Analyzing Timed Systems Using Tree Automata

Sunil Akshay, Paul Gastin, Shankara Narayanan Krishna

To cite this version:

Sunil Akshay, Paul Gastin, Shankara Narayanan Krishna. Analyzing Timed Systems Using Tree Automata. 27th International Conference on Concurrency Theory (CONCUR 2016), 2016, Québec City, Canada. pp.27:1-27:14, 10.4230/LIPIcs.CONCUR.2016.27 . hal-01407942

HAL Id: hal-01407942

https://hal.archives-ouvertes.fr/hal-01407942

Submitted on 2 Dec 2016

Analyzing Timed Systems Using Tree Automata*

S. Akshay[1], Paul Gastin[2], and Shankara Narayanan Krishna[1]

1 **Dept. of CSE, IIT Bombay, Powai, Mumbai 400076, India**
akshayss@cse.iitb.ac.in
2 **LSV, ENS-Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France**
paul.gastin@lsv.ens-cachan.fr
1 **Dept. of CSE, IIT Bombay, Powai, Mumbai 400076, India**
krishnas@cse.iitb.ac.in

Abstract

Timed systems, such as timed automata, are usually analyzed using their operational semantics on timed words. The classical region abstraction for timed automata reduces them to (untimed) finite state automata with the same time-abstract properties, such as state reachability. We propose a new technique to analyze such timed systems using finite tree automata instead of finite word automata. The main idea is to consider timed behaviors as graphs with matching edges capturing timing constraints. Such graphs can be interpreted in trees opening the way to tree automata based techniques which are more powerful than analysis based on word automata. The technique is quite general and applies to many timed systems. In this paper, as an example, we develop the technique on timed pushdown systems, which have recently received considerable attention. Further, we also demonstrate how we can use it on timed automata and timed multi-stack pushdown systems (with boundedness restrictions).

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Timed automata, tree automata, pushdown systems, tree-width

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.27

1 Introduction

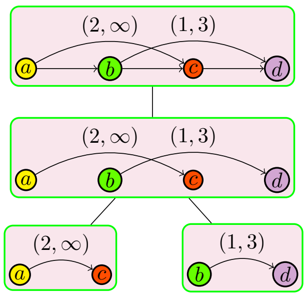
The advent of timed automata [4] marked the beginning of an era in the verification of real-time systems. Today, timed automata form one of the well accepted real-time modelling formalisms, using real-valued variables called clocks to capture time constraints. The decidability of the emptiness problem for timed automata is achieved using the notion of region abstraction. This gives a sound and finite abstraction of an infinite state system, and has paved the way for state-of-the-art tools like UPPAAL, which have successfully been used in the verification of several complex timed systems. In recent times [1, 6, 13] there has been a lot of interest in the theory of verification of more complex timed systems enriched with features such as concurrency, communication between components and recursion with single or multiple threads. In most of these approaches, decidability has been obtained by cleverly extending the fundamental idea of region or zone abstractions.

In this paper, we give a technique for analyzing timed systems, inspired from a different approach based on graphs and tree automata. This approach has been exploited for analyzing various types of *untimed systems*, e.g., [17, 10]. The basic template of this approach has three steps: (1) capture the behaviors of the system as graphs, (2) show that the class of

* This work was partly supported by LIA InForMeL, Indo-French CEFIPRA project AVERTS and DST-INSPIRE faculty award [IFA12-MA-17].

graphs that are actual behaviors of the system is MSO-definable, and (3) show that this class of graphs has bounded tree-width (or clique-width or split-width), or restrict the analysis to such bounded behaviors. Then, non-emptiness of the given system boils down to the satisfiability of an MSO sentence on graphs of bounded tree-width, which is decidable by Courcelle's theorem. Since, graphs of bounded tree-width can be interpreted in binary trees, the problem reduces to non-emptiness of a tree automaton whose existence follows from Courcelle's theorem. But, by providing a direct construction of the tree automaton, it is possible to obtain a good complexity for the decision procedure.

Our technique starts similarly, by replacing timed word behaviors of timed systems with graphs consisting of untimed words with additional time-constraint edges, called words with timing constraints (TCWs). However, the main complication here is that a TCW describes a run of the timed system, where the constraints are recorded but not checked. The TCW corresponds to an actual concrete run iff it is *realizable*. So, we are interested in the class of graphs which are *realizable* TCWs. The structural property that a graph must be a TCW is MSO-definable. However, it is unclear whether realizability is MSO definable over words with timing constraints. Given this, we cannot directly apply the approach of [17, 10]. Instead, we work on decomposition trees and construct a finite tree automaton checking realizability, which is the most involved part of the paper. More precisely, we show that words with timing constraints (TCWs) which are behaviors of certain classes of timed systems (like timed pushdown systems) are graphs of bounded split/tree-width. Hence, these graphs admit binary tree decompositions as depicted in the adjoining figure. Each node of the tree depicts an incomplete behavior/graph of the system, and by combining these behaviors as we go up the tree, we obtain a full or complete behavior (run) of the system. We construct a tree automaton that checks if the generated graph encoded as a tree satisfies the ***ValCoRe*** property (1) *Validity*: The root node depicts a syntactically correct labeled graph (TCW); (2) *Correctness of run*: The graph is indeed a correct run of the underlying timed system and; (3) *Realizability*: The root node depicts a realizable graph, i.e., we can find timestamps that realize all timing-constraints. To check realizability, the tree automaton needs to maintain a *finite* abstraction for each subtree encoding a TCW. Thanks to the bound on split/tree-width, our abstraction keeps a bounded number of positions, called end-points, in the (arbitrarily large) TCW. It subsumes (arbitrarily long) paths of timing constraints in the TCW by new timing constraints between these end-points. The constants in these new constraints are sums of original constants and may grow unboundedly. Hence, a key difficulty is to introduce suitable abstractions which aid in bounding the constants, while at the same time preserving realizability. Using tree decompositions of graph behaviors of bounded split/tree-width and tree automata proved to be a very successful technique for the analysis of *untimed* infinite state systems [17, 11, 10, 2]. This paper opens up this powerful technique for analysis of *timed* systems.



To illustrate the technique, we have reproved the decidability of non-emptiness of timed automata and timed pushdown automata (TPDA), by showing that both these models have a split-width ($|X|+3$ and $4|X|+6$) that is linear in the number of clocks X of the underlying system. This bound directly tells us the amount of information that we need to maintain in the construction of the tree automata. For TPDA we obtain an ExpTime algorithm, matching the known lower-bound for the emptiness problem of TPDA. For timed automata, since the split-trees are word-like (at each binary node, one subtree is small) we may use word automata instead of tree automata, reducing the complexity from ExpTime to PSpace, again matching the lower-bound. Interestingly, if one considers TPDA with no explicit clocks, but the stack is timed, then the split-width is a constant, 2. In this case, we have a polynomial time procedure to decide emptiness, assuming a unary encoding of constants in the system. To further demonstrate the power of our technique, we derive a new decidability result for non-emptiness of timed multi-stack pushdown automata under bounded rounds, by showing that the split-width of this model is again linear in the number of clocks, stacks and rounds. Exploring decidable subclasses of untimed multi-stack pushdown systems is an active research area [5, 12, 14, 16, 15], and our technique can extend these to handle time.

It should be noticed that the tree automata for validity and realizability (the most involved construction of this paper) are independent of the timed system under study. Hence, to apply the technique to other systems, one only needs to prove the bound on split-width and to show that their runs can be captured by tree automata. This is a major difference compared to many existing techniques for timed systems which are highly system dependent. Finally, we mention an orthogonal approach to deal with timed systems given in [6], where the authors show the decidability of the non-emptiness problem for a class of timed pushdown automata by reasoning about sets with timed-atoms. Detailed proofs and illustrative examples, omitted due to lack of space, can be found in [3].

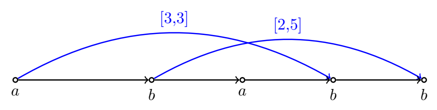
2 Graphs for behaviors of timed systems

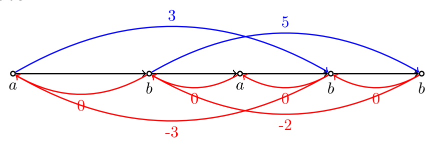
We fix an alphabet Σ and use Σ_ε to denote $\Sigma \cup \{\varepsilon\}$ where ε is the silent action. We also fix a finite set of closed intervals $\mathcal{I}$ which contains the special interval $[0,0]$. For a set S, we use $\leq \subseteq S \times S$ to denote a partial or total order on S. For any $x, y \in S$, we write $x < y$ if $x \leq y$ and $x \neq y$, and $x \lessdot y$ if $x < y$ and there does not exist $z \in S$ such that $x < z < y$.

2.1 Abstractions of timed behaviors

▶ **Definition 1.** A *word with timing constraints* (TCW) over $\Sigma, \mathcal{I}$ is a structure $\mathcal{V} = (P, \to, \lambda, \rhd, \theta)$ where P is a finite set of positions or points, $\lambda \colon P \to \Sigma_\varepsilon$ labels each position, the reflexive transitive closure $\leq\, = \to^*$ is a total order on P and $\to\, = \lessdot$ is the successor relation, $\rhd \subseteq \,<\, = \to^+$ gives the pairs of positions carrying a timing constraint, whose interval is given by $\theta \colon \rhd \to \mathcal{I}$.

For any position $i \in P$, the *indegree* (resp. *outdegree*) of i is the number of positions j such that $(j, i) \in \rhd$ (resp. $(i, j) \in \rhd$). A TCW is *simple* (denoted STCW) if each position has at most one timing constraint (incoming or outgoing) attached to it, i.e., for all $i \in P$, *indegree(i)* + *outdegree(i)* ≤ 1. A TCW is depicted below (left) with positions $1, 2, \ldots, 5$ labelled over $\{a, b\}$. *indegree*(4)=1, *outdegree*(1)=1 and *indegree*(3)=0. The curved edges decorated with intervals connect the positions related by $\rhd$, while straight edges are the successor relation $\to$. Note that this TCW is *simple*.





An ε-*timed word* is a sequence $w = (a_1, t_1) \ldots (a_n, t_n)$ with $a_1 \ldots a_n \in \Sigma_\varepsilon^+$ and $(t_i)_{1 \leq i \leq n}$ a non-decreasing sequence of real time values. If $a_i \neq \varepsilon$ for all $1 \leq i \leq n$, then w is a *timed word*. The projection on Σ of an ε-timed word is the timed word obtained by removing ε-labelled positions. Consider a TCW $W = (P, \to, \lambda, \rhd, \theta)$ with $P = \{1, \ldots, n\}$.

A timed word w is a *realization* of W if it is the projection on Σ of an ε-timed word $w' = (\lambda(1), t_1) \dots (\lambda(n), t_n)$ such that $t_j - t_i \in \theta(i,j)$ for all $(i,j) \in \rhd$. In other words, a TCW is realizable if there exists a timed word w which is a realization of W. For example, the timed word $(a, 0.9)(b, 2.1)(a, 2.1)(b, 3.9)(b, 5)$ is a realization of the TCW depicted above (left), while $(a, 1.2)(b, 2.1)$ $(a, 2.1)(b, 3.9)(b, 5)$ is not.

We can (and often will) view a TCW W as a *directed* weighted graph with edges $E = \rhd \cup \rhd^{-1} \cup \rightarrow^{-1}$ and weights induced by θ as follows: if $(i,j) \in \rhd$ and $\theta(i,j) = [I_\ell, I_r]$ then the weight of the *forward edge* is the upper constraint $\mathsf{wt}(i,j) = I_r$ and the weight of the *back edge* is the negative value of the lower constraint $\mathsf{wt}(j,i) = -I_\ell$. Further, to ensure that time is non-decreasing we add 0-weight back edges between consecutive positions that are not already constrained, i.e., if $(i,j) \in \lessdot \setminus \rhd$ then $\mathsf{wt}(j,i) = 0$. The directed weighted graph depicted above (right) corresponds to the TCW on its left. A *directed* path in W is a sequence of positions $\rho = p_1, p_2, \dots, p_n$ $(n > 1)$ linked with edges: $(p_i, p_{i+1}) \in E$ for all $1 \leq i < n$. It is a cycle or loop if $p_n = p_1$. Its weight is $\mathsf{wt}(\rho) = \sum_{1 \leq i < n} \mathsf{wt}(p_i, p_{i+1})$. Then, we have the following standard result:

▶ Proposition 2 ([7]). A TCW W is realizable iff it has no negative cycles.

Thus, to check if a TCW is realizable, we check for absence of negative weight cycles, which can be done in polynomial time, e.g., using the Bellman Ford algorithm (see [7] for details).

2.2 TPDA and their semantics as simple TCWs

Dense-timed pushdown automata (TPDA), introduced in [1], are an extension of timed automata, and operate on a finite set of real-valued clocks and a stack which holds symbols with their ages. The age of a symbol in the stack represents time elapsed since it was pushed on to the stack. Formally, a TPDA $\mathcal{S}$ is a tuple $(S, s_0, \Sigma, \Gamma, \Delta, X, F)$ where S is a finite set of states, $s_0 \in S$ is the initial state, Σ, Γ, are respectively a finite set of input, stack symbols, Δ is a finite set of transitions, X is a finite set of real-valued variables called clocks, $F \subseteq S$ are final states. A transition $t \in \Delta$ is a tuple $(s, \gamma, a, \mathsf{op}, R, s')$ where $s, s' \in S$, $a \in \Sigma$, γ is a finite conjunction of atomic formulae of the kind $x \in I$ for $x \in X$ and $I \in \mathcal{I}$, $R \subseteq X$ are clocks reset, op is one of the following stack operations:

1. nop does not change the contents of the stack,
2. $\downarrow_c$ where $c \in \Gamma$ is a push operation that adds c on top of the stack, with age 0.
3. $\uparrow_c^I$ where $c \in \Gamma$ is a stack symbol and $I \in \mathcal{I}$ is an interval, is a pop operation that removes the top most symbol of the stack provided it is a c with age in the interval I.

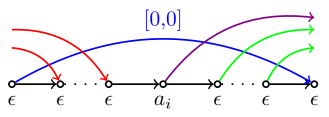
Timed automata (TA) can be seen as TPDA using nop operations only. This definition of TPDA is equivalent to the one in [1], but allows checking conjunctive constraints and stack operations together. In [6], it is shown that TPDA of [1] are expressively equivalent to timed automata with an untimed stack. Nevertheless, our technique is oblivious to whether the stack is timed or not, hence we focus on the syntactically more succinct model TPDA with timed stack and get good complexity bounds.

We define the semantics in terms of simple TCWs. An STCW $\mathcal{V} = (P, \rightarrow, \lambda, \rhd, \theta)$ is *generated* or *accepted* by a TPDA $\mathcal{S}$ if there is an accepting abstract run $\rho = (s_0, \gamma_1, a_1, \mathsf{op}_1, R_1, s_1)$ $(s_1, \gamma_2, a_2, \mathsf{op}_2, R_2, s_2) \cdots$ $(s_{n-1}, \gamma_n, a_n, \mathsf{op}_n, R_n, s_n)$ of $\mathcal{S}$ such that, $s_n \in F$ and

- the sequence of push-pop operations is well-nested: in each prefix $\mathsf{op}_1 \cdots \mathsf{op}_k$, number of pops is at most number of pushes, and in the full sequence $\mathsf{op}_1 \cdots \mathsf{op}_n$, they are equal.

- We have $P = P_0 \uplus P_1 \uplus \cdots \uplus P_n$ with $P_i \times P_j \subseteq \rightarrow^+$ for $0 \le i < j \le n$. Each transition $\delta_i = (s_{i-1}, \gamma_i, a_i, \mathsf{op}_i, R_i, s_i)$ gives rise to a sequence of consecutive points P_i in the STCW. The transition δ_i is simulated by a sequence of "micro-transitions" as depicted below (left) and it represents an STCW shown below (right). Incoming red edges check guards from γ_i (wrt different clocks) while outgoing green edges depict resets from R_i that will be checked later. Further, the outgoing edge on the central node labeled a_i represents a push operation on stack.

$$\begin{tikzcd} s_{i-1} \arrow[r, "\{\zeta\}"] & \delta_i^0 \arrow[r, "\gamma_i^1"] & \delta_i^1 \cdots \delta_i^{h_i-1} \arrow[r, "\gamma_i^{h_i}"] & \delta_i^{h_i} \arrow[r, "a_i, \mathsf{op}_i"] & \delta_i^{x_1} \arrow[loop above, "\{x_1\}"] \arrow[r, "\varepsilon"] & \cdots \delta_i^{x_m} \arrow[loop above, "\{x_m\}"] \arrow[r, "\zeta = 0"] & s_i \end{tikzcd}$$



where $\gamma_i = \gamma_i^1 \wedge \cdots \wedge \gamma_i^{h_i}$ and $R_i = \{x_1, \ldots, x_m\}$. The first and last micro-transitions, corresponding to the reset of a new clock ζ and checking of constraint $\zeta = 0$ ensure that all micro-transitions in the sequence occur simultaneously. We have a point in P_i for each micro-transition (excluding the ε-micro-transitions between $\delta_i^{x_j}$). Hence, P_i consists of a sequence $\ell_i \to \ell_i^1 \to \cdots \to \ell_i^{h_i} \to p_i \to r_i^1 \to \cdots \to r_i^{g_i} \to r_i$ where g_i is the number of timing constraints which are checked later, corresponding to clocks reset during transition i. Thus, the reset-loop on a clock is fired $k \ge 0$ times if k constraints are checked on this clock until its next reset. This ensures that the STCW remains *simple*. Similarly, h_i is the number of timing constraints conjuncted in γ_i. We have $\lambda(p_i) = a_i$ and all other points are labelled ε. The set P_0 encodes the initial resets of clocks that will be checked before their first reset. So we let $R_0 = X$ and P_0 is $\ell_0 \to r_0^1 \to \ldots \to r_0^{g_0} \to r_0$.
- The relation for timing constraints can be partitioned as $\rhd = \rhd^s \uplus \biguplus_{x \in X \cup \{\zeta\}} \rhd^x$ where
  - $\rhd^\zeta = \{(\ell_i, r_i) \mid 0 \le i \le n\}$ and we set $\theta(\ell_i, r_i) = [0,0]$ for all $0 \le i \le n$.
  - We have $p_i \rhd^s p_j$ if $\mathsf{op}_i = \downarrow_b$ is a push and $\mathsf{op}_j = \uparrow_b^I$ is the matching pop (same number of pushes and pops in $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1}$), and we set $\theta(p_i, p_j) = I$.
  - for each $0 \le i < j \le n$ such that the t-th conjunct of γ_j is $x \in I$ and $x \in R_i$ and $x \notin R_k$ for $i < k < j$, we have $r_i^s \rhd^x \ell_j^t$ for some $1 \le s \le g_i$ and $\theta(r_i^s, \ell_j^t) = I$. Therefore, every point ℓ_i^t with $1 \le t \le h_i$ is the target of a timing constraint. Moreover, every reset point r_i^s for $1 \le s \le g_i$ should be the source of a timing constraint: $r_i^s \in \mathsf{dom}(\rhd^x)$ for some $x \in R_i$. Also, for each i, the reset points $r_i^1, \ldots, r_i^{g_i}$ are grouped by clocks (as suggested by the sequence of micro-transitions simulating δ_i): if $1 \le s < u < t \le g_i$ and $r_i^s, r_i^t \in \mathsf{dom}(\rhd^x)$ for some $x \in R_i$ then $r_i^u \in \mathsf{dom}(\rhd^x)$. Finally, for each clock, we require that the timing constraints are well-nested: for all $u \rhd^x v$ and $u' \rhd^x v'$, with $u, u' \in P_i$, if $u < u'$ then $u' < v' < v$.

We denote by $\mathsf{STCW}(\mathcal{S})$ the set of simple TCWs generated by $\mathcal{S}$ and define the language of $\mathcal{S}$ as the set of *realizable* STCWs, i.e., $\mathcal{L}(\mathcal{S}) = \mathrm{Real}(\mathsf{STCW}(\mathcal{S}))$. Indeed, this is equivalent to defining the language as the set of timed words accepted by $\mathcal{S}$, according to a usual operational semantics [1]. The STCW semantics of timed automata (TA) can be obtained from the above discussion by just ignoring the stack components (using nop operations only).

We now identify some important properties satisfied by STCWs generated from a TPDA. Let $\mathcal{V} = (P, \to, \lambda, \rhd, \theta)$ be a STCW. We say that $\mathcal{V}$ is *well timed* w.r.t. a set of clocks Y and a stack s if the $\rhd$ relation can be partitioned as $\rhd = \rhd^s \uplus \biguplus_{x \in Y} \rhd^x$ where

(T_1) the relation $\rhd^s$ corresponds to the matching push-pop events, hence it is well-nested: for all $i \rhd^s j$ and $i' \rhd^s j'$, if $i < i' < j$ then $i' < j' < j$.

(T_2) For each $x \in Y$, the relation $\rhd^x$ corresponds to the timing constraints for clock x and is well-nested: for all $i \rhd^x j$ and $i' \rhd^x j'$, if $i < i'$ are in the same *x-reset block* (i.e., a maximal consecutive sequence $i_1 \lessdot \cdots \lessdot i_n$ of positions in the domain of $\rhd^x$), and $i < i' < j$, then $i' < j' < j$. Each guard should be matched with the closest reset block on its left: for all $i \rhd^x j$ and $i' \rhd^x j'$, if $i < i'$ are not in the same x-reset block then $j < i'$.

It is then easy to check that STCWs defined by a TPDA with set of clocks X are well-timed for the set of clocks $Y = X \cup \{\zeta\}$, i.e., satisfy the properties above. We obtain the same for TA by just ignoring the stack edges, i.e., (T_1) above.

3 Bounding the width of graph behaviors of timed systems

In this section, we check if the graphs (STCWs) introduced in the previous section have a bounded tree-width. As a first step towards that, we introduce *special tree terms* (STTs) from Courcelle [8] and their semantics as labeled graphs. It is known [8] that special tree terms using at most K colors (K-STTs) define graphs of "special" tree-width at most $K-1$. Formally, a (Σ, Γ)-labeled graph is a tuple $G = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda)$ where $\lambda \colon V \to \Sigma$ is the vertex labeling and $E_\gamma \subseteq V^2$ is the set of edges for each label $\gamma \in \Gamma$. Special tree terms form an algebra to define labeled graphs. The syntax of K-STTs over (Σ, Γ) is given by $\tau ::= (i, a) \mid \mathsf{Add}^{\gamma}_{i,j}\,\tau \mid \mathsf{Forget}_i\,\tau \mid \mathsf{Rename}_{i,j}\,\tau \mid \tau \oplus \tau$, where $a \in \Sigma$, $\gamma \in \Gamma$ and $i, j \in [K] = \{1, \ldots, K\}$ are colors. The semantics of each K-STT is a colored graph $[\![\tau]\!] = (G_\tau, \chi_\tau)$ where G_τ is a (Σ, Γ)-labeled graph and $\chi_\tau \colon [K] \to V$ is a partial injective function assigning a vertex of G_τ to atmost one color.

- $[\![(i, a)]\!]$ consists of a single a-labeled vertex with color i.
- $\mathsf{Add}^{\gamma}_{i,j}$ adds a γ-labeled edge to the vertices colored i and j (if such vertices exist).
- Forget_i removes color i from the domain of the color map.
- $\mathsf{Rename}_{i,j}$ exchanges the colors i and j.
- $\oplus$ is the disjoint union of two graphs if they use different colors and is undefined otherwise.

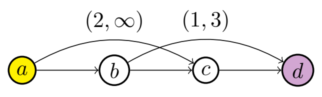
The *special tree-width* of a graph G is defined as the least K such that $G = G_\tau$ for some $(K+1)$-STT τ. See [8] for more details and its relation to tree-width. For TCWs, we have successor edges and $\rhd$-edges carrying timing constraints, so we take $\Gamma = \{\to\} \cup \{(x, y) \mid x \in \mathbb{N}, y \in \overline{\mathbb{N}}\}$ with $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$. In this paper, we will actually make use of STTs with the following restricted syntax, which are sufficient and make our proofs simpler:

$$\mathsf{atomicSTT} ::= (1, a) \mid \mathsf{Add}^{x,y}_{1,2}((1, a) \oplus (2, b))$$
$$\tau ::= \mathsf{atomicSTT} \mid \mathsf{Add}^{\to}_{i,j}\,\tau \mid \mathsf{Forget}_i\,\tau \mid \mathsf{Rename}_{i,j}\,\tau \mid \tau \oplus \tau$$

with $a, b \in \Sigma_\varepsilon$, $0 \le x < M$, $0 \le y < M$ or $y = +\infty$ for some $M \in \mathbb{N}$ and $i, j \in [2K] = \{1, \ldots, 2K\}$. The terms defined by this grammar are called (K, M)-STTs. Here, timing constraints are added directly between leaves in atomic STTs which are then combined using disjoint unions and adding successor edges. For instance, consider the 4-STT given below

$$\tau = \mathsf{Forget}_3\,\mathsf{Add}^{\to}_{1,3}\,\mathsf{Forget}_2\,\mathsf{Add}^{\to}_{2,4}\,\mathsf{Add}^{\to}_{3,2}(\mathsf{Add}^{2,\infty}_{1,2}((1, a) \oplus (2, c)) \oplus \mathsf{Add}^{1,3}_{3,4}((3, b) \oplus (4, d)))$$

where $\mathsf{Add}^{\gamma}_{i,j}((i, \alpha) \oplus (j, \beta))$, $i, j \in \mathbb{N}, \alpha, \beta \in \Sigma_\varepsilon$ is the same as $\mathsf{Rename}_{1,i}\,\mathsf{Rename}_{2,j}\,\mathsf{Add}^{\gamma}_{1,2}((1, \alpha) \oplus (2, \beta))$. Its semantics $[\![\tau]\!]$ is the adjoining STCW where only endpoints labelled a and d are colored, as the other two colors were "forgotten" by τ.Abusing notation, we will also use $[\![\tau]\!]$ for the graph G_τ ignoring the coloring χ_t.



Split-TCWs and split-game. We find it convenient to prove that a STCW has bounded special tree-width by playing a split-game, whose game positions are STCWs in which some successor edges have been cut, i.e., are missing. Formally, a *split*-TCW is a structure $\mathcal{V} = (P, \rightarrow, \dashrightarrow, \lambda, \rhd, \theta)$ where $\rightarrow$ and $\dashrightarrow$ are the present and absent successor edges (also called *holes*), respectively, such that $\rightarrow \cap \dashrightarrow = \emptyset$ and $(P, \rightarrow \cup \dashrightarrow, \lambda, \rhd, \theta)$ is a TCW. Notice that, for a split-TCW, $\lessdot = \rightarrow \cup \dashrightarrow$ and $< = \lessdot^+$. A *block* or *factor* of a split-TCW is a maximal set of points of P connected by $\rightarrow$. We denote by $\mathsf{EP}(\mathcal{V}) \subseteq P$ the set of left and right endpoints of blocks of $\mathcal{V}$. A left endpoint e is one for which there is no f with $f \rightarrow e$. Right endpoints are defined similarly. Points in $P \setminus \mathsf{EP}(\mathcal{V})$ are called internal. The number of blocks is the *width* of $\mathcal{V}$: $\mathsf{width}(\mathcal{V}) = 1 + |\dashrightarrow|$. TCWs may be identified with split-TCWs of width 1, i.e., with $\dashrightarrow = \emptyset$. A split-TCW is *atomic* if it consists of a single point ($|P| = 1$) or a single timing constraint with a hole ($P = \{p_1, p_2\}$, $p_1 \dashrightarrow p_2$, $p_1 \rhd p_2$). The directed weighted graph for a split-TCW is defined on the associated TCW under $\rightarrow \cup \dashrightarrow$ and hence has back edges with $\mathsf{wt} = 0$ across a hole as well.

The split-game is a two player turn based game $\mathcal{G} = (V_\exists \uplus V_\forall, E)$ where Eve's set of game positions $V_\exists$ consists of all connected (wrt. $\rightarrow \cup \rhd$) split-TCWs and Adam's set of game positions $V_\forall$ consists of non-connected split-TCWs. The edges E of $\mathcal{G}$ reflect the moves of the players. Eve's moves consist of splitting a factor in two, i.e., removing one successor edge in the graph. Adam's moves amount to choosing a connected component of the split-TCW. Atomic split-TCWs are terminal positions in the game: neither Eve nor Adam can move from an atomic split-TCW. A play on a split-TCW $\mathcal{V}$ is a path in $\mathcal{G}$ starting from $\mathcal{V}$ and leading to an atomic split-TCW. The cost of the play is the maximum width of any split-TCW encountered in the path. Eve's objective is to minimize the cost, while Adam's objective is to maximize it. Notice that Eve has a strategy to decompose a TCW $\mathcal{V}$ into *atomic* split-TCWs if and only if $\mathcal{V}$ is *simple*, i.e, at most one timing constraint is attached to each point. The *cost* of a strategy σ for Eve from a split-TCW $\mathcal{V}$ is the maximal cost of the plays starting from $\mathcal{V}$ and following strategy σ.

The *split-width* of a simple (split-)TCW $\mathcal{V}$ is the minimal cost of Eve's (positional) strategies starting from $\mathcal{V}$. Let STCW^K (resp. $\mathsf{STCW}^{K,M}$) denote the set of simple TCWs with split-width bounded by K (resp. and using constants at most M) over the fixed alphabet Σ. The crucial link between special tree-width and split-width is given below.

▶ **Lemma 3.** STCWs *of split-width at most K have special tree-width at most $2K - 1$.*

Intuitively, we only need to keep colors for end-points of blocks. Hence, each block of an STCW $\mathcal{V}$ needs at most two colors and if the width of $\mathcal{V}$ is at most K then we need at most $2K$ colors. From this it can be shown that a strategy of Eve of cost at most K can be encoded by a $2K$-STT, which gives a special tree-width of at most $2K - 1$.

Split-width for timed systems. Viewing special tree terms as trees, our goal in the next section is to construct tree automata to recognize sets of (K, M)-STTs, and thus capture (K split-width) bounded behaviors of a given system. To show that these capture *all* behaviors of the given system, we show that we can find K such that all (graph) behaviors of the given system have K-bounded split-width. We do this now for TPDA and timed automata.

▶ **Theorem 4.** *Given a timed system $\mathcal{S}$ using a set of clocks X, all words in its* STCW *language have split-width bounded by K, i.e.,* $\mathsf{STCW}(\mathcal{S}) \subseteq \mathsf{STCW}^K$*, where*

1. *$K = |X| + 4$ if $\mathcal{S}$ is a timed automaton,*
2. *$K = 4|X| + 6$ if $\mathcal{S}$ is a timed pushdown automaton,*

We prove a slightly more general result, by showing that all well-timed split-STCWs for the set of clocks $Y = X \cup \{\zeta\}$ have bounded split-width (lifting the definition of well-timed to split-STCWs). As noted earlier, the STCWs defined by a TPDA with set of clocks X are well-timed for the set of clocks $Y = X \cup \{\zeta\}$ and hence we obtain a bound on the split-width as required above. The following lemma completes the proof of Theorem 4 (2).

▶ **Lemma 5.** *The split-width of a well-timed* STCW *is bounded by* $4|Y| + 2$.

Proof (sketch). We prove this by playing the "split-width game" between *Adam* and *Eve* in which *Eve* has a strategy to disconnect the word without introducing more than $4|Y| + 2$ blocks. *Eve*'s strategy processes the word from right to left. We have three cases as follows.

Case (1) is when the last/right-most event, say j, is an internal point, i.e., it is not the target of a $\rhd$ relation. In this case, Eve will just split the process-edge before the last point with a single cut.

Case (2) is when the last event is the target of $\rhd^x$ for some clock $x \in Y$. In this case, she will detach the last timing constraint $i \rhd^x j$ where j is the last point of the split-TCW. By (T_2) we deduce that i is the *first* point of the *last reset block* for clock x. *Eve* splits three process-edges to detach the matching pair $i \rhd^x j$: these three edges are those connected to i and j. Since the matching pair $i \rhd^x j$ is atomic, to prolong the game *Adam* should choose the remaining split-TCW $\mathcal{V}'$. Note that we now have a hole instead of position i. We call this a reset-hole for clock x. During the inductive process, we have at most one such reset hole for each $x \in Y$, since the hole only widens in the reset block for each clock.

Note that the last event cannot be a push or the source of a timing constraint. So, the remaining Case (3) is a stack edge $i \rhd^s j$ where the pop event j is the last event of the split-TCW, details of which are in [3]. ◀

Now, if the STCW is from a timed automaton then, $\rhd^s$ is empty and Eve's strategy only has the first two cases above. Doing, this we obtain a bound of $|Y| + 3$ on split-width, which proves Theorem 4 (1).

4 The tree automata technique illustrated via TPDA and TA

We now describe our proof technique of using tree automata to analyze timed systems. At a high level, given a timed system $\mathcal{S}$ using constants less than M (say a timed automaton or a TPDA), we want to construct a tree automaton that accepts all (K, M)-STTs whose semantics are STCWs of split-width at most K which are realizable and accepted by $\mathcal{S}$. We break this into three parts. First, recall that STCWs of bounded split-width are graphs of bounded STTs (Lemma 3). However, not all graphs defined by bounded STTs are STCWs. We construct a tree automaton $\mathcal{A}^{K,M}_{\mathsf{valid}}$ which accepts only *valid* (K, M)-STTs, i.e., those representing STCWs of split-width at most K.

▶ Proposition 6. We can build a tree automaton $\mathcal{A}^{K,M}_{\mathsf{valid}}$ of size $\mathcal{O}(M) \cdot 2^{\mathcal{O}(K^2)}$ which accepts only (K, M)-STTs and such that $\mathsf{STCW}^{K,M} = \{[\![\tau]\!] \mid \tau \in \mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{valid}})\}$.

Our next step is to define a tree automaton $\mathcal{A}^{K,M}_{\mathsf{real}}$ which accepts all valid STTs whose semantics are *realizable* STCWs.

▶ Proposition 7. We can build a tree automaton $\mathcal{A}^{K,M}_{\mathsf{real}}$ of size $M^{\mathcal{O}(K^2)} \cdot 2^{\mathcal{O}(K^2 \lg K)}$ such that $\mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{real}}) = \{\tau \in \mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{valid}}) \mid [\![\tau]\!] \text{ is realizable}\}$.

Note that $\mathcal{A}^{K,M}_{\mathsf{real}}$ may not accept *all* (K, M)-STTs which denote realizable STCWs, but it will accept all such *valid* STTs. Once we have this, our third and final step is to build a tree automaton which accepts the valid STTs denoting STCWs accepted by the timed system.

▶ Proposition 8. Let $\mathcal{S}$ be a TPDA of size $|\mathcal{S}|$ (constants encoded in unary) with set of clocks X and using constants less than M. Then, we can build a tree automaton $\mathcal{A}_{\mathcal{S}}^{K,M}$ of size $|\mathcal{S}|^{\mathcal{O}(K^2)} \cdot 2^{\mathcal{O}(K^2(|X|+1))}$ such that $\mathcal{L}(\mathcal{A}_{\mathcal{S}}^{K,M}) = \{\tau \in \mathcal{L}(\mathcal{A}_{\mathsf{valid}}^{K,M}) \mid [\![\tau]\!] \in \mathsf{STCW}(\mathcal{S})\}$.

In Section 5, we detail the most complex tree automaton construction, $\mathcal{A}_{\mathsf{real}}^{K,M}$ for realizability, thus proving Proposition 7. The construction of $\mathcal{A}_{\mathsf{valid}}^{K,M}$ (Proposition 6) is somewhat similar (and easier) and we refer the reader to [3] for its details as well as the proof of (Proposition 8). We remark that for $\mathcal{A}_{\mathsf{valid}}^{K,M}, \mathcal{A}_{\mathcal{S}}^{K,M}$ we can also define an MSO formula and use Courcelle's theorem [9], but the direct tree automata construction gives us better control on complexity bounds and helps for $\mathcal{A}_{\mathsf{real}}^{K,M}$.

Thus, the tree automaton $\mathcal{A}$ checking ***ValCoRe*** (i.e., validity, correctness and realizability) is $\mathcal{A} = \mathcal{A}_{\mathsf{real}}^{K,M} \cap \mathcal{A}_{\mathcal{S}}^{K,M}$. We have $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff there exist some *realizable* STCWs in $\mathsf{STCW}(\mathcal{S}) \cap \mathsf{STCW}^{K,M}$. Since checking emptiness of a finite tree automaton is decidable in PTIME, we obtain that emptiness is decidable for the corresponding timed system restricted to STCWs of split-width at most K.

▶ **Theorem 9.** *Checking whether the timed system $\mathcal{S}$ accepts a* realizable STCW *of split-width at most K is decidable.*

By Theorem 4, all STCWs in the semantics of a TPDA $\mathcal{S}$ have split-width bounded by some fixed K and Theorem 9 gives a complete decision procedure for checking emptiness of TPDA. From these bounds on split-width and the size of the tree automata for validity, realizability and the system given in the above propositions, we obtain ExpTime decision procedures for checking emptiness of TPDA.

In the above technique, the only system-specific component is the automaton $\mathcal{A}_{\mathcal{S}}^{K,M}$ for the timed system $\mathcal{S}$. However, Proposition 8 can easily be adapted for timed automata and for several other timed systems, which are discussed in Section 6. Hence, this technique is generic and can be used for several other timed systems.

Moreover, for timed automata, it can be seen, for instance, from the analysis of Cases (1) and (2) of proof of Lemma 5 that one of the connected components (the pair $i \rhd^x j$) is always atomic. Therefore the split-tree is "word-like", i.e., for each binary node, one subtree is small, in our case atomic. Therefore, we can encode the subtree in the label of the binary node itself and use word automata instead of tree automata to check for emptiness (in NLogSpace instead of PTime), yielding the complexity stated below.

▶ **Corollary 10.** *Emptiness of TPDA and TA are decidable in* ExpTime *and* PSpace *respectively.*

5 Tree automata for realizable valid (K, M)-STTs

Our goal in this section is to define a finite bottom-up tree automaton $\mathcal{A}_{\mathsf{real}}^{K,M}$ that runs on (K, M)-STTs and accepts only *valid* (K, M)-STTs whose semantics are realizable STCWs. Let us first give a high-level picture. A state of the tree automaton will be a split-TCW with at most K blocks and $2K$ points. At any stage of the run, while processing a subtree τ of the (K, M)-STT, the state, i.e., split-TCW q reached will be a finite abstraction of the split-TCW $[\![\tau]\!]$ generated by τ, such that q is valid and realizable iff the TCW $[\![\tau]\!]$ is. At a leaf, the state of an atomic-STT is just a single matching edge with a hole. At each subsequent step going up, the tree automaton simulates the operations of τ: at a $\oplus$ move, it combines two split-TCW q_1 and q_2 to form a new valid split-TCW q by guessing an ordering between the blocks such that no new negative cycle is introduced (i.e., q continues to be realizable), and at an $\mathsf{Add}_{i,j}^{\rightarrow}$ node, it adds a process edge to fill up the corresponding hole in the split-TCW. At a Forget_i node, it removes an internal point, but to maintain realizability, the constraints on internal positions must be propagated to the end-points of the block and this process is continued. Finally, at the root, we obtain a TCW which is a finite abstraction of the semantics $[\![\tau]\!]$ of a valid (K, M)-STT τ such that $[\![\tau]\!]$ is a realizable TCW. Then, we show that the tree automaton accepts all such (K, M)-STTs, which concludes the proof of Proposition 7. There are two key difficulties that we have glossed over in this sketch:

- first, the propagation of constraints can increase the bounds arbitrarily, along an arbitrarily long (even if finite) run. Fixing this is the hardest part and we carefully define abstractions that bound the constraints by a constant $M' = \mathcal{O}(M)$, while preserving realizability.
- This leads to another subtle issue: while checking that realizability is preserved under our operations (of combining split-TCW and adding process edges), it is no longer sufficient to just check whether this combination is "safe". It may be that currently no negative cycle is formed, but at a later stage, some other operation ($\oplus$) gives rise to a negative cycle, which we do not observe since we capped the value of timing constraints. So, we need to show that all operations are safe no matter what happens in the future. For this we start by defining the notion of preserving realizability "under all contexts" as well as the formal notion of a "shuffle" used at $\oplus$ nodes.

Shuffle and Realizability under contexts. Let $\mathcal{V}_1 = (P_1, \rightarrow_1, \dashrightarrow_1, \lambda_1, \rhd_1, \theta_1)$ and $\mathcal{V}_2 = (P_2, \rightarrow_2, \dashrightarrow_2, \lambda_2, \rhd_2, \theta_2)$ be two split-TCWs such that their respective set of positions P_1 and P_2 are disjoint. Further, let $\leq$ be a total order on $P = P_1 \cup P_2$ such that $\dashrightarrow_1 \cup \dashrightarrow_2 \subseteq <$ and $\rightarrow_1 \cup \rightarrow_2 \subseteq \lessdot$. Such orders are called *admissible*. Then, we define the split-TCW $\mathcal{V} = (P, \rightarrow, \dashrightarrow, \lambda, \rhd, \theta)$ by $P = P_1 \uplus P_2$, $\lambda = \lambda_1 \cup \lambda_2$, $\rightarrow = \rightarrow_1 \cup \rightarrow_2$, $\dashrightarrow = \lessdot \setminus \rightarrow$, $\rhd = \rhd_1 \cup \rhd_2$, and $\theta = \theta_1 \cup \theta_2$. Indeed, this corresponds to *shuffling* the blocks $\mathcal{V}_1$ and $\mathcal{V}_2$ with respect to the admissible order $\leq$ and is called a shuffle, denoted by $\mathcal{V} = \mathcal{V}_1 \sqcup\!\sqcup_{\leq} \mathcal{V}_2$.

Let M be a positive integer. An *M-context* C is a split-TCW such that the maximal constant in the intervals is strictly smaller than the fixed constant M. Given a context C and a split-TCW $\mathcal{V}$, we define an operation $C \circ \mathcal{V}$ if $\mathsf{width}(C) = \mathsf{width}(\mathcal{V}) + 1$. $C \circ \mathcal{V}$ is the split-TCW obtained by *shuffling* the blocks of C and $\mathcal{V}$ in strict alternation. Two split-TCWs U and V are *equivalent*, denoted $U \sim_M V$, iff they have the same number of blocks and preserve realizability under all M-contexts. That is, there exists $k \in \mathbb{N}$ such that $\mathsf{width}(U) = \mathsf{width}(V) = k$ and for all M-contexts $C \in \mathsf{STCW}$ with $\mathsf{width}(C) = k + 1$, $C \circ U$ is realizable iff $C \circ V$ is realizable. It is easy to see that $\sim_M$ is an equivalence relation. A function $f : \mathsf{STCW} \to \mathsf{STCW}$ is said to be *sound* if it preserves realizability under all M-contexts, i.e., for all $W \in \mathsf{STCW}$ we have $W \sim_M f(W)$. The idea is to define a sound abstraction of finite index, so that a finite tree automaton can work only on the abstractions.

▶ **Lemma 11.** (Congruence lemma) *Let U_1, U_2, U_1' and U_2' be split-*TCWs *such that $U_1 \sim_M U_1'$ and $U_2 \sim_M U_2'$. Then, $U_1 \sqcup\!\sqcup_{\leq} U_2 \sim_M U_1' \sqcup\!\sqcup_{\leq} U_2'$ for all admissible orders $\leq$ on the blocks.*

A (possibly infinite) tree automaton for realizability. We now build the tree automaton for realizability in two steps. First, we detail a construction which is correct and sound (i.e., preserves realizability under all contexts), but in which constants can grow unboundedly. Subsequently, we show conditions under which it has finitely many states and additional abstractions to ensure finiteness.

▶ Proposition 12. We can build a tree automaton $\mathcal{A}^{K,M}_{\mathsf{inf}}$ such that $\mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{inf}}) = \{\tau \in \mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{valid}}) \mid [\![\tau]\!] \text{ is realizable}\}$.

Proof. The construction builds on the construction of $\mathcal{A}^{K,M}_{\mathsf{valid}}$, which is detailed in [3]. The states of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ are pairs (q, wt) where $q = (P, <, \rightarrow)$ is a state of $\mathcal{A}^{K,M}_{\mathsf{valid}}$, i.e., $P \subseteq [2K]$, $<$ is a total order on P, $\rightarrow \subseteq \lessdot$ is the successor relation between points in the same block, q has at most K blocks; and $\mathsf{wt}\colon P^2 \to \overline{\mathbb{Z}} = \mathbb{Z} \cup \{+\infty\}$ gives the timing constraints. The first component is finite but weights can grow unboundedly. We assume $\mathsf{wt}(k,k) = 0$ for all $k \in P$ and if $i < j$ then $\mathsf{wt}(j,i) \le 0 \le \mathsf{wt}(i,j)$. We identify (q, wt) with a split-TCW (ignoring $\rhd, \Sigma$, as these are irrelevant for realizability).

We first give the invariant that will be maintained by the automaton. Let τ be a (K, M)-STT with $[\![\tau]\!] = (V, \rightarrow, \lambda, \rhd, \theta, \chi)$. If a (bottom-up) run of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ reads τ and reaches state (q, wt) with $q = (P, <, \rightarrow)$, it induces a total order on blocks of $[\![\tau]\!]$ and turns it into a split-TCW $([\![\tau]\!], \dashrightarrow)$. We say that the abstraction (q, wt) of τ computed by $\mathcal{A}^{K,M}_{\mathsf{inf}}$ is *sound* if it preserves realizability under contexts, i.e., $([\![\tau]\!], \dashrightarrow) \sim_M (q, \mathsf{wt})$. *The key invariant is that $\mathcal{A}^{K,M}_{\mathsf{inf}}$ always computes a sound abstraction of the given* STT. We now formalize the definition of the tree automaton.

- AtomicSTTs: When reading the atomic STT $\tau = (1, a)$ with $a \in \Sigma$, $\mathcal{A}^{K,M}_{\mathsf{inf}}$ moves to state (q, wt) where $q = (\{1\}, \emptyset, \emptyset)$ and $\mathsf{wt}(1,1) = 0$. Similarly, when reading an atomic STT $\tau = \mathsf{Add}^{c,d}_{1,2}((1,a) \oplus (2,b))$, $\mathcal{A}^{K,M}_{\mathsf{inf}}$ moves to state (q, wt) where $q = (\{1,2\}, 1 < 2, \emptyset)$, $\mathsf{wt}(1,1) = 0 = \mathsf{wt}(2,2)$, $\mathsf{wt}(1,2) = d$ and $\mathsf{wt}(2,1) = -c$. In both cases, it is easy to check that (q, wt) is a sound abstraction of τ.
- $\mathsf{Rename}_{i,j}$: We define transitions $(q, \mathsf{wt}) \xrightarrow{\mathsf{Rename}_{i,j}} (q', \mathsf{wt}')$ where (q', wt') is obtained by exchanging colors i and j in (q, wt), which clearly preserves soundness.
- $\mathsf{Add}^{\rightarrow}_{i,j}$: We define $(q, \mathsf{wt}) \xrightarrow{\mathsf{Add}^{\rightarrow}_{i,j}} (q', \mathsf{wt})$, when q' is obtained from $q = (P, <, \rightarrow)$ by adding a successor edge $(i,j) \in \lessdot \setminus \rightarrow$. Then, if $\tau' = \mathsf{Add}^{\rightarrow}_{i,j}\, \tau$ and (q, wt) is a sound abstraction of τ, it follows that (q', wt') is a sound abstraction of τ' (adding edges only reduces number of contexts to be considered to show equivalence of realizability under contexts.)
- $\oplus$: We define transitions $(q_1, \mathsf{wt}_1), (q_2, \mathsf{wt}_2) \xrightarrow{\oplus} (q, \mathsf{wt})$ when $q = (P, <, \rightarrow)$ is a shuffle of q_1 and q_2 and for all $i, j \in P = P_1 \uplus P_2$, $\mathsf{wt}(i,j)$ is $\mathsf{wt}_1(i,j)$ if $i,j \in P_1$ and $\mathsf{wt}_2(i,j)$ if $i,j \in P_2$. If they do not come from the same state, i.e., if $(i,j) \in (P_1 \times P_2) \cup (P_2 \times P_1)$, then $\mathsf{wt}(i,j)$ is ∞ if $i < j$ and 0 otherwise, i.e., $i \ge j$. Now, if $\tau = \tau_1 \oplus \tau_2$ and $(q_1, \mathsf{wt}_1), (q_2, \mathsf{wt}_2)$ are sound abstractions of τ_1, τ_2 then (q, wt) is a sound abstraction of τ. The total ordering $<$ of q indicates how blocks of q_1 and q_2 are shuffled. Hence $(q, \mathsf{wt}) = (q_1, \mathsf{wt}_1) \sqcup\!\sqcup_{\le} (q_2, \mathsf{wt}_2)$. Now, the induced ordering on the blocks of $[\![\tau]\!]$ corresponds to the same shuffle of blocks, i.e., $([\![\tau]\!], \dashrightarrow) = ([\![\tau_1]\!], \dashrightarrow_1) \sqcup\!\sqcup_{\le} ([\![\tau_2]\!], \dashrightarrow_2)$. Now, applying the congruence Lemma 11, we obtain that (q, wt) is a sound abstraction of τ.
- Forget_i: We define transitions $(q, \mathsf{wt}) \xrightarrow{\mathsf{Forget}_i} (q', \mathsf{wt}')$ when the following hold
  - i is not an endpoint, q' is obtained from $q = (P, <, \rightarrow)$ by removing internal point i,
  - i is not part of a negative cycle of length 2: for all $j \neq i$ we have $\mathsf{wt}(j,i) + \mathsf{wt}(i,j) \ge 0$,
  - for all $j, k \in P' = P \setminus \{i\}$, we define $\mathsf{wt}'(j,k) = \min(\mathsf{wt}(j,k), \mathsf{wt}(j,i) + \mathsf{wt}(i,k))$, i.e., wt' is obtained by eliminating i.

  If the second condition above is not satisfied then the tree automaton $\mathcal{A}^{K,M}_{\mathsf{inf}}$ has no transitions from (q, wt) reading Forget_i. With this we can prove that if $\tau' = \mathsf{Forget}_i\, \tau$ and (q, wt) is a sound abstraction of τ, then (q', wt') is a sound abstraction of τ'.
- Accepting condition: Finally, we define a state (q, wt) to be accepting if q consists of a single block with no internal points, left endpoint i and right endpoint j (possibly $i = j$), and the pair (q, wt) is realizable, i.e., $\mathsf{wt}(i,j) + \mathsf{wt}(j,i) \ge 0$.

We can now check that $\mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{inf}}) = \{\tau \in \mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{valid}}) \mid [\![\tau]\!] \text{ is realizable}\}$. ◀

Observe that the constants in wt' increase only at *forget* transitions, where a back edge $j > k$ with $j > i > k$ grows in absolute value with the update $\mathsf{wt}'(j,k) = \min(\mathsf{wt}(j,k), \mathsf{wt}(j,i) + \mathsf{wt}(i,k))$. A forward edge $j < k$ may get a big value only if $\mathsf{wt}(j,k) = \infty$, else it can only decrease due to the min operation. A first question is if there are classes where they will not grow *unboundedly.* A simple solution is to consider time-bounded classes where all behaviors must occur within some global time bound T: if some back edge grows $> T$ in absolute value after a forget move we reject the STT; while if the same happens with a forward edge, then replace it with ∞. Thus, we obtain,

▶ **Corollary 13.** *If the system is time-bounded by some constant T, then there exists a finite tree automaton $\mathcal{A}^{K,M}_{\mathsf{real}}$ of size at most $T^{\mathcal{O}(K^2)} \cdot 2^{\mathcal{O}(K^2 \lg K)}$ for checking realizability.*

However, when we do not assume a global time bound the constants in the states of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ may grow unboundedly. We next show how to modify the above construction so that the constants are always bounded. This generalizes the above corollary with a better complexity.

Bounding the constants. The finite tree automaton $\mathcal{A}^{K,M}_{\mathsf{real}}$ will work on a finite subset of the states of $\mathcal{A}^{K,M}_{\mathsf{inf}}$. More precisely, a state (q, wt) of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ with $q = (P, <, \rightarrow)$ is a state of $\mathcal{A}^{K,M}_{\mathsf{real}}$ if for all $i, j \in P$ we have $\mathsf{wt}(i,j) = +\infty$ or $|\mathsf{wt}(i,j)| \leq 8KM$.

Now, to bound back edges we define a transformation β which reduces the weight of a back edge when it goes above a certain constant, while *preserving realizability under all contexts.* In fact, we define it on back edges across a block. Let (q, wt) be a state of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ with $q = (P, <, \rightarrow)$. A pair of points $(j, i) \in P^2$ is said to be a *block back edge* (denoted BBE) if $i < j$ are the end points of a block in q, i.e., $i \rightarrow^{+} j$ and this $\rightarrow$-path cannot be extended (on the left or on the right). A *big block back edge (*BBBE*)* is block back edge e such that $M + \mathsf{wt}(e) \leq 0$. For any two positions $i < j$, we define $\mathsf{BBE}(i,j)$ to be the set of block back edges between i and j. That is, $\mathsf{BBE}(i,j) = \{(\ell, k) \mid (\ell, k) \text{ is a BBE and } i \leq k < \ell \leq j\}$. We also define $\mathcal{B}(i,j)$ to be the set of big block back edges between i and j: $\mathcal{B}(i,j) = \{e \in \mathsf{BBE}(i,j) \mid e \text{ is big}\}$. We now define $\beta(q, \mathsf{wt}) = (q, \mathsf{wt}')$ where, for any $i < j$,

$$\mathsf{wt}'(i,j) = \mathsf{wt}(i,j) + \sum_{e \in \mathcal{B}(i,j)} (M + \mathsf{wt}(e)) \qquad \mathsf{wt}'(j,i) = \mathsf{wt}(j,i) - \sum_{e \in \mathcal{B}(i,j)} (M + \mathsf{wt}(e))$$

The idea is to change the weight of *big* BBE to $-M$ by adding an offset to all the other edges (backward and forward) crossing this block. Note that this does not increase the absolute value of *any* constant. Further, after the backward abstraction, the absolute value of weights of block back edges is bounded by M, i.e., for all BBE $i \curvearrowleft j$, we have $\mathsf{wt}'(j,i) \geq -M$. Indeed, either the edge was big and we get $\mathsf{wt}'(j,i) = -M$ or it was not big and $\mathsf{wt}'(j,i) = \mathsf{wt}(j,i) > -M$. Notice also that a BBE is big in (q, wt) iff it is big in $\beta(q, \mathsf{wt})$. The crucial property is that we leave the weights of all cycles unchanged (under all contexts).

▶ **Lemma 14.** *For all states $W = (q, \mathsf{wt})$ of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ with $q = (P, <, \rightarrow)$ such that all points are endpoints $P = \mathsf{EP}(W)$, we have $W \sim_M \beta(W)$.*

While block back edges are now bounded (and back edges across holes can also be bounded by $-M$), this does not suffice to bound all back edges. To obtain such a bound on all back edges, we need to relate large back edges to edges contained within them.

▶ **Definition 15.** A split-TCW W is said to satisfy the *back edge property* (BEP) if for all $i \leq j \leq k \leq \ell$ with either $j \dashrightarrow k$ or $j = k$, we have $\mathsf{wt}(\ell, i) > \mathsf{wt}(\ell, k) - M + \mathsf{wt}(j, i)$.

With this, we have our second and crucial invariant, that we maintain inductively in the tree automaton, (I2): $\mathcal{A}_{\mathsf{real}}^{K,M}$ always satisfies BEP. Preserving this invariant requires a slight transformation of the shuffle operation (at a $\oplus$ node). Namely, after every shuffle we must *strengthen* the constraints of the back edges. Formally, we define a map σ, $\sigma(q, \mathsf{wt}) = (q, \mathsf{wt}')$ where for all $i < j$, $\mathsf{wt}'(i,j) = \mathsf{wt}(i,j)$ and $\mathsf{wt}'(j,i) = \min\{\mathsf{wt}(j',i') \mid i \le i' \le j' \le j\}$ and perform this after every $\oplus$ move of the tree automaton. It is easy to check that σ preserves realizability under contexts and this allows us to show that the invariant (I2) is preserved. Now, under the BEP assumption, we can show that all back edges are bounded,

▶ **Lemma 16.** *Let $W = (q, \mathsf{wt})$ be a state of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ with $q = (P, <, \rightarrow)$ such that $P = \mathsf{EP}(W)$. If $\beta(W)$ satisfies* BEP*, then the weight of all back edges in $\beta(W)$ are bounded by $2KM$.*

Finally, *forward abstraction* γ removes all forward edges (i.e., changes their weight to ∞) that are too large to be useful for creating negative cycles. Let $W = (q, \mathsf{wt})$ be a state of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ with $q = (P, <, \rightarrow)$. A forward edge $(i,j) \in P^2$ with $i < j$ is called *big* if $\mathsf{wt}(i,j) + \sum_{e \in \mathsf{BBE}(i,j)} \mathsf{wt}(e) \ge (3K-1)M$. Note, $\mathsf{wt}(e) \le 0$ as it is a (block) back edge. Then, we define $\gamma(q, \mathsf{wt}) = (q, \mathsf{wt}')$ where, for any $i < j$, $\mathsf{wt}'(j,i) = \mathsf{wt}(j,i)$ and $\mathsf{wt}'(i,j) = \infty$ if (i,j) is *big* and unchanged otherwise. While the definition of this abstraction is simple, showing that it is sound (i.e., preserves realizability under all contexts) is rather tricky. We have,

▶ **Lemma 17.** *If $W = (q, \mathsf{wt})$ is a state of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ which satisfies BEP, then we have $W \sim_M \gamma(W)$.*

Thus, $\mathcal{A}_{\mathsf{real}}^{K,M}$ is derived from $\mathcal{A}_{\mathsf{inf}}^{K,M}$ by applying the abstractions at $\oplus$ nodes and at Forget_i nodes. More precisely, $(q_1, \mathsf{wt}_1), (q_2, \mathsf{wt}_2) \xrightarrow{\oplus} \sigma(q, \mathsf{wt})$ is in $\mathcal{A}_{\mathsf{real}}^{K,M}$ if $(q_1, \mathsf{wt}_1), (q_2, \mathsf{wt}_2) \xrightarrow{\oplus} (q, \mathsf{wt})$ is in $\mathcal{A}_{\mathsf{inf}}^{K,M}$. Similarly, if $(q, \mathsf{wt}) \xrightarrow{\mathsf{Forget}_i} (q', \mathsf{wt}')$ is a transition in $\mathcal{A}_{\mathsf{inf}}^{K,M}$ then $(q, \mathsf{wt}) \xrightarrow{\mathsf{Forget}_i} (q'', \mathsf{wt}'')$ is in $\mathcal{A}_{\mathsf{real}}^{K,M}$ where $(q'', \mathsf{wt}'') = \gamma(\beta(q', \mathsf{wt}'))$ if q' has no internal points and $(q'', \mathsf{wt}'') = (q', \mathsf{wt}')$ otherwise. The reason for assuming that q' has no internal points before applying the abstractions is that it is a precondition for Lemmas 14 and 16. Note that reachable states of $\mathcal{A}_{\mathsf{valid}}^{K,M}$ (and hence $\mathcal{A}_{\mathsf{real}}^{K,M}$) can have at most two internal points. Thus, along a run, if a state (q, wt) has no internal points, then the constants are bounded by $4KM$, otherwise, the constants are bounded by $8KM$. Thus the constants never exceed $8KM$ in states of $\mathcal{A}_{\mathsf{real}}^{K,M}$, which bounds our state space.

Since the transformations σ, β, γ preserve realizability under contexts (Lemma 14 and Lemma 17) we conclude that the key invariant holds, i.e., $\mathcal{A}_{\mathsf{real}}^{K,M}$ always computes a sound abstraction of the given STT. The acceptance condition of $\mathcal{A}_{\mathsf{real}}^{K,M}$ is the same as for $\mathcal{A}_{\mathsf{inf}}^{K,M}$, and the correctness follows as for $\mathcal{A}_{\mathsf{inf}}^{K,M}$. This completes the proof of Proposition 7.

6 Discussion and Future work

The main contribution of this paper is the technique for analyzing timed systems via tree automata. For simplicity, we only considered closed intervals in this paper, but our technique can be easily adapted to work for all kinds of intervals, i.e., open, half-open etc. Similarly, diagonal constraints of the form $x - y \in I$ can be handled easily by adding matching edges and changing $\mathcal{A}_{\mathcal{S}}^{K,M}$ appropriately.

As another application of our technique, we now consider the model of dense-timed multi-stack pushdown automata (dtMPDA), which have several stacks. The reachability problem for untimed multi-stack pushdown automata (MPDA) is already undecidable, but several restrictions have been studied on (untimed) MPDA, like bounded rounds [14], bounded phase, bounded scope and so on to regain decidability. We look at dtMPDA with the restriction of "bounded rounds". To the best of our knowledge, this timed model has not been investigated until now. Under this restriction, any run of a dtMPDA can be broken into a finite number of rounds, such that in each round only a single stack is used. As before, the sequence of push-pop operations of any stack must be well-nested. Now, lifting the definition of well-timed STCWs to k-round well-timed STCWs, we can show that such STCWs have split-width at most $(4nk+4)(|X|+2)$, where n is the number of stacks. Thus all STCWs generated by runs of dtMPDA using at most k rounds have a bounded split-width. Now, by modifying $\mathcal{A}_{S}^{K,M}$ appropriately (see [3] for details), we obtain

▶ **Theorem 18.** *Checking emptiness for k-round* dtMPDA *is decidable in* EXPTIME.

We believe that our techniques can be extended to other restrictions for dtMPDA such as bounded scope and phase and to the more general model [13] of recursive hybrid automata. Another interesting direction is to use our technique to go beyond reachability and show results on model checking for timed systems. While model-checking against untimed specifications is easy to obtain with our approach, the challenge is to extend it to timed specifications.

References

1 P. Abdulla, M. F. Atig, and J. Stenman. Dense-timed pushdown automata. In *LICS Proceedings*, pages 35–44, 2012.

2 C. Aiswarya and P. Gastin. Reasoning about distributed systems: WYSIWYG (invited talk). In *FSTTCS Proceedings*, pages 11–30, 2014.

3 S. Akshay, P. Gastin, and S. N. Krishna. Analyzing timed systems using tree automata. *CoRR*, abs/1604.08443, 2016. URL: `http://arxiv.org/abs/1604.08443`.

4 R. Alur and D. Dill. A theory of timed automata. *In TCS*, 126(2):183–235, 1994.

5 M. F. Atig. Model-checking of ordered multi-pushdown automata. *LMCS*, 8(3), 2012.

6 L. Clemente and S. Lasota. Timed pushdown automata revisited. In *LICS Proceedings*, pages 738–749, 2015.

7 T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2nd edition, 2001.

8 B. Courcelle. Special tree-width and the verification of monadic second-order graph properties. In *FSTTCS Proceedings*, pages 13–29, 2010.

9 B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications.* CUP, 2012.

10 A. Cyriac. *Verification of Communicating Recursive Programs via Split-width.* Thèse de doctorat, LSV, ENS Cachan, January 2014.

11 A. Cyriac, P. Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR Proceedings*, pages 547–561, 2012.

12 W. Czerwinski, P. Hofman, and S. Lasota. Reachability problem for weak multi-pushdown automata. In *CONCUR Proceedings*, pages 53–68. Springer, 2012.

13 S. N. Krishna, L. Manasa, and A. Trivedi. What's decidable about recursive hybrid automata? In *HSCC Proceedings*, pages 31–40, 2015.

14 S. La Torre, P. Madhusudan, and G. Parlato. The language theory of bounded context-switching. In *LATIN Proceedings*, pages 96–107, 2010.

15 S. La Torre, M. Napoli, and G. Parlato. Scope-bounded pushdown languages. In *DLT Proceedings*, pages 116–128. Springer, 2014.

16 S. La Torre, M. Napoli, and G. Parlato. A unifying approach for multistack pushdown automata. In *MFCS Proceeedings*, pages 377–389. Springer, 2014.

17 P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL Proceedings*, pages 283–294, 2011.