDAs.

1 Introduction

In this tutorial, we describe a timed extension of the widely used model of Pushdown Automata (PDA) [2, 1]. A PDA computes by moving between states according to some given transition rules. Additionally, a PDA may utilize a stack to store information. This information is encoded in *stack symbols*, and the PDA may add a symbol (*push*) to or remove a symbol (*pop*) from the stack. The defining feature of a stack is that it has ordering on its elements, traditionally from *top* to *bottom*; the PDA can only access the topmost element.

An interesting question is what happens to this model when we extend it with quantitative properties. Will basic problems, such as state reachability, still be decidable? In particular, we are interested in extending the model with continuous time in a similar manner in which Timed Automata [5] extend Finite Automata. Thus, we consider Timed Pushdown automata TPDA. A TPDA is a PDA that is augmented with a finite number of *clocks*. It operates in the following manner:

- at any point in the computation, time may elapse by some real number, increasing the values of all clocks
- the values of clocks constrain the actions of the automaton

In addition to the set of clocks, we also store the age of each stack symbol. We can view this as an additional clock. Accordingly, the ages of stack symbols increase whenever time elapses. Furthermore, possible actions of the automaton may be restricted by the age of topmost stack symbol.

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 to or popped from the stack). Notice that a TPDA induces a system that is infinite

Submitted to:
QFM 2012

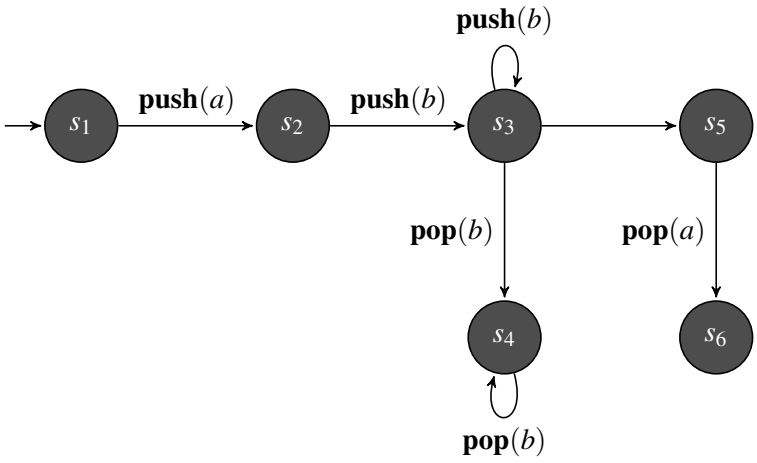


Figure 1: A simple PDA

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.

Outline In the next section, we present an overview of Pushdown Automata. In Section 3, we describe the timed extension of PDA and show some examples of computations. In Section 4, we recall and extend the notion of regions, and show how we can use them to define a symbolic encoding of TPDA configurations. Finally, in Section 5 we describe how to construct a PDA which simulates a given TPDA. The section ends with a detailed example of how the aforementioned TPDA computation is simulated.

2 PDA

In this section, we informally describe the model of Pushdown Automata. A Pushdown Automaton (PDA) is a tuple $(S, s_{init}, \Gamma, \Delta)$ consisting of a finite set of *states* S, an initial state s_{init}, a finite *stack alphabet* Γ, and a finite set of *transition rules* Δ. During the operation of a PDA, it may store information in a stack. It may add information, which is referred to as *pushing*, or it may remove information, which is called *popping*. The stack is a last-in, first-out queue, and access is restricted to the first element. The stack alphabet contains all possible symbols that may be stored in the stack, and the set of transition rules describe the manner in which the automaton is allowed to move between states. Each transition rule is of the form $(s, \mathbf{op}, t)$. The rule contains a source state s, a target state t and a stack operation **op**. The stack operation is either $\mathbf{push}(a)$, $\mathbf{pop}(a)$ or **nop** (here, a is an arbitrary symbol from the stack alphabet). A transition rule describes that the automaton may move from s to t while performing the stack operation **op**. The operation $\mathbf{push}(a)$ pushes a onto the stack, and $\mathbf{pop}(a)$ pops it. The operation **nop** is an "empty" operation which can be used to change state without modifying the stack. Figure 1 shows a PDA with the state set $\{s_1, s_2, s_3, s_4, s_5, s_6\}$ and stack alphabet $\{a, b\}$. The initial state of the automaton is s_1. The transition rules are drawn as arrows between states, labeled with the stack operation (missing labels mean **nop**).

At any point during a computation, the PDA is in a certain *configuration*, defined by the current state and the current stack content. Figure 2 shows the configurations that appear along a computation in

which the automaton starts from its initial configuration (the state is s_1 and the stack is empty), moves to s_2 while pushing a, then moves to s_3 while pushing b, and finally pops b and moves to s_4.

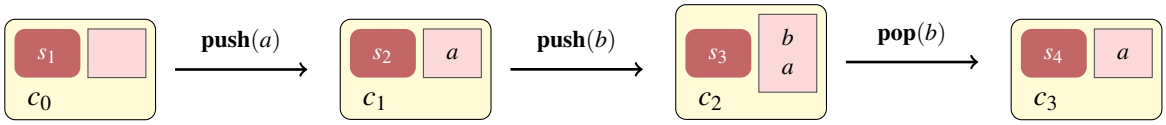


Figure 2: Computation of a PDA

Reachability Given a pushdown automaton, the reachability problem is the problem of deciding whether the automaton can reach a particular state s. In other words, we ask whether there is a computation of the automaton (starting from the initial configuration) that visits a configuration where the state is s, regardless of the content of the stack. It turns out that for the automaton in Figure 1, the state s_4 is reachable but the state s_6 is not. This is because in order to move from s_5 to s_6, the automaton has to pop a. However, the topmost symbol when the automaton is in state s_5 will always be b. For PDA, reachability is decidable in polynomial time [6].

3 Timed Pushdown Automata

The classical model of Timed Automata extends finite state automata with a finite set of real-valued *clocks*. We extend PDA in a similar way, in the sense that a Timed Pushdown Automaton (TPDA) consists of a finite set of states S, an initial state s_{init}, a finite stack alphabet Γ, a finite set of transition rules Δ, and a finite set of clocks X. The transition rules are also extended in the sense that they can read and write the values of clocks. More specifically, a transition rule $(s, \mathbf{op}, t)$ refers not only to stack operations. Instead, **op** can also be one of the *clock operations* $x \in I\,?$ and $x \leftarrow I$. The operation $x \in I\,?$ checks whether the value of the clock x is in the interval I. For example, the transition rule $(s, x \in [1:3]\,?, t)$ can only be performed when the value of x is between 1 and 3. The operation $x \leftarrow I$ *nondeterministically* resets the value of the clock x to some value in the interval I. Additionally, each stack symbol is equipped with a value representing its *age*. We modify the stack operations to use these values: $\mathbf{push}(a, I)$ pushes a and nondeterministically sets its initial age to some value in the interval I, while $\mathbf{pop}(a, I)$ may only pop the topmost stack symbol if it is equal to a and its age is in the given interval I.

As with PDA, the semantics of TPDA are given by a transition system over configurations. The configurations of a TPDA need to contain additional information, namely the values of all clocks and the ages of all stack symbols. The values of all clocks are given by a *clock valuation*; a mapping $X \mapsto \mathbb{R}^{\geq 0}$ (where $\mathbb{R}^{\geq 0}$ stands for the non-negative real numbers). To capture the ages of clocks symbols, we store tuples in the stack. Each tuple consists of (i) a stack symbol from the stack alphabet Γ and (ii) its corresponding age. Figure 4 and Figure 5 show an example computation of a TPDA (note that this computation is not related to the automaton in Figure 3). For example, in the configuration c_0 in Figure 4, the automaton is in the state s_1 with an empty stack, and the values of the two clocks x and y are 0. In the configuration c_3 in the same figure, the stack consists of a symbol a which has age 2.4.

There are two different types of transitions between configurations of a TPDA; *discrete* and *timed*. Discrete transitions are direct applications of the transition rules in Δ. Timed transitions simulate the passage of time. At any point in the computation, the automaton may take a timed transition, which means that all clock values and ages of stack symbols are increased by a positive real number. Figures 4

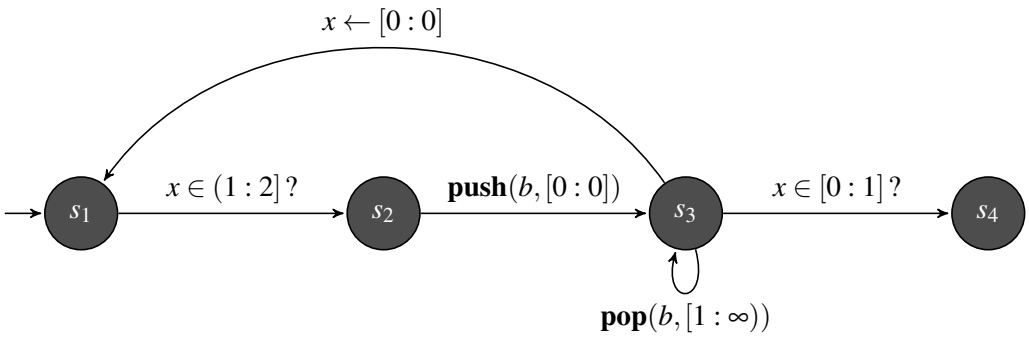


Figure 3: A simple TPDA

and 5 show a computation of a TPDA with clocks $X = \{x,y\}$ and stack alphabet $\Gamma = \{a,b,c,d\}$. We will describe the effect of each type of transition with an example from these figures.

Between c_2 and c_3, the TPDA moves from s_2 to s_3 and pushes the symbol a onto an empty stack, setting its initial age to 2.4, a value which is in the allowed interval $[1:3)$. Recall that the initial age is nondeterministically chosen from the given interval; in the push between c_6 and c_7 the same interval is given, but the chosen value happens to be 2.9 instead. The operation $x \leftarrow I$ chooses and assigns a value nondeterministically. From c_7, the automaton resets the value of x. Its value, which was previously 6.1, is set to some value in the interval $[2:3]$, in this case 2.1. Assume that Δ contains a transition rule $(s_1, y \in (1:\infty)\,?, s_5)$. In c_{21}, the TPDA tests if the value of y is strictly greater than 1. It is, so the transition rule is applied, and the state changes to s_5, as shown in configuration c_{22}. The above transitions are all examples of discrete transitions, i.e. transitions that are induced by transition rules in Δ. Figure 4 and Figure 5 also contain a number of timed transitions. For example, the transition between c_8 and c_9 represents the passage of 0.9 time units. In c_9, the values of x and y and the ages of a and b have all been increased by 0.9.

Reachability In a similar manner to the reachability problem for PDA, the reachability problem for TPDA is the problem of deciding whether a particular state is reachable from the initial configuration or not. In other words, we ask whether it is possible to reach a configuration c such that the state of c is the given target state.

Notice that in the definition of the reachability problem, we do not place any restrictions on the stack contents or on the values of the clocks. However, the reachability of a state in a TPDA may, in general, depend on the clock values and the ages of the stack symbols. For example, the state s_4 in Figure 3 is not reachable because of timing limitations.

Since the set of configurations in a TPDA is infinite, we can not solve the reachability problem by iteratively computing the successors of the initial configuration until a fixed point is reached. Furthermore, we cannot use the classical techniques that solve the reachability problem for PDA [6] since those constructions rely on the stack alphabet being finite. Therefore, we will now describe a symbolic representation of clock valuations and ages of stack symbols. We will use this representation to construct a *symbolic* PDA that simulates the behavior of the given TPDA.

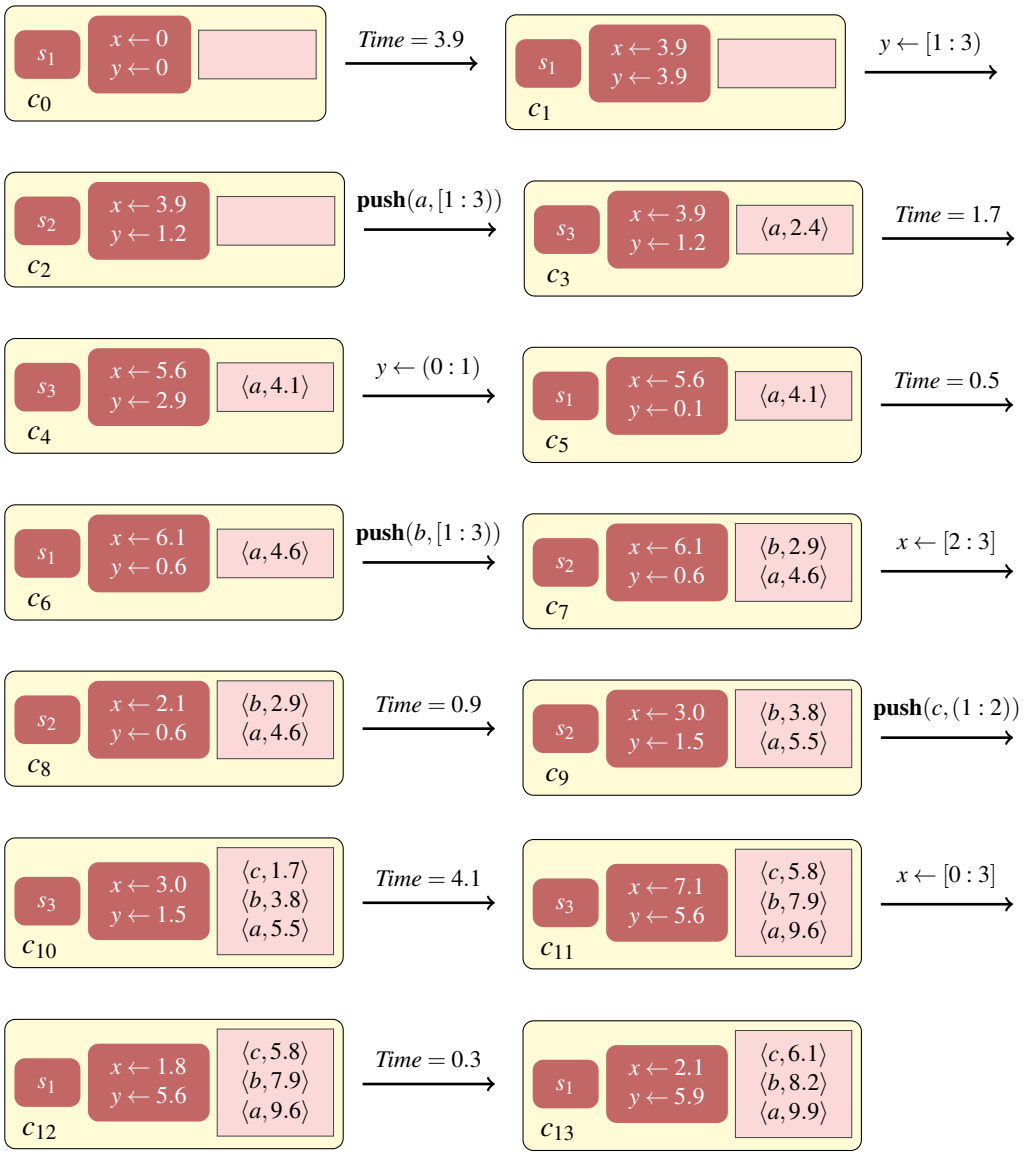


Figure 4: A computation of a TPDA

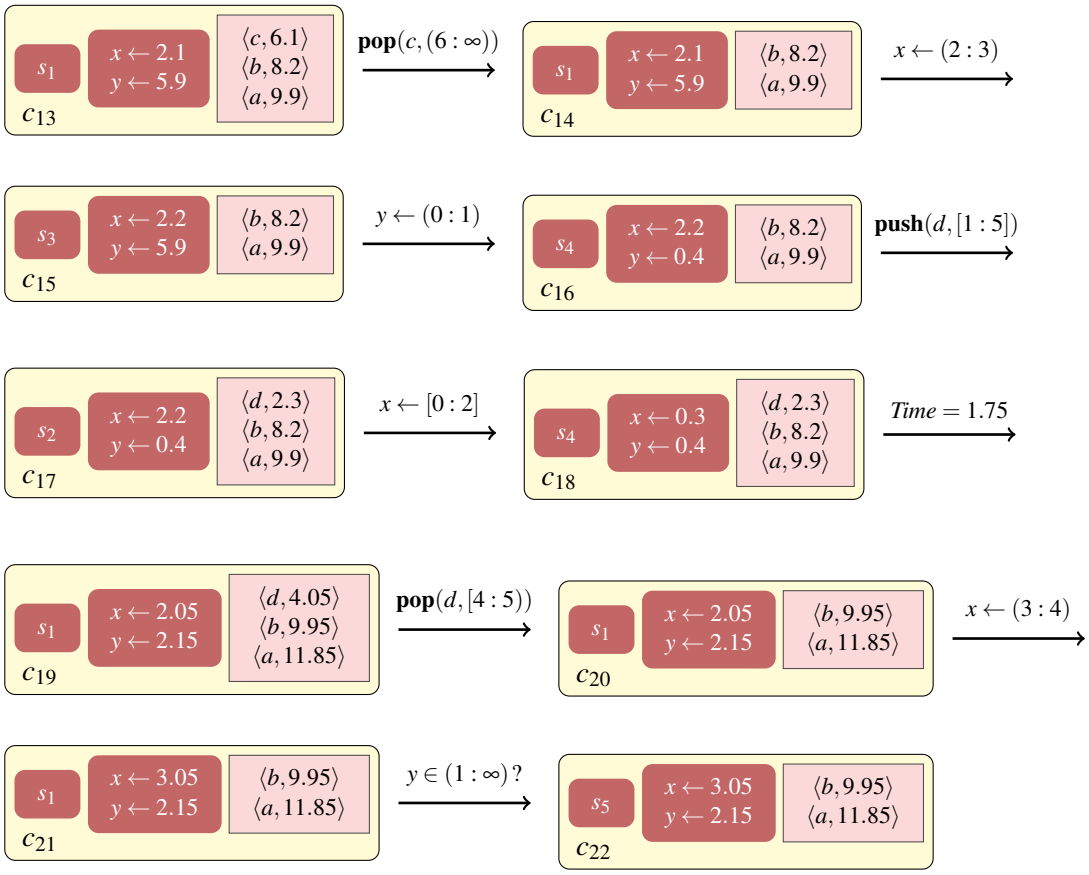


Figure 5: A computation of a TPDA (continued)

4 Regions

In this section, we describe a symbolic region encoding to represent the infinitely many clock valuations of a TPDA in a finite way. In the following section, we show how to construct, using this encoding, a symbolic PDA that simulates the behavior of a TPDA.

In the classical paper by Alur and Dill on timed automata [5], a *region* represents a set of clock valuations with "similar behaviors". The representation splits a real number into two parts: its *integral value*, i.e. its value rounded down to the nearest integer, and its *fractional part*, i.e. what is left when we subtract it by its integral value. For example, the integral value of π is 3, and its fractional part is $0.141592\ldots$. The main idea is that two configurations are equivalent if the following conditions hold:

- the integral values are identical in both valuations, up to a constant c_{max}
- the fractional part of any clock is either 0 in both valuations, or positive in both valuations
- the orderings of the fractional parts of all clocks are identical in both valuations

If the integral values are the same, the valuations will satisfy the same set of constraints. If the two valuations agree on the ordering of the fractional parts, they agree on the order in which the clocks will change integral values (and therefore in which order the constrained transitions will be enabled or disabled). The constant c_{max} is the largest constant appearing syntactically in the automaton. All values that are above c_{max} are indistinguishable form each other, so we can represent them symbolically with ω. In our example computation (Figure 4 and Figure 5), this constant is 7.

We will use a representation of regions inspired by [3, 4], that suites our purposes. In our representation, regions are sequences of sets. Each set contains one or more clocks together with their integral values. Their positioning in the sequence encodes the ordering of the fractional parts. If two clocks are in the same set, their fractional parts are equal. The first set contains all clocks with fractional part 0, and, for technical reasons, is the only set which may be empty. For example, the region R_1 in Figure 6 represents clock valuations in which the values of x_1 and x_2 are exactly 0 and 2, respectively. Furthermore, the integral value of x_3 is 1 and the integral value of x_4 is 2, and so on. Finally, the clocks are ordered in the sequence by increasing fractional part. Thus, the fractional parts of all clocks except x_1 and x_2 are strictly positive, and the fractional parts of x_6 and x_7 are the largest in the sequence (they are in the same set, so their fractional parts are equal).

Region rotations Given a region, we may simulate passage of time by *rotating* it. When time passes, one of two things may happen:

- Some items have fractional part 0, in which case any passage of time is enough to "push" them out
- No items have fractional part 0, in which case the items with the largest fractional part reach their next integral values.

For instance, consider the region R_2 in Figure 6. The next change in the region representation is that the values of x_6 and x_7 reach 4 and 1, respectively.

5 Translation

Our goal is is to reduce the reachability problem for TPDA to the reachability problem for PDA by translating the given TPDA to a PDA which simulates it. We will first describe a naive approach for

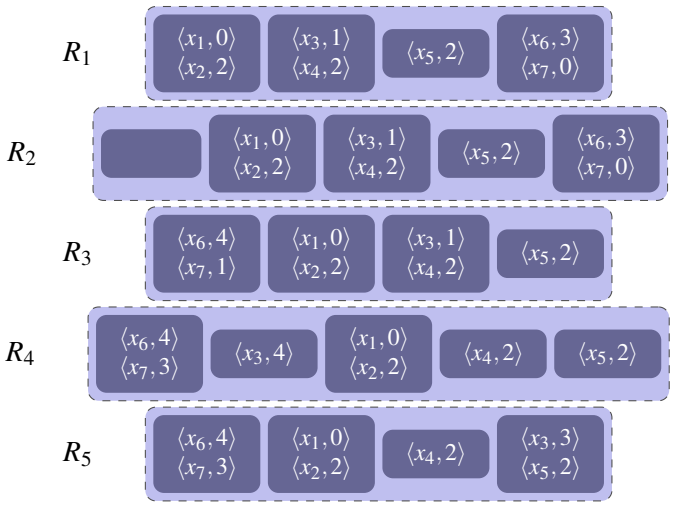


Figure 6: Example regions

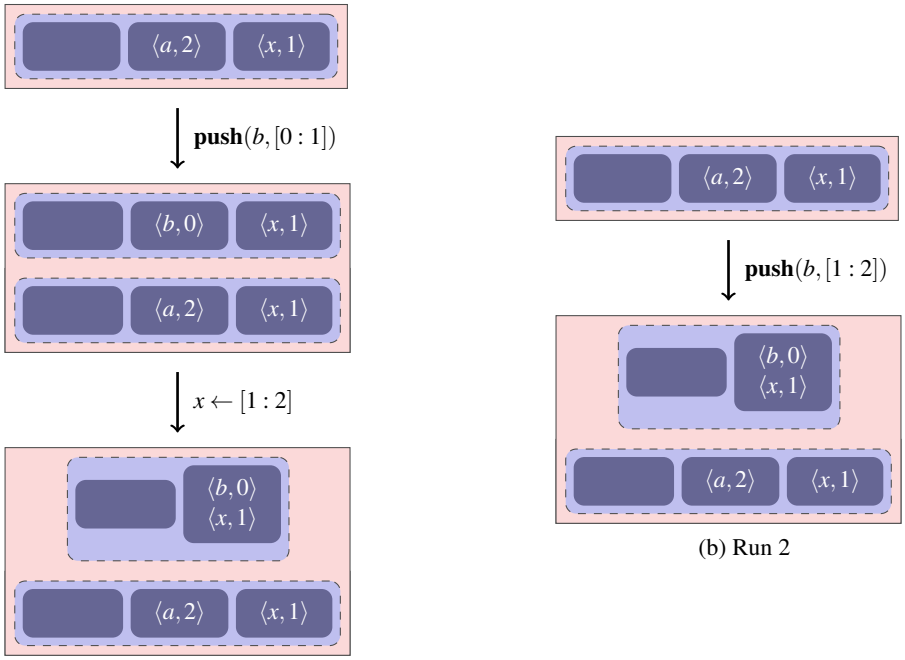
constructing such a PDA. Then we show the problem with this approach and explain how to amend it. At the end of this section, we show in detail how the computation in Figure 4 and Figure 5 is simulated by the PDA.

In the original paper on timed automata [5], the timed automaton is simulated by a region automaton, i.e. a finite state automaton that encodes the regions in its states. This abstraction relies on the fact that the set of clocks is 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 entirely on this abstraction.

Instead, we store the regions in the stack. Each symbol in the stack of the TPDA is represented in the stack of the PDA by a region that relates the stack symbol with all clocks. For example, consider Run 1 shown in Figure 7. At the beginning, the stack contains a region in which the integral values of a and x are 2 and 1, respectively, and the fractional part of x is larger than the fractional part of a, which is in turn larger than 0. The PDA then simulates the pushing of b with an initial age in $[0:1]$. This creates a new region on top of the stack which relates b to x. The region shown in the run is one of 4 possible regions. Next, the value of x is set to some value in $[1:2]$. In our case, it happens that x gets the same fractional part as b.

Unfortunately, it is not enough to relate each stack symbol to all clocks. Consider the final stack of Run 1 in Figure 7. What is the resulting stack if we now pop b? It is clear that the resulting stack must contain a and x. As for constraints on their values, we know from the topmost region that the fractional part of x is positive. We also know, from the region below, that the fractional part of a is positive. If we combine this information, we end up with one of the stacks in Figure 8.

To see the problem, consider Run 2 in 7. This run ends up with the same stack. However, the fractional part of x in this run can not be equal to the fractional part of a, since the value of x has not been reset. This rules out the stack in the middle in Figure 8. Therefore, we need to relate the fractional parts of a and b. A tempting solution is to simply record the value of a in the region representing b. However, since a PDA needs to have a finite stack alphabet, we can only record the values of finitely many previous stack symbols. At the same time, it is easy to construct counter-examples (similar to the one above) in which we need to keep the relationship between stack symbols that lie arbitrarily far apart in the stack. In [1], we show that we can in fact enrich the regions in a finite way in order to construct a PDA which simulates a TPDA. We will now explain the main points of this construction.



(a) Run 1

(b) Run 2

Figure 7: Example of information loss

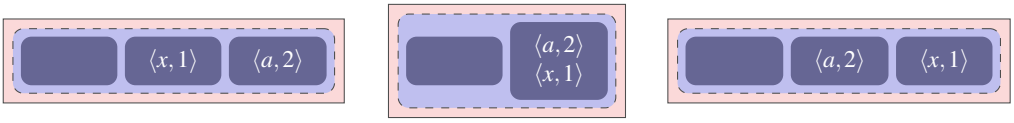


Figure 8: Result of popping

First, let us define the notion of items. An *item* is either a *plain item* or a *shadow item*. A plain item represents the value of a clock or the age of a stack symbol. We add a special reference clock $\vdash$, which is always 0 except when simulating a pop transition. In other words, this reference clock is not changed when we simulate timed transitions. Thus, the set of plain items consists of $X \cup \Gamma \cup \{\vdash\}$. On the other hand, shadow items record the values of the corresponding plain items in the region below. For each clock x and stack symbol a, the set of shadow items contains the symbols $x^\bullet$ and $a^\bullet$. Additionally, this set includes a shadow copy $\vdash^\bullet$ of the reference clock. The shadow items are used to remember the amount of time that elapses while the plain items they represent are not on the top of the stack. A region is then represented by a sequence of sets of items.

To illustrate this, let us simulate a push transition. Assume that the region R_1 in Figure 9 is the topmost region in the stack. The region 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 initial 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.

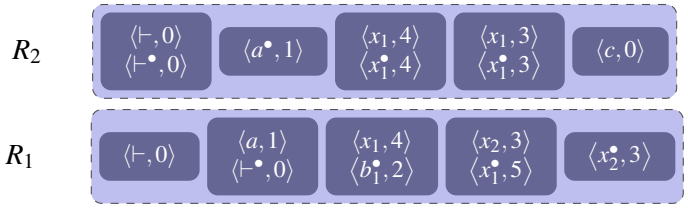


Figure 9: Example regions with shadow items

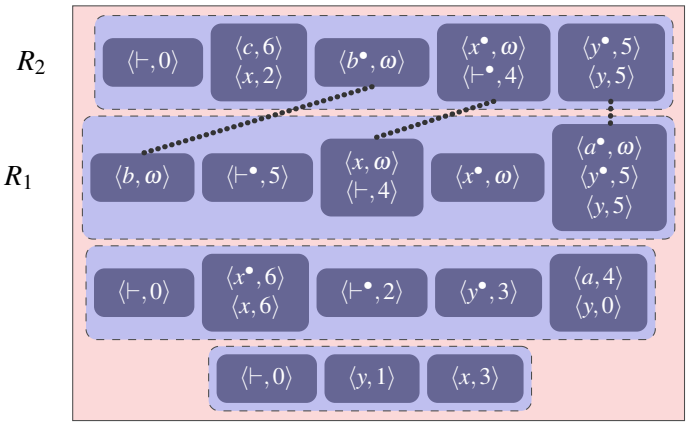


Figure 10: Simulating pop

Simulation We will now describe how to simulate the rest of the transitions, i.e. timed transitions, $x \in I?$, $x \leftarrow I$, and $\mathbf{pop}(a, I)$.

Timed transitions are simulated by rotating the top-most region, as described in the previous section. Note that the reference clock $\vdash$ is not affected by these rotations. For example, the rotation of the topmost region between S_{18} and S_{19} simulates the timed transition between c_{18} and c_{19} in Figure 4. The reference clock $\vdash$ stays in the first set, but all other items are rotated in a way which is consistent with the passage of 1.75 time units.

The operation $x \in I?$ checks whether the value of x is in the interval I or not. For every transition rule $(s, x \in I?, t)$ in the TPDA and every region that satisfies the condition $x \in I$, we create a sequence of two transition rules which first pops the region in question and then pushes it back.

The reset operation $x \leftarrow I$ sets the value of clock x to some value in the interval I. We simulate this by first popping the topmost region and then nondeterministically pushing a region which is identical except for the fact that x has been updated so that $x \in I$. Note that there may be several regions satisfying this; the region we push is chosen nondeterministically from these.

The interesting operation is pop: the operation merges the information in two different regions. The simulation is performed in two steps. First, the next top-most region is "refreshed", by repeatedly rotating it until its items are updated in a manner that reflects their current values. This is illustrated in Figure 10: the region R_1 is rotated until the shadow items in R_2 match their plain counterparts in R_1. In the figure, this matching is illustrated by dotted lines. Next, we combine the regions in the following way:

- The plain *stack symbol* is selected from the lower region (R_1)
- The plain *clock symbols* are selected from the upper region (R_2); it contains their most recent values

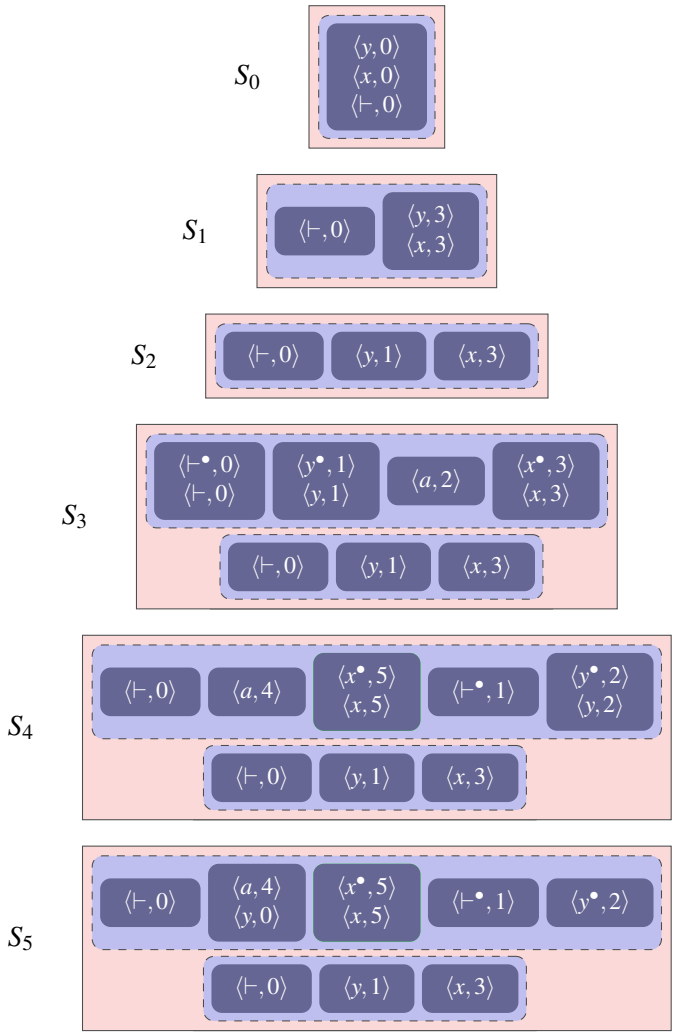
- Shadow items are selected from the lower region (R_1)

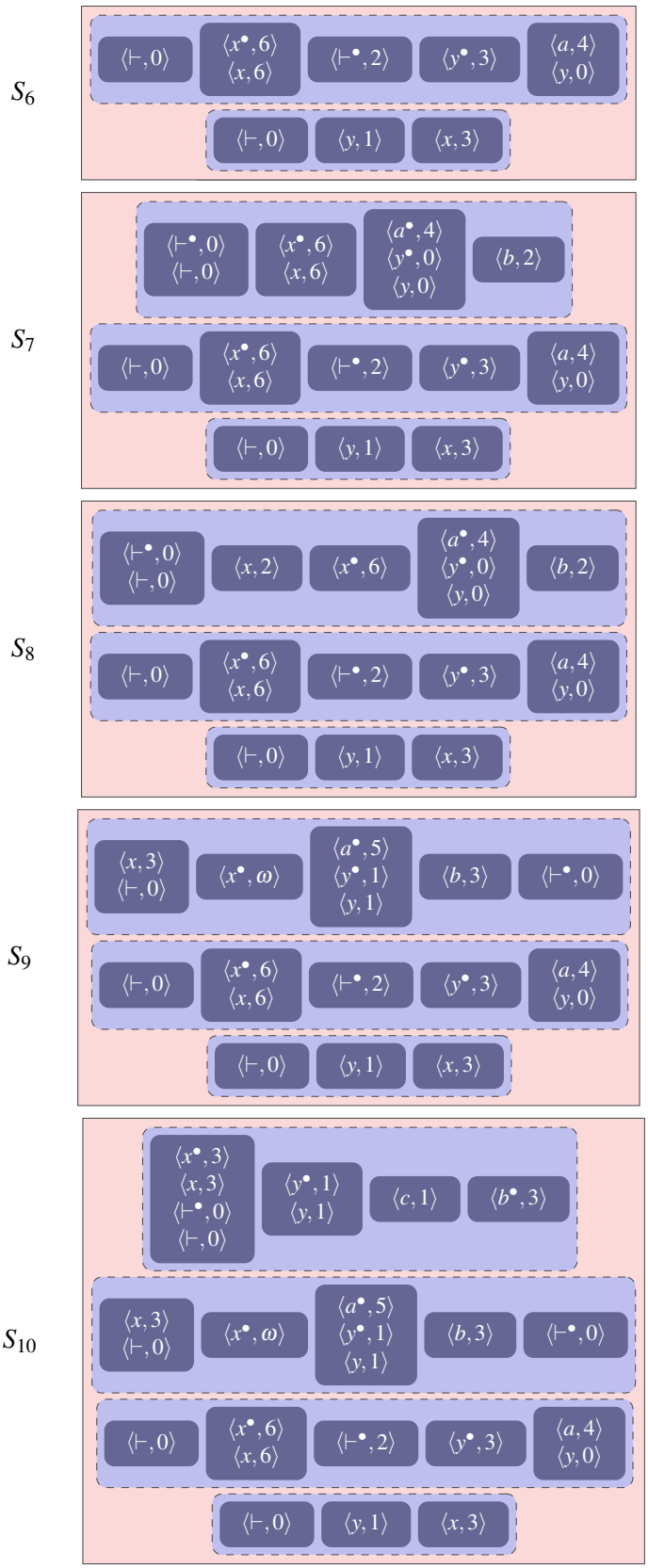
For example, the result of combining R_1 and R_2 is the topmost region in S_{14}. In this way, we simulate the passage of time only on the topmost region, but the effect "ripples" down the stack when popping. Thus, we only encode a finite amount of additional information in the regions, so the stack alphabet is kept finite.

Results Given a TPDA, we can solve the reachability problem by constructing a PDA which simulates it, as described in this section. The target state is reachable in the TPDA if and only if it is reachable in the PDA. However, the size of the PDA might be exponential in the size of the TPDA. The following theorem states the main result in [1]:

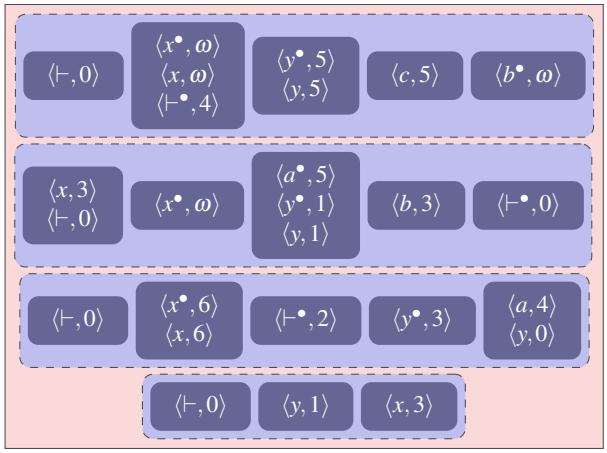
Theorem 1 *The reachability problem for* TPDA *is* EXPTIME*-complete.*

Figure 11: Simulation of a TPDA computation

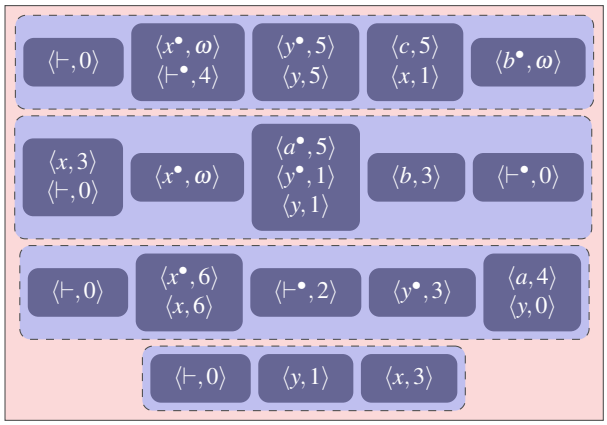



S_6
$\langle \vdash, 0\rangle$ $\langle x^\bullet, 6\rangle$ $\langle x, 6\rangle$ $\langle \vdash^\bullet, 2\rangle$ $\langle y^\bullet, 3\rangle$ $\langle a, 4\rangle$ $\langle y, 0\rangle$
$\langle \vdash, 0\rangle$ $\langle y, 1\rangle$ $\langle x, 3\rangle$
S_7
$\langle \vdash^\bullet, 0\rangle$ $\langle \vdash, 0\rangle$ $\langle x^\bullet, 6\rangle$ $\langle x, 6\rangle$ $\langle a^\bullet, 4\rangle$ $\langle y^\bullet, 0\rangle$ $\langle y, 0\rangle$ $\langle b, 2\rangle$
$\langle \vdash, 0\rangle$ $\langle x^\bullet, 6\rangle$ $\langle x, 6\rangle$ $\langle \vdash^\bullet, 2\rangle$ $\langle y^\bullet, 3\rangle$ $\langle a, 4\rangle$ $\langle y, 0\rangle$
$\langle \vdash, 0\rangle$ $\langle y, 1\rangle$ $\langle x, 3\rangle$
S_8
$\langle \vdash^\bullet, 0\rangle$ $\langle \vdash, 0\rangle$ $\langle x, 2\rangle$ $\langle x^\bullet, 6\rangle$ $\langle a^\bullet, 4\rangle$ $\langle y^\bullet, 0\rangle$ $\langle y, 0\rangle$ $\langle b, 2\rangle$
$\langle \vdash, 0\rangle$ $\langle x^\bullet, 6\rangle$ $\langle x, 6\rangle$ $\langle \vdash^\bullet, 2\rangle$ $\langle y^\bullet, 3\rangle$ $\langle a, 4\rangle$ $\langle y, 0\rangle$
$\langle \vdash, 0\rangle$ $\langle y, 1\rangle$ $\langle x, 3\rangle$
S_9
$\langle x, 3\rangle$ $\langle \vdash, 0\rangle$ $\langle x^\bullet, \omega\rangle$ $\langle a^\bullet, 5\rangle$ $\langle y^\bullet, 1\rangle$ $\langle y, 1\rangle$ $\langle b, 3\rangle$ $\langle \vdash^\bullet, 0\rangle$
$\langle \vdash, 0\rangle$ $\langle x^\bullet, 6\rangle$ $\langle x, 6\rangle$ $\langle \vdash^\bullet, 2\rangle$ $\langle y^\bullet, 3\rangle$ $\langle a, 4\rangle$ $\langle y, 0\rangle$
$\langle \vdash, 0\rangle$ $\langle y, 1\rangle$ $\langle x, 3\rangle$
S_{10}
$\langle x^\bullet, 3\rangle$ $\langle x, 3\rangle$ $\langle \vdash^\bullet, 0\rangle$ $\langle \vdash, 0\rangle$ $\langle y^\bullet, 1\rangle$ $\langle y, 1\rangle$ $\langle c, 1\rangle$ $\langle b^\bullet, 3\rangle$
$\langle x, 3\rangle$ $\langle \vdash, 0\rangle$ $\langle x^\bullet, \omega\rangle$ $\langle a^\bullet, 5\rangle$ $\langle y^\bullet, 1\rangle$ $\langle y, 1\rangle$ $\langle b, 3\rangle$ $\langle \vdash^\bullet, 0\rangle$
$\langle \vdash, 0\rangle$ $\langle x^\bullet, 6\rangle$ $\langle x, 6\rangle$ $\langle \vdash^\bullet, 2\rangle$ $\langle y^\bullet, 3\rangle$ $\langle a, 4\rangle$ $\langle y, 0\rangle$
$\langle \vdash, 0\rangle$ $\langle y, 1\rangle$ $\langle x, 3\rangle$

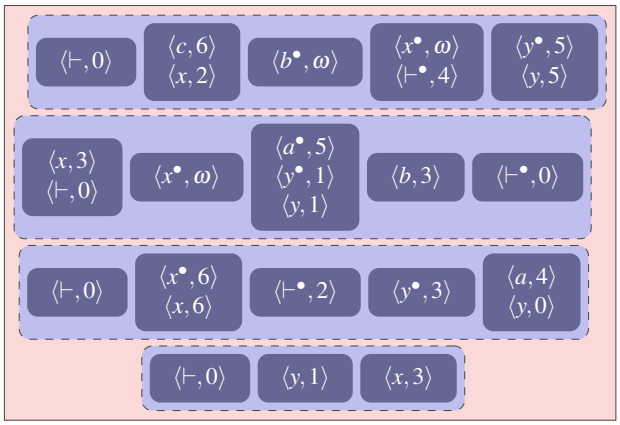
S_{11}



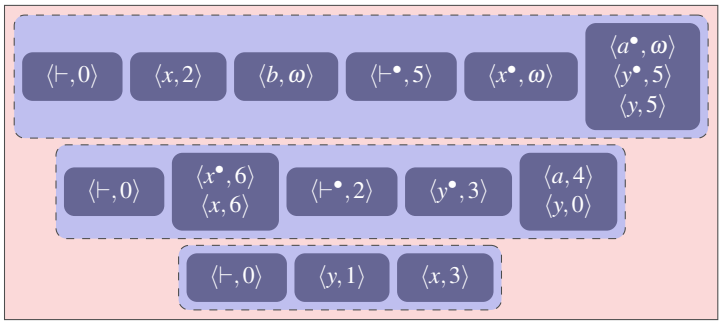
S_{12}



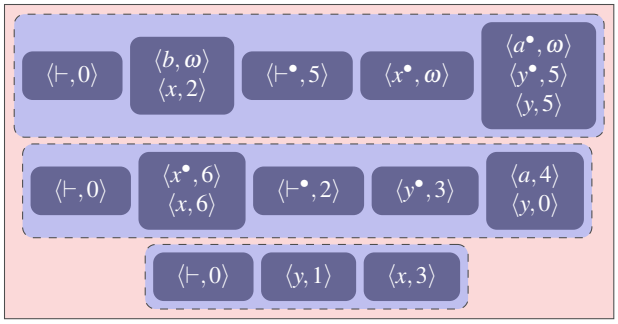
S_{13}



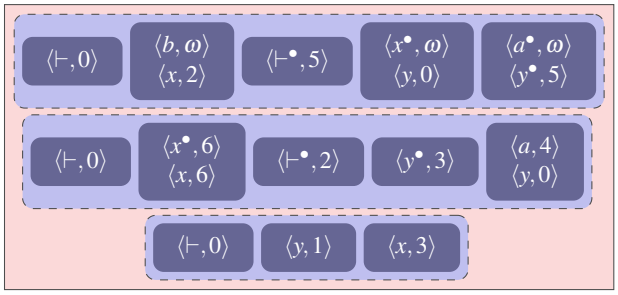
S_{14}



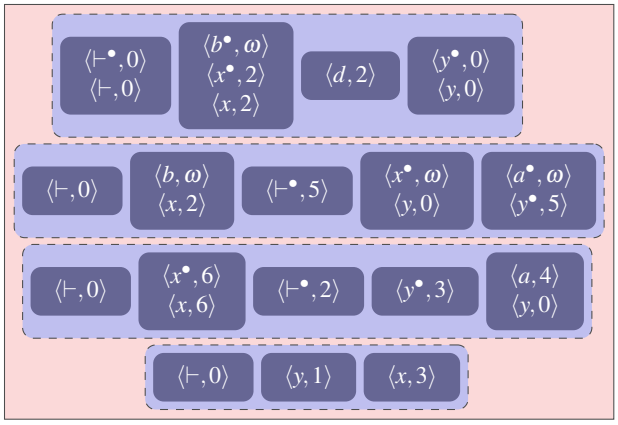
S_{15}



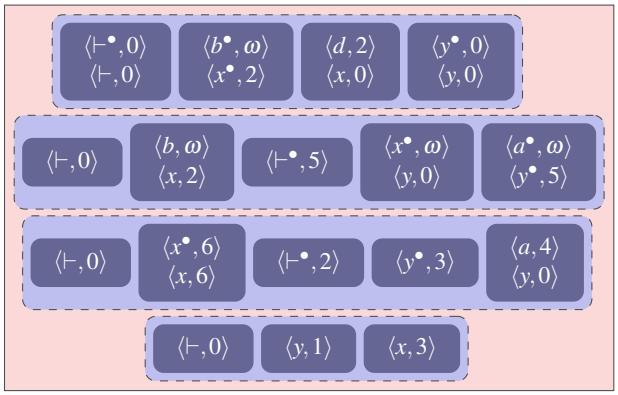
S_{16}

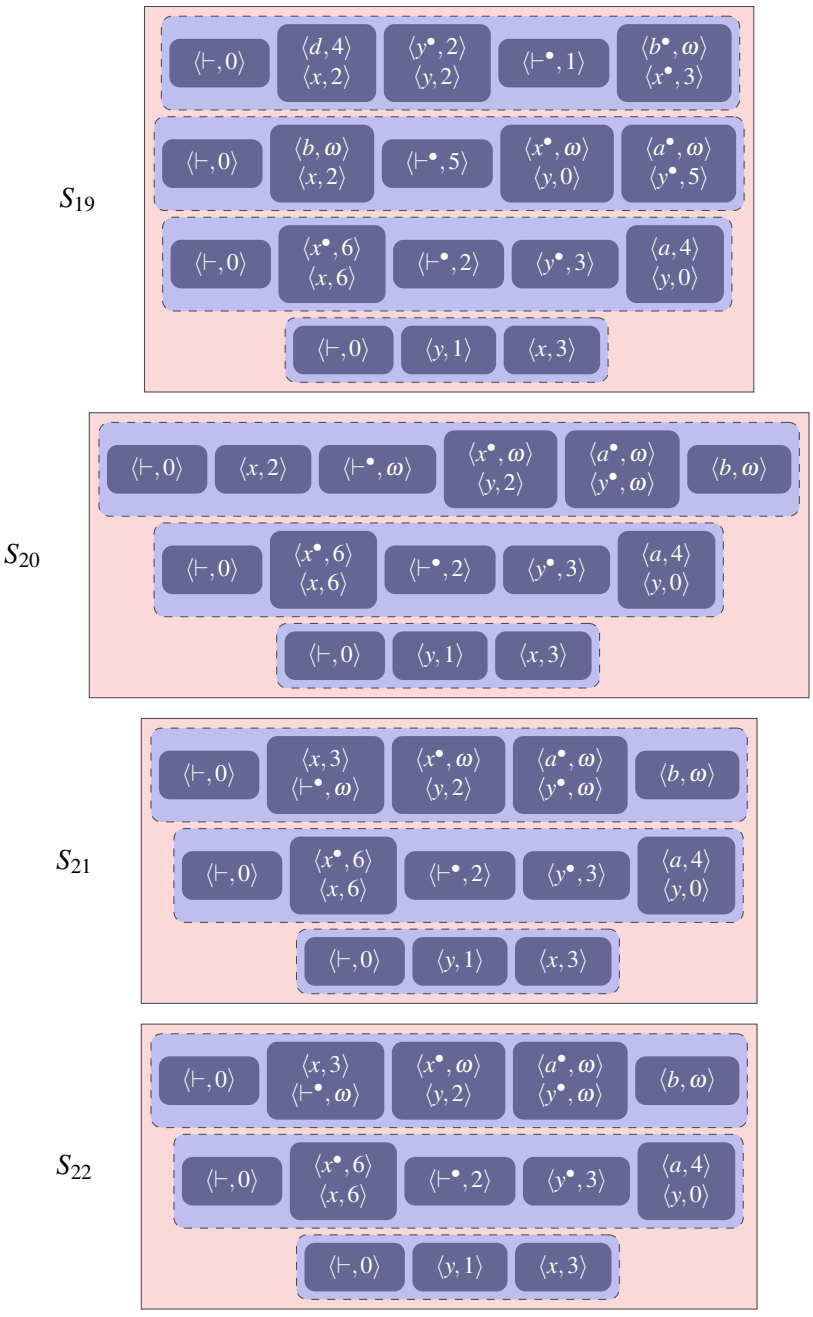


S_{17}



S_{18}





References

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

[2] P.A. Abdulla, M.F. Atig, and J. Stenman. The minimal cost reachability problem in priced timed pushdown systems. *Language and Automata Theory and Applications*, pages 58–69, 2012.

[3] P.A. Abdulla and B. Jonsson. Verifying networks of timed processes. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 298–312, 1998.

[4] P.A. Abdulla and B. Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–264, 2003.

[5] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[6] 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.