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,krishnas}@cse.iitb.ac.in`
2 **LSV, ENS-Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France**
`paul.gastin@lsv.ens-cachan.fr`

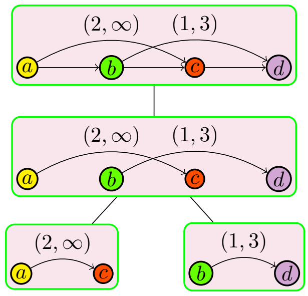
Abstract. Timed systems, such as timed automata, are usually analyzed using their operational semantics on timed words. The classical region abstraction for timed automata reduces them to (untimed) finite state automata with the same time-abstract properties, such as state reachability. We propose a 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).

1 Introduction

The advent of timed automata [3] 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 [5], 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 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 an abstract 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, we conjecture that realizability is not MSO definable over words with timing constraints. Given this, we cannot directly appeal to 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 ($|X|+2$ and $4|X|+2$) that is linear in the number of clocks. 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 a very active research area [4, 12, 14, 16, 15], and our technique can easily 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.

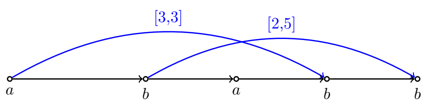
2 Graphs for behaviors of timed systems

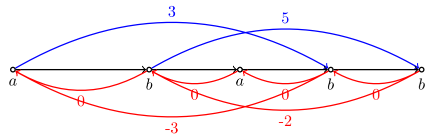
We fix an alphabet Σ and use Σ_ε to denote $\Sigma \cup \{\varepsilon\}$ where ε is the silent action. We also fix a finite set of closed intervals $\mathcal{I}$ which contains the special interval $[0, 0]$. For a set S, we use $\leq \subseteq S \times S$ to denote a partial or total order on S. For 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 $\mathcal{V} = (P, \rightarrow, \lambda, \triangleright, \theta)$ where P is a finite set of positions or points, $\lambda \colon P \to \Sigma_\varepsilon$ labels each position, the reflexive transition closure $\leq\, = \,\rightarrow^*$ is a total order on P and $\rightarrow\, = \,\lessdot$ is the successor relation, $\triangleright \subseteq\, <\, = \,\rightarrow^+$ gives the pairs of positions carrying a timing constraint, whose interval is given by $\theta \colon \triangleright \to \mathcal{I}$.

For any position $i \in P$, the *indegree* (resp. *outdegree*) of i is the number of positions j such that $(j, i) \in \triangleright$ (resp. $(i, j) \in \triangleright$). 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 $\triangleright$, while straight edges are the successor relation $\rightarrow$. 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 $(t_i)_{1 \leq i \leq n}$ is 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 $W = (P, \rightarrow, \lambda, \triangleright, \theta)$ 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 $w' = (\lambda(1), t_1) \ldots (\lambda(n), t_n)$ such that $t_j - t_i \in \theta(i, j)$ for all $(i, j) \in \triangleright$. 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 $E = \triangleright \cup \triangleright^{-1} \cup \rightarrow^{-1}$ and weights induced by θ as follows: if $(i, j) \in \triangleright$ 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 $(i, j) \in \lessdot \setminus \triangleright$ 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 $\rho = p_1, p_2, \ldots, p_n$ $(n > 1)$ 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.

Proof. This follows from the well-known method for checking feasibility of systems of difference constraints using constraint graphs. The characterization can be found in any standard textbook on algorithms and constraint solving, for instance, see [7, Chapter 24.4]. (We have additional constraints of the form $x_i \geq 0$, but this is easy to handle since if $(s_1, \ldots, s_n)$ is a solution vector then $(s_1 + d, \ldots; s_n + d)$ is also a solution vector for all constants d.) □

To check if a TCW is realizable, we only need to check for absence of negative weight cycles, which can be done in polynomial time, for instance, 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 the clocks reset, op is one of the following stack operations:

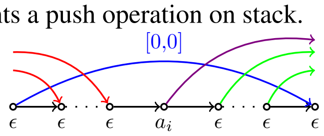
1. nop does not change the contents of the stack,
2. $\downarrow_c$ where $c \in \Gamma$ is a push operation that adds c on top of the stack, with age 0.
3. $\uparrow_c^I$ where $c \in \Gamma$ is a stack symbol and $I \in \mathcal{I}$ is an interval, is a pop operation that removes the top most symbol 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 $\mathcal{V} = (P, \to, \lambda, \triangleright, \theta)$ is said to be 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 $\mathsf{op}_1 \cdots \mathsf{op}_k$ with $1 \leq k \leq n$, 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 $P_i \times P_j \subseteq \to^+$ for $0 \leq i < j \leq n$. 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.

$$s_{i-1} \xrightarrow{\{\zeta\}} \delta_i^0 \xrightarrow{\gamma_i^1} \delta_i^1 \ \cdots \ \delta_i^{h_i-1} \xrightarrow{\gamma_i^{h_i}} \delta_i^{h_i} \xrightarrow{a_i, \mathsf{op}_i} \overset{\{x_1\}}{\delta_i^{x_1}} \xrightarrow{\varepsilon} \cdots \overset{\{x_m\}}{\delta_i^{x_m}} \xrightarrow{\zeta = 0} s_i$$



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 corresponding to clocks reset during transition i and checked afterwards. Similarly, h_i is the number of timing constraints checked 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 being 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 $\triangleright = \triangleright^s \uplus \biguplus_{x \in X \cup \{\zeta\}} \triangleright^x$ where

- $\rhd^\zeta = \{(\ell_i, r_i) \mid 0 \leq i \leq n\}$ and we set $\theta(\ell_i, r_i) = [0,0]$ for all $0 \leq i \leq n$.
- 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 $0 \leq i < j \leq n$ 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 $1 \leq s \leq g_i$ and $\theta(r_i^s, \ell_j^t) = I$. Therefore, every point ℓ_i^t with $1 \leq t \leq h_i$ is the target of a timing constraint. Moreover, every reset point r_i^s for $1 \leq s \leq g_i$ 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 $1 \leq s < u < t \leq g_i$ 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 request 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). To illustrate these ideas, we now provide a simple example of a timed automaton and an STCW that is generated by it.

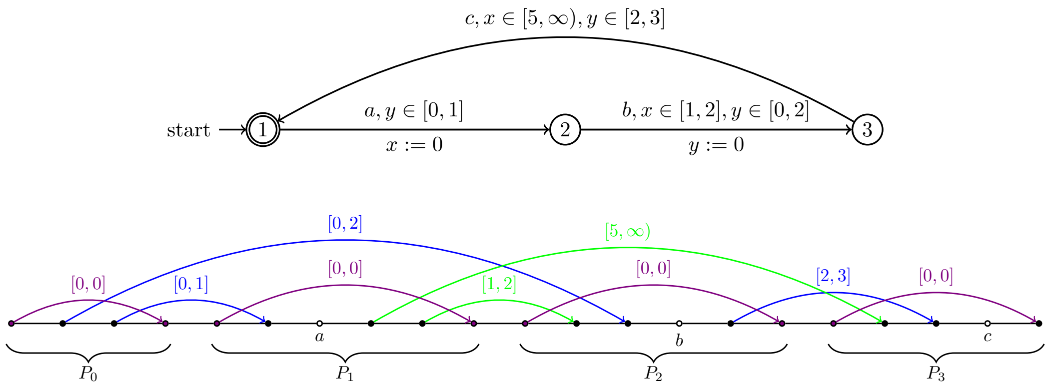


Figure 1 A timed automaton (top) with 2 clocks x, y. An STCW generated by an accepting run of the TA is depicted just below. The blue edges represent matching relations induced by clock y, while the green represent those induced by clock x. The violet edges are the $[0,0]$ timing constraints for the extra clock ζ ensuring that all points in some P_i representing transition i occur precisely at the same time. Black lines are process edges.

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

$$\tau ::= (i,a) \mid \mathsf{Add}_{i,j}^{\gamma}\, \tau \mid \mathsf{Forget}_i\, \tau \mid \mathsf{Rename}_{i,j}\, \tau \mid \tau \oplus \tau$$

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 some colors.

- $[\![(i,a)]\!]$ 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.

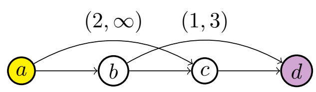
The *special tree-width* of a graph G is defined as the least K such that $G = G_\tau$ for some $(K+1)$-STT τ. See [8] for more details and its relation to tree-width. For TCWs, we have successor edges and $\triangleright$-edges carrying timing constraints, so we take $\Gamma = \{\rightarrow\} \cup \{(x,y) \mid x \in \mathbb{N}, y \in \overline{\mathbb{N}}\}$ 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:

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

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

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

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. The term τ is depicted as a tree in Figure 3 in Appendix A.



3.1 Split-TCWs and split-game

We find it convenient to prove that a simple TCW has bounded special tree-width by playing a split-game, whose game positions are simple TCWs in which some successor edges have been cut, i.e., are missing. Formally, a *split*-TCW is a structure $\mathcal{V} = (P, \rightarrow, \dashrightarrow, \lambda, \triangleright, \theta)$ where $\rightarrow$ and $\dashrightarrow$ are the present and absent successor edges (also called *holes*), respectively, such that $\rightarrow \cap \dashrightarrow = \emptyset$ and $(P, \rightarrow \cup \dashrightarrow, \lambda, \triangleright, \theta)$ is a TCW. A *block* or *factor* of a split-TCW is a maximal connected component of $(P, \rightarrow)$. 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-$\mathsf{TCW}$ is *atomic* if it consists of a single point ($|P| = 1$) or a single timing constraint with a hole ($P = \{p_1, p_2\}$, $p_1 \dashrightarrow p_2$, $p_1 \triangleright p_2$). 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. $\rightarrow \cup \triangleright$) split-$\mathsf{TCW}$s 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 $\mathsf{STCW}^K(\Sigma)$ (resp. $\mathsf{STCW}^{K,M}(\Sigma)$) denote the set of simple TCWs with split-width bounded by K (resp. and using constants at most M). The crucial link between special tree-width and split-width is given by the following lemma.

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

Starting from a STCW having constants $\leq M$, and a strategy of Eve of cost $\leq K$, we show in Appendix A how to build a (K, M)-STT.

3.2 Split-width for timed systems

Viewing these terms as trees, our goal in the next section will be to construct tree automata to recognize sets of (K, M)-STTs, and thus capture the (K split-width) bounded behaviors of a given system. A possible way to show that these capture *all* behaviors of the given system, is to show that we can find a K such that all the (graph) behaviors of the given system have a K-bounded split-width. We do this now for a TPDA and also mention how to modify the proof for a timed automaton. In Section 7, we also show how it extends to multi-pushdown systems.

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 identifying some properties satisfied by STCWs generated by a TPDA, and showing that all STCWs satisfying these properties have bounded split-width. Let $\mathcal{V} = (P, \rightarrow, \dashrightarrow, \lambda, \rhd, \theta)$ be a split STCW and let $\lessdot = \rightarrow \cup \dashrightarrow$. 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 $j < i'$ (see Figure 4 in Appendix A).

It is 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 (Appendix A, Claim 21). Then, 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 $\mathcal{V}'$. Notice that we have now a hole instead of position i. We call this a reset-hole for clock x. During the inductive process, we may have at most one such reset hole for each clock $x \in Y$, 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. If there is already a hole before the push event i, or if i is the first event of the split-TCW, then Eve detaches the atomic matching pair $i \rhd^s j$ by splitting after i and before j and the game continues from the remaining split-TCW. Otherwise, we cannot proceed as before, since this would create a push-hole and the pushes are not arranged in blocks (which results in holes widening, rather than increasing their number). Hence, removing push-pop edges as we removed timing constraints would create an unbounded number of holes. Instead, Eve divides the TCW in two parts, the left one contains the points before i from which she detaches the resets that will be checked in the right part, the right one contains all points between i and j together with the matching resets. This may create at most $|Y|$ reset holes in the left part and requires at most $2|Y| + 1$ cuts. The resulting split-TCW is not connected. Indeed, push-pop edges are well-nested (T_1) and since j is the last point of the split-TCW, there are no push-pop edges crossing position i: $i' \rhd^s j'$ and $i' < i$ implies $j' < i$. Hence, only clock constraints may cross position i. See Appendix A (page 21) for details and examples.

Using this strategy, we can show that after a move of Adam, the split-TCW has at most $2|Y| + 1$ blocks, and Eve will disconnect it using at most $2|Y|+1$ cuts. Therefore Eve wins without introducing more than $4|Y| + 2$ blocks. The details are in Appendix A (page 21). $\square$

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. Thus, we obtain a bound of $|Y| + 3$ on split-width, which proves Theorem 4 (1).

Figure 2 illustrates our strategy starting from the STCW depicted in Figure 1, generated by the TA shown in that figure.

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 (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. This is the hardest part of the proof due to timing constraints (over dense time).

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 $\mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{real}}) = \{\tau \in \mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{valid}}) \mid [\![\tau]\!] \text{ is realizable}\}$.

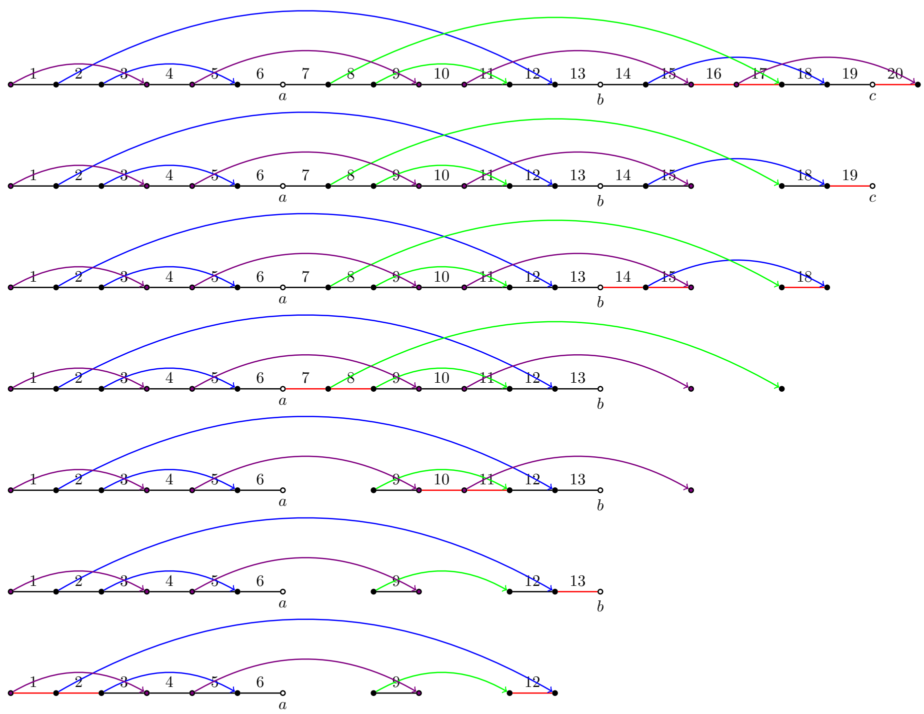


Figure 2 The first 4 steps of the split-game on the STCW of Figure 1. Note that we do not write the time intervals or transition names as they are irrelevant for this game. In the first step, Eve detaches the last timing constraint by cutting the 3 successor edges 16, 17 and 20. Adam chooses the non atomic STCW of the second line. In the second step, Eve detaches the last internal event labelled c by cutting edge 19. The resulting STCW is on the third line, from which Eve detaches the last timing constraint by cutting edges 14, 15, 18. She continues by cutting edges 7, 8 to detach the green timing constraint, and so on. Notice that we have three clocks $X = \{x, y, \zeta\}$ and that the maximal number of blocks $|X| + 3 = 6$ is reached on the last STCW when Eve cuts edges 1, 2 and 12.

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 $\mathcal{A}^{K,M}_{\mathcal{S}}$ of size $|\mathcal{S}|^{\mathcal{O}(K^2)} \cdot 2^{\mathcal{O}(K^2(|X|+1))}$ such that

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

Thus, the tree automaton $\mathcal{A}$ checking ***ValCoRe*** (i.e., validity, correctness and realizability) is $\mathcal{A} = \mathcal{A}^{K,M}_{\mathsf{real}} \cap \mathcal{A}^{K,M}_{\mathcal{S}}$. 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 7. 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. This is illustrated in the decomposition of the STCW in Figure 2, where Eve, at each step, breaks a split-STCW into two, an atomic and a non-atomic split-STCW (and subsequently Adam always picks the non-atomic STCW to proceed). 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 such that we can 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.

5.1 Shuffle and Realizability under contexts

Let $\mathcal{V}_1 = (P_1, \to_1, \dashrightarrow_1, \lambda_1, \rhd_1, \theta_1)$ and $\mathcal{V}_2 = (P_2, \to_2, \dashrightarrow_2, \lambda_2, \rhd_2, \theta_2)$ 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 $\to_1 \cup \to_2 \subseteq \lessdot$. Such orders are called *admissible*. Then, we define the split-TCW $\mathcal{V} = (P, \to, \dashrightarrow, \lambda, \rhd, \theta)$ by $P = P_1 \uplus P_2$, $\lambda = \lambda_1 \cup \lambda_2$, $\to = \to_1 \cup \to_2$, $\dashrightarrow = \lessdot \setminus \to$, $\rhd = \rhd_1 \cup \rhd_2$, 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 $\mathcal{V} = \mathcal{V}_1 ⧢_{\leq} \mathcal{V}_2$.

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 come up with a sound abstraction of finite index, so that a finite tree automaton can be defined which works only on the representatives. The operation ⧢ preserves the equivalence between split-TCW (Appendix C).

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, $U_1 ⧢_{\leq} U_2 \sim_M U_1' ⧢_{\leq} U_2'$ for all admissible orders $\leq$ on the blocks.*

5.2 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 (i) conditions under which it has finitely many states and (ii) additional abstractions to ensure that it is always finite.

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

Proof. The construction builds on the construction of $\mathcal{A}_{\mathsf{valid}}^{K,M}$, which is detailed in Appendix B. States of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ are pairs (q, wt) where $q = (P, <, \to)$ is a state of $\mathcal{A}_{\mathsf{valid}}^{K,M}$, i.e., $P \subseteq [2K]$, $<$ is a total order on P, $\to \subseteq \lessdot$ 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 $\mathsf{wt}(j,i) \leq 0 \leq \mathsf{wt}(i,j)$. 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 $[\![\tau]\!] = (V, \to, \lambda, \rhd, \theta, \chi)$. If a (bottom-up) run of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ reads τ and reaches state (q, wt) with $q = (P, <, \to)$, it induces a total order on blocks of $[\![\tau]\!]$ and turns it into a split-TCW $([\![\tau]\!], \dashrightarrow)$ (this is Property (A_3) of $\mathcal{A}_{\mathsf{valid}}^{K,M}$ proved formally in Appendix B). We say that the abstraction (q, wt) of τ computed by $\mathcal{A}_{\mathsf{inf}}^{K,M}$ is *sound* if it preserves realizability under contexts, i.e., $([\![\tau]\!], \dashrightarrow) \sim_M (q, \mathsf{wt})$. *The key invariant is that $\mathcal{A}_{\mathsf{inf}}^{K,M}$ 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 $a \in \Sigma$, $\mathcal{A}_{\mathsf{inf}}^{K,M}$ moves to state (q, wt) where $q = (\{1\}, \emptyset, \emptyset)$ and $\mathsf{wt}(1,1) = 0$. Similarly, when reading an atomic STT $\tau = \mathsf{Add}_{1,2}^{c,d}((1,a) \oplus (2,b))$, $\mathcal{A}_{\mathsf{inf}}^{K,M}$ 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 transitions $(q, \mathsf{wt}) \xrightarrow{\mathsf{Add}^{\rightarrow}_{i,j}} (q', \mathsf{wt})$, when q' is obtained from $q = (P, <, \rightarrow)$ by adding a successor edge between $(i,j) \in \lessdot \setminus \rightarrow$. 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 an edge only reduces the 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., $i \geq j$. 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 $(q, \mathsf{wt}) = (q_1, \mathsf{wt}_1) \sqcup\!\sqcup_{\leq} (q_2, \mathsf{wt}_2)$. Now, the induced ordering on the blocks of $[\![\tau]\!]$ corresponds to the same shuffle of blocks, i.e., $([\![\tau]\!], \dashrightarrow) = ([\![\tau_1]\!], \dashrightarrow_1) \sqcup\!\sqcup_{\leq} ([\![\tau_2]\!], \dashrightarrow_2)$. 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 and q' is obtained from $q = (P, <, \rightarrow)$ by removing the internal point i,
  - i is not part of a negative cycle of length 2: for all $j \neq i$ we have $\mathsf{wt}(j,i) + \mathsf{wt}(i,j) \geq 0$,
  - 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' (Claim 28 in Appendix C).
- 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., $\mathsf{wt}(i,j) + \mathsf{wt}(j,i) \geq 0$.

We can now check (see Appendix C.1) 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, in general, 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.

5.3 Bounding the constants

The finite tree automaton $\mathcal{A}_{\mathsf{real}}^{K,M}$ will work on a finite subset of the states of $\mathcal{A}_{\mathsf{inf}}^{K,M}$. More precisely, a state (q, wt) of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ with $q = (P, <, \rightarrow)$ is a state of $\mathcal{A}_{\mathsf{real}}^{K,M}$ 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 $\mathcal{A}_{\mathsf{inf}}^{K,M}$ 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). Thus, we have (Appendix C.2, page 31),

Lemma 14. *For all states $W = (q, \mathsf{wt})$ of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ 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 $\mathsf{wt}'(j, i) = \min\{\mathsf{wt}(j', i') \mid i \leq i' \leq j' \leq j\}$ and perform this after every $\oplus$ move of the tree automaton. It is easy to check that σ preserves realizability under contexts (Appendix C.2, page 30) and this allows us to show that the invariant (I2) is preserved (Lemma 36 in Appendix C.3). Now, under the BEP assumption, we can show that all back edges are bounded (Appendix C.2, page 32).

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 $\mathsf{wt}(i,j) + \sum_{e \in \mathsf{BBE}(i,j)} \mathsf{wt}(e) \geq (3K-1)M$. Note, $\mathsf{wt}(e) \leq 0$ 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, it turns out that showing that it is sound (i.e., it preserves realizability under all contexts) is rather tricky. With details in Appendix C.2 (page 32) 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 of the construction now follows as for $\mathcal{A}_{\mathsf{inf}}^{K,M}$. This completes the proof of Proposition 7.

6 Dense time multi-stack pushdown systems

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.

In this section, we consider dtMPDA with the restriction of "bounded rounds". To the best of our knowledge, this timed model has not been investigated until now. Our goal is to illustrate how our technique can easily be applied here with a minimal overhead (in difficulty and complexity).

Formally, a dtMPDA is a tuple $\mathcal{S} = (S, \Sigma, \Gamma, X, s_0, F, \Delta)$ similar to a TPDA defined in Section 2.2. The only difference is the stack operation op which now specifies which stack is being operated on. That is,

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

A sequence $\sigma = \mathsf{op}_1 \cdots \mathsf{op}_m$ of operations is a *round* if it can be decomposed in $\sigma = \sigma_1 \cdots \sigma_n$ where each factor σ_i is a possibly empty sequence of operations of the form $\mathsf{nop}, \downarrow_c^i, \uparrow_{c \in I}$.

Let us fixed an integer bound k on the number of rounds. The semantics of the dtMPDA in terms of STCWs is exactly the same as for TPDA, except that the sequence of stack operations along any run is restricted to (at most) k rounds. Thus, any run of dtMPDA can be broken into a finite number of contexts, such that in each context only a single stack is used. As before, the sequence of push-pop operations of any stack must be well-nested. We denote by $\mathsf{STCW}(\mathcal{S}, k)$ the set of simple TCWs generated by runs of $\mathcal{S}$ using at most k rounds. We let $\mathcal{L}(\mathcal{S}, k)$ be the corresponding language of realizable STCWs in $\mathsf{STCW}(\mathcal{S}, k)$.

We now lift the definition of well-timed STCWs to k-round well-timed STCWs and show that such STCWs have bounded split-width and thus all simple TCWs in $\mathsf{STCW}(\mathcal{S}, k)$ have bounded split-width (note that the realizable STCWs are a subset of this and hence will also have a bounded split-width). An STCW $\mathcal{V} = (P, \to, \lambda, \rhd, \theta)$ is k-round *well timed* with respect to a set of clocks Y and stacks $1 \le s \le n$ if it uses at most k-rounds and the $\rhd$ relation for timing constraints can be partitioned as $\rhd = \biguplus_{1 \le s \le n} \rhd^s \uplus \biguplus_{x \in Y} \rhd^x$ where for each $1 \le s \le n$, the relation $\rhd^s$ corresponds to the matching push-pop events of stack s as in (T_1), and for each $x \in Y$, the relation $\rhd^x$ corresponds to the timing constraints for clock x and is well-nested as in (T_2).

Lemma 18. *A k-round well-timed simple* TCW *has split-width at most $(4nk + 4)(|Y| + 1)$, where n is the number of stacks.*

Proof sketch. Again, we play the split-game between *Adam* and *Eve*. *Eve* should have a strategy to disconnect the word without introducing more than $(4nk + 4)(|Y| + 1)$ blocks. The strategy of *Eve* is as follows: Given the k-round word w, *Eve* first breaks this into two words. The first word only has stack 1 edges, and the second word has stack edges corresponding to stacks $2, \dots, n$. The first word can now be dealt with as we did in the case of TPDA. *Eve* then breaks the second word into two words, the first of which has only stack 2 edges, while the second word has edges of stacks 3,4 $\dots, n$, and so on. Finally, we obtain n split-STCW's, each having edges corresponding to only one stack. Once this is achieved, these words can be processed as was done in the case of TPDA. The only thing to calculate is the number of cuts required in isolating each word, whose details are in Appendix E. $\square$

Having established a bound on the split-width for dtMPDA restricted to k rounds, we now discuss the construction of a tree automaton that checks ***ValCoRe*** when the underlying system is a dtMPDA. As a first step, we keep track of the current round (context) number in the finite control. This makes sure that the tree automaton only accepts runs using at most k-rounds. The validity and realizability checks (Val and Re parts) are as discussed in Appendices B and C. The only change pertains to the automaton that checks correctness of the underlying run, namely, $\mathcal{A}_{\mathcal{S}}^{K,M}$, as we need to handle n stacks (and k rounds) instead of the single stack. As shown in Appendix E, this results in the blow up of the number of locations of $\mathcal{A}_{\mathcal{S}}^{K,M}$ by $n \cdot k$. Thus the size of the tree automaton $\mathcal{A}_{\mathcal{S}}^{K,M}$ that checks correctness when the underlying system is a k-round dtMPDA is $(nk|\mathcal{S}|)^{\mathcal{O}(K^2)} \cdot 2^{\mathcal{O}(nK^2(|X|+1))}$, where $K = (4nk + 4)(|X| + 2)$.

Combining this with Lemma 18, we can apply our technique (i.e., exactly the same reasoning as) explained in Section 4 to obtain our decidability result.

Theorem 19. *Checking emptiness for k-round* dtMPDA *is decidable in* EXPTIME.

7 Discussion and Future work

The main contribution of this paper is the technique for analyzing timed systems via tree automata. As a result, we have made a few simplifying assumptions to best illustrate this method. However, our technique can easily be adapted to remove these. We give two examples of this:

- 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. Of course this requires us to consider a few more cases, but it is not hard to see that our complexity bounds do not change.
- Diagonal constraints of the form $x - y \in I$ can be handled easily by adding matching edges. For a constraint $x - y \in [1, 5]$, we add an edge between the last reset of x and last reset of y with $[1, 5]$ interval. Thus, we can check the diagonal constraint at the time of last reset. Unlike in the

classical removal of diagonal guards, this does not induce any blowup in the number of states of the system.

Future work We can also extend our results to other restrictions for dtMPDA such as bounded scope and phase. Further, our techniques can be applied to the more general model [13] of recursive hybrid automata, where we can already see some (new) decidability results.

As future work, an interesting question 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. We also see a strong potential to investigate the emptiness problem for classes of alternating timed automata and hybrid automata.

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

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

5 G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 200–236, 2004.

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. 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. 2014.

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

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

Appendix

In this appendix, we give details of proofs as well as a few more illustrative examples and explanations. In order to make the appendix easier to read and somewhat self-contained, we have chosen to repeat some of the material given in the main paper. The appendix is organized section-wise.

A Details for Section 3

Special tree terms We start by giving the formal semantics of specital tree terms in more detail together with an example. To help reading this appendix, we also recall some definitions from Section 3.

The formal syntax of K-STTs over (Σ, Γ) is given by

$$\tau ::= (i,a) \mid \mathsf{Add}_{i,j}^{\gamma}\,\tau \mid \mathsf{Forget}_i\,\tau \mid \mathsf{Rename}_{i,j}\,\tau \mid \tau \oplus \tau$$

where $a \in \Sigma$, $\gamma \in \Gamma$ and $i,j \in [K] = \{1,\ldots,K\}$ are colors.

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.

Each K-STT represents 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 some colors.

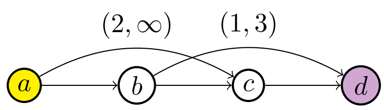
- $[\![(i,a)]\!]$ consists of a single a-labeled vertex with color i.
- $\mathsf{Add}_{i,j}^{\gamma}$ adds a γ-labeled edge to the vertices colored i and j (if such vertices exist). Formally, if $[\![\tau]\!] = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda, \chi)$ then $[\![\mathsf{Add}_{i,j}^{\alpha}\,\tau]\!] = (V, (E'_\gamma)_{\gamma\in\Gamma}, \lambda, \chi)$ with $E'_\gamma = E_\gamma$ if $\gamma \neq \alpha$ and $E'_\alpha = \begin{cases} E_\alpha & \text{if } \{i,j\} \not\subseteq \mathsf{dom}(\chi) \\ E_\alpha \cup \{(\chi(i),\chi(j))\} & \text{otherwise.}\end{cases}$
- Forget_i removes color i from the domain of the color map. Formally, if $[\![\tau]\!] = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda, \chi)$ then $[\![\mathsf{Forget}_i\,\tau]\!] = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda, \chi')$ with $\mathsf{dom}(\chi') = \mathsf{dom}(\chi) \setminus \{i\}$ and $\chi'(j) = \chi(j)$ for all $j \in \mathsf{dom}(\chi')$.
- $\mathsf{Rename}_{i,j}$ exchanges the colors i and j. Formally, if $[\![\tau]\!] = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda, \chi)$ then $[\![\mathsf{Rename}_{i,j}\,\tau]\!] = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda, \chi')$ with $\chi'(\ell) = \chi(\ell)$ if $\ell \in \mathsf{dom}(\chi) \setminus \{i,j\}$, $\chi'(i) = \chi(j)$ if $j \in \mathsf{dom}(\chi)$ and $\chi'(j) = \chi(i)$ if $i \in \mathsf{dom}(\chi)$.
- Finally, $\oplus$ constructs the disjoint union of the two graphs provided they use different colors. This operation is undefined otherwise. Formally, if $[\![\tau_i]\!] = (G_i, \chi_i)$ for $i = 1,2$ and $\mathsf{dom}(\chi_1) \cap \mathsf{dom}(\chi_2) = \emptyset$ then $[\![\tau_1 \oplus \tau_2]\!] = (G_1 \uplus G_2, \chi_1 \uplus \chi_2)$. Otherwise, $\tau_1 \oplus \tau_2$ is not a valid STT.

The special tree-width of a graph G is the least K such that $G = G_\tau$ for some $(K+1)$-STT τ.

For TCWs, we have successor edges and $\triangleright$-edges carrying timing constraints, so we take $\Gamma = \{\to\} \cup \{(x,y) \mid x \in \mathbb{N}, y \in \overline{\mathbb{N}}\}$ with $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$. For instance, consider the 4-STT

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

The term τ is depicted as a binary tree on the left of Figure 3 and its semantics $[\![\tau]\!]$ is the following simple TCW where only the endpoints labelled a and d are colored.



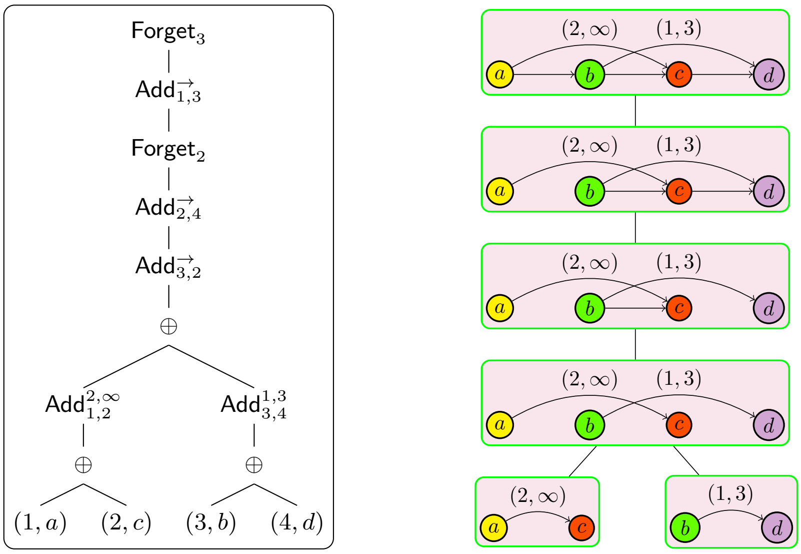


Figure 3 An STT and a simple TCW with its split-tree

Split-game and split-tree Split-TCWs and the split-game were defined in Section 3. A strategy for Eve from a split-TCW $\mathcal{V}$ can be described with a *split-tree* T which is a binary tree labeled with split-TCWs satisfying:

1. The root is labeled by $\mathcal{V} = \mathsf{labroot}(T)$.
2. Leaves are labeled by atomic split-TCWs.
3. Eve's move: Each unary node is labeled with some connected (wrt. $\to \cup \rhd$) split-TCW $\mathcal{V}$ and its child is labeled with some $\mathcal{V}'$ obtained by splitting a factor of $\mathcal{V}$ in two, i.e., by removing one successor edge. Thus, $\mathsf{width}(\mathcal{V}') = 1 + \mathsf{width}(\mathcal{V})$.
4. Adam's move: Each binary node is labeled with some non connected split-TCW $\mathcal{V} = \mathcal{V}_1 \uplus \mathcal{V}_2$ where $\mathcal{V}_1$ and $\mathcal{V}_2$ are the labels of its children. Note that $\mathsf{width}(\mathcal{V}) = \mathsf{width}(\mathcal{V}_1) + \mathsf{width}(\mathcal{V}_2)$.

The *width* of a split-tree T, denoted $\mathsf{width}(T)$, is the maximum width of the split-TCWs labeling the nodes of T. In other words, the cost of the strategy encoded by T is $\mathsf{width}(T)$. A K-split-tree is a split-tree of width at most K.

An example of a split-tree is given in Figure 3 (right). Observe that the width of the split-tree is 4. Hence the split-width of the simple TCW labeling the root is at most four.

Let $\mathcal{V} = (P, \to, \dashrightarrow, \lambda, \rhd, \theta)$ be a split-TCW. Recall that we denote by $\mathsf{EP}(\mathcal{V})$ the subset of events that are endpoints of blocks of $\mathcal{V}$. A left endpoint is an event $e \in V$ such that there are no f with $f \to e$. We define similarly right endpoints. Note that an event may be both a left and right endpoint. The number of endpoints is at most twice the number of factors: $|\mathsf{EP}(\mathcal{V})| \leq 2 \cdot \mathsf{width}(\mathcal{V})$.

Lemma 20. *A (split) simple* TCW *of split-width at most K has special tree-width at most $2K - 1$.*

Proof. We associate with every split-tree T of width at most K a $2K$-STT $\overline{T}$ such that $[\![\overline{T}]\!] = (\mathcal{V}, \chi)$ where $\mathcal{V} = \mathsf{labroot}(T)$ is the label of the root of T and the range of χ is the set of endpoints of $\mathcal{V}$: $\mathsf{Im}(\chi) = \mathsf{EP}(\mathcal{V})$. Notice that $\mathsf{dom}(\chi) \subseteq [2K]$ since $\overline{T}$ is a $2K$-STT. The construction is by induction on T.

Assume that $\mathsf{labroot}(T)$ is atomic. Then it is either an internal event labeled $a \in \Sigma$, and we let $\overline{T} = (1, a)$. Or, it is a pair of events $e \rhd f$ with a timing constraint $\theta(e, f) = [c, d]$ and we let $\overline{T} = \mathsf{Add}^{c,d}_{1,2}((1, \lambda(e)) \oplus (2, \lambda(f)))$.

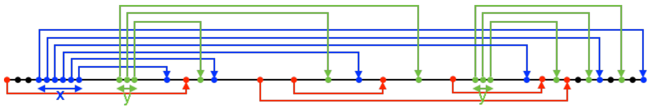


Figure 4 Well-timed: relations $\rhd^s$ in red, $\rhd^x$ in blue and $\rhd^y$ in green.

If the root of T is a binary node and the left and right subtrees are T_1 and T_2 then $\mathsf{labroot}(T) = \mathsf{labroot}(T_1) \uplus \mathsf{labroot}(T_2)$. By induction, for $i = 1, 2$ the STT $\overline{T_i}$ is already defined and we have $[\![\overline{T_i}]\!] = (\mathsf{labroot}(T_i), \chi_i)$. We first rename colors that are active in both STTs. To this end, we choose an injective map $f \colon \mathsf{dom}(\chi_1) \cap \mathsf{dom}(\chi_2) \to [2K] \setminus (\mathsf{dom}(\chi_1) \cup \mathsf{dom}(\chi_2))$. This is possible since $|\mathsf{dom}(\chi_i)| = |\mathsf{Im}(\chi_i)| = |\mathsf{EP}(\mathsf{labroot}(T_i))|$. Hence, $|\mathsf{dom}(\chi_1)| + |\mathsf{dom}(\chi_2)| = |\mathsf{EP}(\mathsf{labroot}(T))| \leq 2K$.

Assuming that $\mathsf{dom}(f) = \{i_1, \ldots, i_m\}$, we define

$$\overline{T} = \overline{T_1} \oplus \mathsf{Rename}_{i_1, f(i_1)} \cdots \mathsf{Rename}_{i_m, f(i_m)} \overline{T_2} \,.$$

Finally, assume that the root of T is a unary node with subtree T'. Then, $\mathsf{labroot}(T')$ is obtained from $\mathsf{labroot}(T)$ by splitting one factor, i.e., removing one word edge, say $e \to f$. We deduce that e and f are endpoints of $\mathsf{labroot}(T')$, respectively right and left endpoints. By induction, the STT $\overline{T'}$ is already defined. We have $[\![\overline{T'}]\!] = (\mathsf{labroot}(T'), \chi')$ and $e, f \in \mathsf{Im}(\chi')$. So let i, j be such that $\chi'(i) = e$ and $\chi'(j) = f$. We add the process edge with $\tau = \mathsf{Add}^{\rightarrow}_{i,j}\, \overline{T'}$. Then we forget color i if e is no more an endpoint, and we forget j if f is no more an endpoint:

$$\tau' = \begin{cases} \tau & \text{if } e \text{ is still an endpoint,} \\ \mathsf{Forget}_i\, \tau & \text{otherwise} \end{cases} \qquad \overline{T} = \begin{cases} \tau' & \text{if } f \text{ is still an endpoint,} \\ \mathsf{Forget}_j\, \tau' & \text{otherwise.} \end{cases}$$

◻

Bound on split-width for TPDA. We complete this section by providing a detailed proof of Theorem 4. We provide more intuition and pictures to elucidate the ideas presented succinctly in Section 3. We show that for a TPDA $\mathcal{S}$, all words in $\mathsf{STCW}(\mathcal{S})$ have bounded split-width. As mentioned, we identify some properties satisfied by all simple TCWs generated by a TPDA, and then we show that all simple TCWs satisfying these properties have bounded split-width.

Let $\mathcal{V} = (P, \to, \dashrightarrow, \lambda, \rhd, \theta)$ be a simple-TCW. We say that $\mathcal{V}$ is *well timed* with respect to 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) An x-reset block is a maximal consecutive sequence $i_1 \lessdot \cdots \lessdot i_n$ of positions in the domain of the relation $\rhd^x$. 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 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 $j < i'$, (see Figure 4).

Claim 21. *Simple* TCWs *defined by a TPDA with set of clocks X are well-timed wrt. set of clock $Y = X \cup \{\zeta\}$, i.e., satisfy properties (T1) and (T2).*

Proof. The first condition (T_1) is satisfied by $\mathsf{STCW}(\mathcal{S})$ by definition. For (T2), let $i \rhd^x j$ and $i' \rhd^x j'$ for some clock $x \in X$. If i, i' are points in the same x-reset block for some $x \in X$, then by construction of $\mathsf{STCW}(\mathcal{S})$, if $i < i'$ then $i' < j' < j$ which gives well nesting. Similarly, if $i < i'$ are

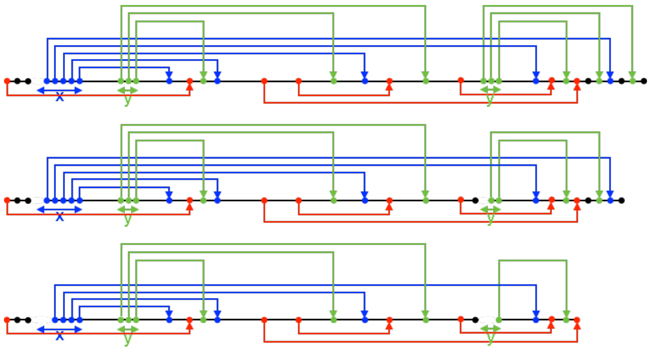


Figure 5 Removing timing constraints. At most one reset hole per clock. Edges below are stack edges. Clock edges labeled with x, y.

points in different x-reset blocks, then by definition of $\mathsf{STCW}(\mathcal{S})$, we have $j < i'$. Also, it is clear that the new clock ζ satisfies (T_2). $\square$

Then, the following lemma completes the proof of Theorem 4 (2).

Lemma 22. *The split-width of a well-timed simple* TCW *is bounded by* $4|Y| + 2$.

This lemma is proved by playing the split-width game between *Adam* and *Eve*. *Eve* should have a strategy to disconnect the word without introducing more than $4|Y| + 2$ blocks. The strategy of *Eve* uses three operations processesing the word from right to left.

Removing an internal point. If the last/right-most event on the word (say event j) is not the target of a $\triangleright$ relation, then she will split the $\rightarrow$-edge before the last point, i.e., the edge between point j and its predecessor.

Removing a clock constraint Assume that we have a timing constraint $i \triangleright^x j$ where j is the last point of the split-TCW. Then, by (T_2) we deduce that i is *the first* point of the *the last reset block* for clock x. *Eve* splits three word-edges to detach the matching pair $i \triangleright^x j$: these three edges are those connected to i and j. Since the matching pair $i \triangleright^x j$ is atomic, *Adam* should continue the game from the remaining split-TCW $\mathcal{V}'$. Notice that we have now a hole instead of position i. We call this a reset-hole for clock x. For instance, starting from the split-TCW of Figure 4, and removing the last timing constraint of clock x, we get the split-TCW on top of Figure 5. Notice the reset hole at the beginning of the x reset block.

During the inductive process, we may have at most one such reset hole for each clock $x \in Y$. Note that the first time we remove the last point which is a timing constraint for clock x, we create a hole in the last reset block of x which contains a sequence of reset points for x, by removing two edges. This hole is created by removing the leftmost point in the reset block. As we keep removing points from the right which are timing constraints for x, this hole widens in the reset block, by removing each time, just one edge in the reset block. Continuing the example, if we detach the last internal point and then the timing constraint of clock y we get the split-TCW in the middle of Figure 5. Now, we have one reset hole for clock x and one reset hole for clock y.

We continue by removing from the right, one internal point, one timing constraint for clock x, one timing constraint for clock y, and another internal point, we get the split-TCW at the bottom of Figure 5. Notice that we still have a *single* reset hole for each clock x and y.

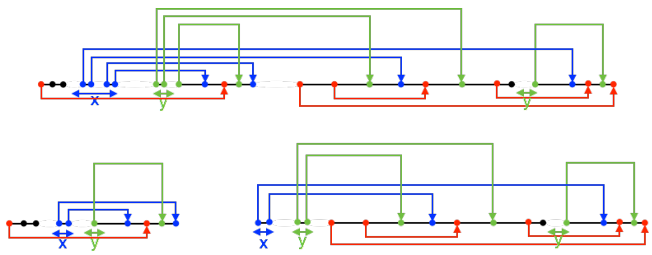


Figure 6 Splitting the TCW at a position with no crossing stack edge.

Removing a push-pop edge Assume now that the last event is a pop: we have $i \rhd^s j$ where j is the last point of the split-TCW. If there is already a hole before the push event i or if i is the first point of the split-TCW, then we split after i and before j to detach the atomic matching pair $i \rhd^s j$. Adam should choose the remaining split-TCW and the game continues.

Note that if i is not the first point of the split-TCW and there is no hole before i, we cannot proceed as we did in the case of clock constraints, since this would create a push-hole and the pushes are not arranged in blocks as the resets. Hence, removing push-pop edges as we removed timing constraints would create an unbounded number of holes. Instead, we split the TCW just before the matching push event i. Since push-pop edges are well-nested (T_1) and since j is the last point of the split-TCW, there are no push-pop edges crossing position i: $i' \rhd^s j'$ and $i' < i$ implies $j' < i$. Hence, only clock constraints may cross position i.

Consider some clock x having timing constraints crossing position i. All these timing constraints come from the last reset block B_x of clock x which is before position i. Moreover, these resets form the left part of the reset block B_x. We detach this left part with two splits, one before the reset block B_x and one after the last reset of block B_x whose timing constraint crosses position i. We proceed similarly for each clock of Y. Recall that we also split the TCW just before the push event i. As a result, the TCW is not connected anymore. Notice that we have used at most $2|Y| + 1$ new splits to disconnect the TCW. For instance, from the bottom split-TCW of Figure 5, applying the procedure above, we obtain the split-TCW on top of Figure 6 which has two connected components. Notice that to detach the left part of the reset block of clock x we only used one split since there was already a reset hole at the beginning of this block. The two connected components are depicted separately at the bottom of Figure 6.

Invariant. The split-TCW at the bottom right of Figure 6 is representative of the split-TCW which may occur during the split-game using the strategy of *Eve* described above. These split-TCW satisfy the following invariant.

(I_1) The split-TCW starts with reset blocks, at most one for each clock in Y. For instance, the split-TCW of Figure 7 starts with two *reset blocks*, one for clock x and one for clock y (see the two hanging reset blocks on the left, one of x and one of y).

(I_2) After these reset blocks, the split-TCW may have reset holes, at most one for each clock in Y. For instance, the split-TCW of Figure 7 has two *reset holes*, one for clock z and one for clock y. A reset hole for clock x is followed by the last reset block of clock x, if any. Hence, for all timing constraints $i \rhd^x j$ such that j is on the right of the hole, the reset event i is in the reset block that starts just after the hole.

Lemma 23. *A split-TCW satisfying (I_1–I_2) has at most $2|Y| + 1$ blocks. It is disconnected by Eve's strategy using at most $2|Y| + 1$ new splits, and the resulting connected components satisfy (I_1–I_2). Therefore, its split-width is at most $4|Y| + 2$.*

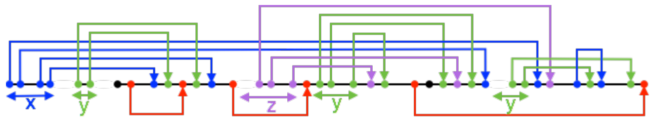


Figure 7 A split-TCW satisfying (T$_1$–T$_2$) and (I$_1$–I$_2$).

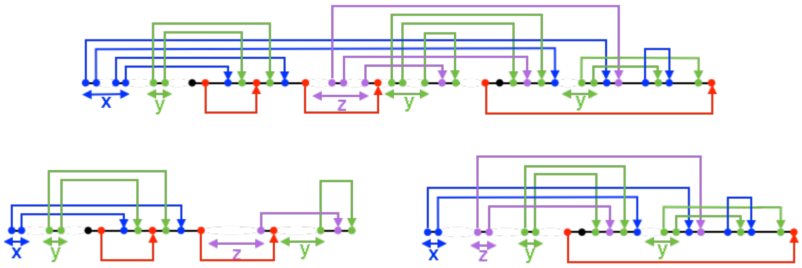


Figure 8 Splitting the TCW at a position with no crossing stack edge.

Proof. Let us check that starting from a split-TCW satisfying (I$_1$–I$_2$), we can apply *Eve's* strategy and the resulting connected components also satisfy (I$_1$–I$_2$). This is trivial when the last point is internal in which case *Eve* makes one split before this last point.

In the second case, the last event checks a timing constraint for some clock x: we have $i \rhd^x j$ and j is the last event. If there is already a reset hole for clock x in the split-TCW, then by (I$_2$) and (T$_2$), the reset event i must be just after the reset hole for clock x. So with at most two new splits *Eve* detaches the atomic edge $i \rhd^x j$ and the resulting split-TCW satisfies the invariants. If there is no reset hole for clock x then we consider the last reset block B_x for clock x. By (T$_2$), i must be the first event of this block. Either B_x is one of the first reset blocks of the split-TCW (I$_1$) and *Eve* detaches with at most two splits the atomic edge $i \rhd^x j$. Or *Eve* detaches this atomic edge with at most three splits, creating a reset hole for clock x in the resulting split-TCW. In both cases, the resulting split-TCW satisfies the invariants.

The third case is when the last event is a pop event: $i \rhd^s j$ and j is the last event. Then there are two subcases: either, there is a hole before event i then *Eve* detaches with two splits the atomic edge $i \rhd^x j$ and the resulting split-TCW satisfies the invariants. Or we cut the split-TCW before position i. Notice that no push-pop edges cross i: if $i' \rhd^s j'$ and $i' < i$ then $j' < i$. As above, for each clock x having timing constraints crossing position i, we consider the last reset block B_x for clock x which is before position i. The resets of the timing constraints for clock x crossing position i form a left factor of the reset block B_x. We detach this left factor with at most two splits. We proceed similarly for each clock of Y. The resulting split-TCW is not connected anymore and we have used at most $2|Y| + 1$ more splits. For instance, if we split the TCW of Figure 7 just before the last push following the procedure described above, we get the split-TCW on top of Figure 8. This split-TCW is not connected and its left and right connected components are drawn below.

To see that the invariants are maintained by the connected components, let us inspect the splitting of block B_x. First, B_x could be one of the beginning reset blocks (I$_1$). This is the case for clock x in Figures 7 and 8. In which case *Eve* use only one split to divide B_x in B_x^1 and B_x^2. The left factor B_x^1 corresponds to the edges crossing position i and will form one of the reset block (I$_1$) of the right connected component. On the other hand, the suffix B_x^2 stays a reset block of the left connected component. Second, B_x could follow a reset hole for clock x. This is the case for clock z in Figures 7 and 8. In which case again *Eve* only needs one split to detach the left factor B_x^1 which becomes a reset block (I$_1$) of the right component. The reset hole before B_x stays in the left component. Finally, assume that B_x is neither a beginning reset block (I$_1$), nor follows a reset hole for clock x. This is the case for clock y in Figures 7 and 8. Then *Eve* detaches the left factor B_x^1 which becomes a reset block (I_1) of the right component and creates a reset hole in the left component. ◻

B Tree automata for Validity

Not all graphs defined by (K, M)-STTs are TCWs of split-width at most K. Indeed, if τ is such an STT, the edge relation $\to$ may have cycles or may be branching, which is not possible in a TCW. Also, the timing constraints given by $\triangleright$ need not comply with the $\to$ relation: for instance, we may have a timing constraint $e \triangleright f$ with $f \to^+ e$. Moreover, some subterm may define graphs having more than K blocks. So we use $\mathcal{A}_{\mathsf{valid}}^{K,M}$ to check for validity.

Proposition 24. We can build a tree automaton $\mathcal{A}_{\mathsf{valid}}^{K,M}$ of size $\mathcal{O}(M) \cdot 2^{\mathcal{O}(K^2)}$ such that

(V_1) $\mathcal{A}_{\mathsf{valid}}^{K,M}$ accepts *only* (K, M)-STTs τ such that $[\![\tau]\!]$ is a simpleTCW of split-width at most K.
(V_2) $\mathcal{A}_{\mathsf{valid}}^{K,M}$ accepts *all* (K, M)-STTs $\overline{T}$ arising from split-trees T of width at most K and using constants less than M.

In particular, we have $\mathsf{STCW}^{K,M} = \{[\![\tau]\!] \mid \tau \in \mathcal{L}(\mathcal{A}_{\mathsf{valid}}^{K,M})\}$.

A (K, M)-STT τ defines a graph $[\![\tau]\!] = (V, \to, (\triangleright^{x,y})_{y\in\{0,\ldots,M,\infty\}}^{x\in\{0,\ldots,M\}}, \lambda)$. The graph $[\![\tau]\!]$ is a TCW if it satisfies several MSO-definable conditions. First, $\to$ should be the successor relation of a total order on V. Second, each timing constraint relation should be compatible with the total order: $\triangleright^{x,y} \subseteq \to^+$. Also, a pair of points may have at most one timing constraint, hence the matching relations should be disjoint: $\triangleright^{x,y} \cap \triangleright^{x',y'} = \emptyset$ if $(x, y) \neq (x', y')$. If these conditions are satisfied, with $\triangleright = \bigcup_{x,y} \triangleright^{x,y}$ and $\theta(i, j) = (x, y)$ for $(i, j) \in \triangleright^{x,y}$ we see that $[\![\tau]\!]$ is a TCW. Further, we should check that this TCW is simple. Since being a simple TCW is a graph property which is MSO definable and since a graph $[\![\tau]\!]$ has an MSO-interpretation in the tree τ, we deduce that there is a tree automaton that accepts *all* (K, M)-STTs τ such that $[\![\tau]\!]$ is a simple TCW (cf. [9]).

But this is not exactly what we want/need. So we provide in the proof below a direct construction of another tree automaton. There are several reasons for directly constructing the tree automaton $\mathcal{A}_{\mathsf{valid}}^{K,M}$. First, this allows to have a clear upper-bound on the size of $\mathcal{A}_{\mathsf{valid}}^{K,M}$ (though maybe not impossible, this would be technical via MSO). Second, simplicity of the TCW and the bound on split-width can be enforced with no additional cost. Third and most importantly, $\mathcal{A}_{\mathsf{valid}}^{K,M}$ does not accept *all* (K, M)-STTs denoting TCWs. Instead, it accepts the STTs $\overline{T}$ arising from split-trees T. We will take advantage of the special form of these STTs when constructing the tree automaton $\mathcal{A}_{\mathsf{real}}^{K,M}$ checking the realizability of simple-TCWs.

Proof. A state of $\mathcal{A}_{\mathsf{valid}}^{K,M}$ will be an abstraction of the graph defined by the STT read so far. The finite abstraction will keep only the colored points of the graph. The abstraction is enriched with a guessed total order $<$ on the colored points. This guessed total order will be compatible both with the $\triangleright$-edges corresponding to the timing constraints defined at atomic STTs and with the $\to$-edges added later. When adding a new $\to$-edge, we will check that it is compatible with the guessed total order $<$. This will ensure that the graph defined by the STT is in fact a split-TCW.

Formally, *states of* $\mathcal{A}_{\mathsf{valid}}^{K,M}$ are tuples of the form $q = (P, <, \to)$ where $P \subseteq [2K]$ is the set of points, $<$ is a total order on P, $\to \subseteq \lessdot$ is the successor relation between points that are in the same block, and q has at most K blocks, i.e., $(P, \to)$ has at most K connected components.

The automaton $\mathcal{A}_{\mathsf{valid}}^{K,M}$ is non-deterministic since at $\oplus$-nodes it has to guess how the blocks will eventually be ordered in the TCW denoted by the full STT. To help understanding the construction, we first define properties that will be maintained by the automaton.

Let τ be a (K, M)-STT with $[\![\tau]\!] = (V, \to, \lambda, \triangleright, \theta, \chi)$. Let $q = (P, <, \to)$ be a state of $\mathcal{A}_{\mathsf{valid}}^{K,M}$. Notice some overloading of notations for $\to$ but since we assume that P and V are disjoint this is not a problem. We say that q is an *abstraction* of τ if it satisfies the following properties

(A_1) $\mathsf{dom}(\chi) = P$ and $\mathsf{EP}([\![\tau]\!]) \subseteq \chi(P)$ i.e., the points of q are in bijection with the colored points of $[\![\tau]\!]$ and the endpoints of $[\![\tau]\!]$ are colored,
(A_2) for all $i, j \in P$, we have $i \to^+ j$ in q iff $\chi(i) \to^+ \chi(j)$ in $[\![\tau]\!]$,
(A_3) Let $\dashrightarrow\, = \chi(\lessdot \setminus \to)$ be the relation on V defined by $\{(\chi(i), \chi(j)) \mid (i, j) \in \lessdot \setminus \to\}$. Then, $([\![\tau]\!], \dashrightarrow)$ is a split-STCW, i.e., $\to \cup \dashrightarrow$ is the *direct successor* relation of a total order $(\to \cup \dashrightarrow)^+$ on V which is compatible with the timing constraints $\rhd \subseteq (\to \cup \dashrightarrow)^+$.

Claim 25. *The blocks (resp. left/right endpoints) of q are in 1-to-1 correspondance with blocks (resp. left/right endpoints) of $[\![\tau]\!]$.*

Proof. Let $i \in P$ be a right endpoint, i.e., i has no outgoing $\to$-edge. Assume that $\chi(i)$ is not a right endpoint of $[\![\tau]\!]$. Since $(V, \to)$ is acyclic by (A_3), following from $\chi(i)$ a $\to$-path, we must reach a $\to$-maximal point $f \in V$, i.e., a right endpoint. Since endpoints are colored, we find $j \in P$ such that $f = \chi(j)$. Now, we have $\chi(i) \to^+ \chi(j)$ and we deduce that $i \to^+ j$, a contradiction. Conversely, a right endpoint e of $(V, \to)$ is colored hence we find $i \in P$ with $\chi(i) = e$. Now, if i is not a right endpoint then $i \to j$ for some $j \in P$. We deduce that $\chi(i) \to^+ \chi(j)$, a contradiction. Therefore, χ is a bijection between right endpoints of q and right endpoints of $[\![\tau]\!]$. Similarly, χ is a bijection between left endpoints of q and left endpoints of $[\![\tau]\!]$. Now, we deduce that (i, j) is a block in q (i.e., i is a left endpoint, $i \to^* j$ and j is a right endpoint) iff $(\chi(i), \chi(j))$ is a block of $[\![\tau]\!]$. ☐

Now we can define the *transitions of $\mathcal{A}^{K,M}_{\mathsf{valid}}$* so that if $\mathcal{A}^{K,M}_{\mathsf{valid}}$ admits a run on τ reaching state q at the root of τ, then q is an abstraction of τ.

- Atomic STTs: On an atomic STT $\tau = \mathsf{Add}^{c,d}_{1,2}((1, a) \oplus (2, b))$ the automaton $\mathcal{A}^{K,M}_{\mathsf{valid}}$ moves to state $q_{1,2} = (\{1, 2\}, 1 < 2, \emptyset)$ (we do not describe the intermediary transitions). When reading an atomic STT $\tau = (1, a)$ the automaton $\mathcal{A}^{K,M}_{\mathsf{valid}}$ moves to state $q_1 = (\{1\}, \emptyset, \emptyset)$. In both cases, the state reached by $\mathcal{A}^{K,M}_{\mathsf{valid}}$ on an atomic STT τ is indeed an abstraction of τ.
- $\mathsf{Rename}_{i,j}$: We define transitions $q \xrightarrow{\mathsf{Rename}_{i,j}} q'$ when q' is obtained from q by exchanging i and j. If $\tau' = \mathsf{Rename}_{i,j}\, \tau$ and q is an abstraction of τ then q' is an abstraction of τ'.
- Forget_i: We define transitions $q = (P, <, \to) \xrightarrow{\mathsf{Forget}_i} q' = (P', <', \to')$ when the following hold
  - $i \in P$ is not an endpoint (we do not allow forgetting the color of an endpoint), i.e., we find $i', i'' \in P$ with $i' \to i \to i''$,
  - $P' = P \setminus \{i\}$, $<'$ is the restriction of $<$ to P', $\to' = \to \setminus \{(i', i), (i, i'')\} \cup \{(i', i'')\}$.

  If $\tau' = \mathsf{Forget}_i\, \tau$ and q is an abstraction of τ then we can check that q' is an abstraction of τ'. In particular, condition (A_3) follows from $\lessdot' \setminus \to' = \lessdot \setminus \to$. Note that if the first condition above is not satisfied then the tree automaton $\mathcal{A}^{K,M}_{\mathsf{valid}}$ has no transitions from q reading Forget_i.
- $\mathsf{Add}^{\to}_{i,j}$: We define transitions $q = (P, <, \to) \xrightarrow{\mathsf{Add}^{\to}_{i,j}} q' = (P', <', \to')$ when the following hold
  - q has no internal points, i.e., there are no $k', k, k'' \in P$ with $k' \to k \to k''$ (internal points should be abstracted away with Forget_k before adding a successor edge),
  - $(i, j) \in \lessdot \setminus \to$: successor edges are only added between $<$-consecutive points,
  - $P' = P$, $<' = <$ and $\to' = \to \cup \{(i, j)\}$.

  If $\tau' = \mathsf{Add}^{\to}_{i,j}\, \tau$ and q is an abstraction of τ then q' is an abstraction of τ'. Indeed, conditions (A_1—A_2) for q' are easy to check. Condition (A_3) follows from the fact that the relation $\to' \cup \chi(\lessdot' \setminus \to')$ in $[\![\tau']\!]$ equals the relation $\to \cup \chi(\lessdot \setminus \to)$ in $[\![\tau]\!]$ since the pair $(\chi(i), \chi(j))$ is added to $\to'$ and removed from $\chi(\lessdot' \setminus \to')$. Again, if the first two conditions above are not satisfied then the tree automaton $\mathcal{A}^{K,M}_{\mathsf{valid}}$ has no transitions from q reading $\mathsf{Add}^{\to}_{i,j}$.
- $\oplus$: We define transitions $q_1, q_2 \xrightarrow{\oplus} q$ when the following hold (with $q_1 = (P_1, <_1, \to_1)$ and $q_2 = (P_2, <_2, \to_2)$)

- there are no internal points in q_1 and q_2 (internal points should be forgotten first),
- $P_1 \cap P_2 = \emptyset$: the $\oplus$ operation on STTs requires that the "active" colors of the two arguments are disjoint,
- $q = (P, <, \rightarrow)$ where $P = P_1 \cup P_2$, $\rightarrow\, = \,\rightarrow_1 \cup \rightarrow_2$, the total order $<$ is obtained by *guessing* how the blocks of q_1 and q_2 are shuffled: it could be any total order satisfying both $\rightarrow\, \subseteq\, \lessdot$ and $<_1 \cup <_2\, \subseteq\, <$.

If $\tau = \tau_1 \oplus \tau_2$ and q_1, q_2 are abstractions of τ_1, τ_2 then q is an abstraction of τ. Again, conditions (A_1—A_2) for q are easy to check. Now the relation $\lessdot \setminus \rightarrow$ on q defines how the blocks of q_1 and q_2 are shuffled in q. Similarly, the relation $\dashrightarrow\, = \chi(\lessdot \setminus \rightarrow)$ defines how the blocks of $[\![\tau_1]\!]$ and $[\![\tau_2]\!]$ are shuffled in $[\![\tau]\!]$. Notice that if $(i,j) \in\, \lessdot \setminus \rightarrow$ then i is a right endpoint and j is a left endpoint. Hence, the relation $\dashrightarrow$ connects right enpoints of $[\![\tau]\!]$ to left endpoints of $[\![\tau]\!]$, following the order dictated by q. We deduce that (A_3) holds for q.
Again if the first two conditions above are not satisfied then the tree automaton $\mathcal{A}^{K,M}_{\mathsf{valid}}$ has no transitions from q_1, q_2 reading $\oplus$.

Claim 26. *The number of* internal points *in each reachable state* (q, wt) *of* $\mathcal{A}^{K,M}_{\mathsf{valid}}$ *is at most 2.*

Proof. To see this, notice that when we begin, the states reached after reading atomic STTs have no internal points. A $\oplus$ move can be taken only if there are no internal points. Similarly, an $\mathsf{Add}^{\rightarrow}_{i,j}$ move can be taken only if there are no internal points, but after this move, we may end up introducing either one or at most two internal points. Finally, a $\mathsf{Rename}_{i,j}$ move preserves the number of internal points and a Forget_i move can only decrease the number of internal points. Hence overall, at any reachable state, we can have only a maximum of two internal points. ◻

Accepting condition The accepting states of $\mathcal{A}^{K,M}_{\mathsf{valid}}$ should correspond to abstractions of TCWs. Hence the accepting states are of the form $(\{i\}, \emptyset, \emptyset)$ or $(\{i,j\}, i<j, i \rightarrow j)$ for $i,j \in [2K]$. With this, we are able to show the correctness of the construction, i.e., properties (V_1) and (V_2) hold.

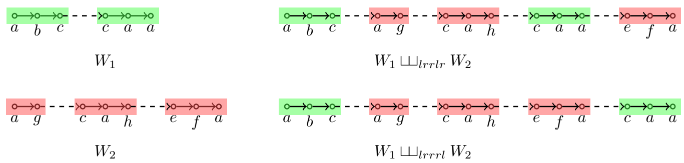
(V_1) Let τ be an STT accepted by $\mathcal{A}^{K,M}_{\mathsf{valid}}$. There is an accepting run of $\mathcal{A}^{K,M}_{\mathsf{valid}}$ reading τ and reaching state q at the root of τ. The state q is an abstraction of τ, hence $([\![\tau]\!], \dashrightarrow)$ is a split-STCW. But since q is accepting, we have $\dashrightarrow\, = \emptyset$. Hence $[\![\tau]\!]$ is a simple TCW. Moreover, for every subterm τ' of τ, the number of blocks in $[\![\tau']\!]$ is the number of blocks in the state q' that labels τ' in this accepting run. By definition, the number of blocks of q' is at most K. Therefore, τ describes a split-decomposition of $[\![\tau]\!]$ of width at most K.

(V_2) Let $\overline{T}$ be a (K,M)-STTs arising from a split-tree T of width at most K and using constants less than M. Following the inductive construction of $\overline{T}$ from T given in the proof of Lemma 20, it is easy to see that $\mathcal{A}^{K,M}_{\mathsf{valid}}$ admits a run on $\overline{T}$. The conditions for applying the transitions of $\mathcal{A}^{K,M}_{\mathsf{valid}}$ are always satisfied. The only non-deterministic transitions of $\mathcal{A}^{K,M}_{\mathsf{valid}}$ are at $\oplus$ nodes, where the automaton should guess how blocks of the children are shuffled. The correct guess is obtained by following the total order in the simple TCW $[\![\overline{T}]\!]$. The state q reached by $\mathcal{A}^{K,M}_{\mathsf{valid}}$ when reading $\overline{T}$ is an abstraction of a simple TCW, hence it must be one of the accepting states of $\mathcal{A}^{K,M}_{\mathsf{valid}}$.

Finally, we show that $\mathsf{STCW}^{K,M} = \{[\![\tau]\!] \mid \tau \in \mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{valid}})\}$. The right to left inclusion follows immediately from (V_1). Conversely, let $\mathcal{V} \in \mathsf{STCW}^{K,M}$. By Lemma 20 there exists a (K,M)-STT τ such that $[\![\tau]\!] = \mathcal{V}$. Moreover, $\tau = \overline{T}$ where T is a split-tree for $\mathcal{V}$. By (V_2) we deduce that $\tau \in \mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{valid}})$. ◻

C Section 5: Tree automaton for realizability

Shuffle operation 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 $\mathcal{V} = \mathcal{V}_1 \sqcup\!\sqcup_{\leq} \mathcal{V}_2$. This total order can be unambiguously specified by a word over the alphabet $\{\ell, r\}$ as follows. If $\mathsf{width}(\mathcal{V}_1) = k$ and $\mathsf{width}(\mathcal{V}_2) = m$, then the shuffle of $\mathcal{V}_1$, $\mathcal{V}_2$ is specified by a word $x \in \{\ell, r\}^{k+m}$ denoted $\mathcal{V}_1 \sqcup\!\sqcup_x \mathcal{V}_2$. If the ith letter of x is ℓ, then the ith block, $1 \leq i \leq k+m$ of $\mathcal{V}_1 \sqcup\!\sqcup_x \mathcal{V}_2$ is taken from $\mathcal{V}_1$; otherwise it is taken from $\mathcal{V}_2$, respecting $<$. An example is shown below.



Contexts, realizability under contexts 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. For $\mathsf{width}(\mathcal{V}) = k$, $C \circ \mathcal{V} = C \sqcup\!\sqcup_x \mathcal{V}$ with $x = (\ell r)^k \ell$. 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 $C \in \mathsf{TCW}$ 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 come up with a sound abstraction of finite index, so that a finite tree automaton can be defined which works only on the representatives. The operation $\sqcup\!\sqcup$ preserves the equivalence between split-TCW as shown below.

Lemma 27 (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, for all $x \in \{\ell, r\}^n$ with $n = \mathsf{width}(U_1) + \mathsf{width}(U_2)$, we have $U_1 \sqcup\!\sqcup_x U_2 \sim_M U_1' \sqcup\!\sqcup_x U_2'$.*

Proof. Let $\mathsf{width}(U_1) = K_1$, $\mathsf{width}(U_2) = K_2$ and let $C \in \mathsf{TCW}$ be a context of width $K_1 + K_2 + 1$. Also, Let $U = U_1 \sqcup\!\sqcup_x U_2$ and let $U' = U_1' \sqcup\!\sqcup_x U_2'$ with $x \in \{\ell, r\}^{K_1+K_2}$. Then it can be seen from Figure 9 that there exist contexts $C_1 \in \mathsf{TCW}$ of width $K_2 + 1$ and $C_2 \in \mathsf{TCW}$ of width $K_1 + 1$ such that $C \circ U = C_2 \circ U_1 \sim C_2 \circ U_1' = C_1 \circ U_2 \sim C_1 \circ U_2' = C \circ U'$. ◻

C.1 The (possibly infinite) tree automaton for realizability

We now build a tree automaton for realizability, which can possibly have infinitely many states. Subsequently, we show (i) some conditions under which it has finitely many states and (ii) additional abstractions ensuring that it is always finite.

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

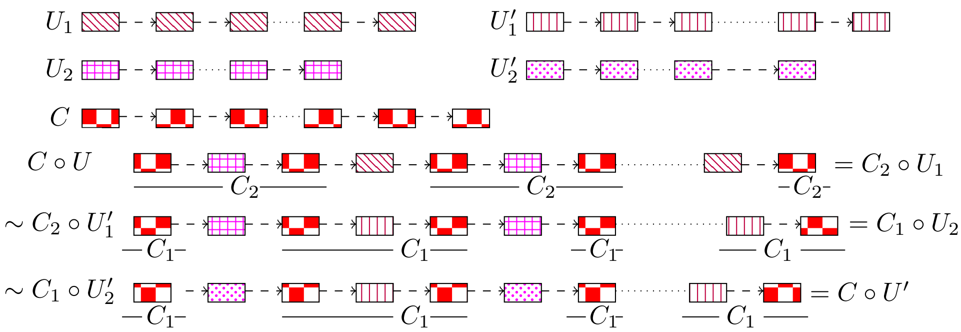


Figure 9 The congruence Lemma

This construction will build on the construction of $\mathcal{A}^{K,M}_{\mathsf{valid}}$ from Appendix B. States of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ will be pairs (q, wt) where $q = (P, <, \rightarrow)$ is a state of $\mathcal{A}^{K,M}_{\mathsf{valid}}$ and $\mathsf{wt} \colon P^2 \to \overline{\mathbb{Z}} = \mathbb{Z} \cup \{+\infty\}$ gives the timing constraints. Note that as $\mathcal{A}^{K,M}_{\mathsf{valid}}$ is finite the first component is finite. But as we will see below the weights in the states can grow unboundedly in general. We assume that $\mathsf{wt}(k,k) = 0$ for all $k \in P$. Also, if $i < j$ are points then $\mathsf{wt}(j,i) \leq 0 \leq \mathsf{wt}(i,j)$.

Below, we identify such a pair (q, wt) with a split-TCW, ignoring the $\rhd$ relation since the weight function is totally defined on P^2, and ignoring the Σ-labelings of nodes since they are irrelevant for the realizability of TCWs.

To help understand the construction, we first give the invariant that will be maintained by the automaton. Let τ be a (K,M)-STT with $[\![\tau]\!] = (V, \rightarrow, \lambda, \rhd, \theta, \chi)$. Assume that there is a (bottom-up) run of $\mathcal{A}^{K,M}_{\mathsf{inf}}$ reading τ and reaching state (q, wt) with $q = (P, <, \rightarrow)$. Then, projecting this run on the first component, we obtain a run of $\mathcal{A}^{K,M}_{\mathsf{valid}}$ reading τ and reaching state q. From (A_3) of $\mathcal{A}^{K,M}_{\mathsf{valid}}$, the state q induces a total order on the blocks of $[\![\tau]\!]$ and turns $[\![\tau]\!]$ into a split-STCW $([\![\tau]\!], \dashrightarrow)$. 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:

(S) $([\![\tau]\!], \dashrightarrow) \sim_M (q, \mathsf{wt})$.

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.

- Atomic STTs: When reading the atomic STT $\tau = (1, a)$ with $a \in \Sigma$, the tree automaton $\mathcal{A}^{K,M}_{\mathsf{inf}}$ moves to state (q, wt) where $q = (\{1\}, \emptyset, \emptyset)$ is the state reached by $\mathcal{A}^{K,M}_{\mathsf{valid}}$ on $(1, a)$, and $\mathsf{wt}(1,1) = 0$. Similarly, when reading an atomic STT $\tau = \mathsf{Add}^{c,d}_{1,2}((1,a) \oplus (2,b))$, the tree automaton $\mathcal{A}^{K,M}_{\mathsf{inf}}$ moves to state (q, wt) where $q = (\{1,2\}, 1 < 2, \emptyset)$ is the state reached by $\mathcal{A}^{K,M}_{\mathsf{valid}}$ on τ, and $\mathsf{wt}(1,1) = 0 = \mathsf{wt}(2,2)$, $\mathsf{wt}(1,2) = d$ 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 i and j in (q, wt). Exchanging colors preserves soundness: if $\tau' = \mathsf{Rename}_{i,j}\,\tau$ and (q, wt) is a sound abstraction of τ then (q', wt') is a sound abstraction of τ'.
- $\mathsf{Add}^{\rightarrow}_{i,j}$: We define transitions $(q, \mathsf{wt}) \xrightarrow{\mathsf{Add}^{\rightarrow}_{i,j}} (q', \mathsf{wt}')$, when $q \xrightarrow{\mathsf{Add}^{\rightarrow}_{i,j}} q'$ is a transition in $\mathcal{A}^{K,M}_{\mathsf{valid}}$, and $\mathsf{wt}' = \mathsf{wt}$. Then, if $\tau' = \mathsf{Add}^{\rightarrow}_{i,j}\,\tau$ and (q, wt) is a sound abstraction of τ, it immediately follows that (q', wt') is a sound abstraction of τ'. This is because adding of an edge only reduces the 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 the following hold
  - $q_1, q_2 \xrightarrow{\oplus} q = (P, <, \rightarrow)$ is a transition in $\mathcal{A}^{K,M}_{\mathsf{valid}}$,

- the weights are inherited and completed as follows: for all $i, j \in P = P_1 \uplus P_2$,

$$\mathsf{wt}(i,j) = \begin{cases} \mathsf{wt}_1(i,j) & \text{if } i,j \in P_1 \\ \mathsf{wt}_2(i,j) & \text{if } i,j \in P_2 \\ \infty & \text{if } (i,j) \in (P_1 \times P_2) \cup (P_2 \times P_1) \text{ and } i < j \\ 0 & \text{if } (i,j) \in (P_1 \times P_2) \cup (P_2 \times P_1) \text{ and } i > j \end{cases}$$

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. Indeed, by definition of $\oplus$-transitions in $\mathcal{A}^{K,M}_{\mathsf{valid}}$, the total ordering $<$ of q indicates how blocks of q_1 and q_2 are shuffled. Hence $(q, \mathsf{wt}) = (q_1, \mathsf{wt}_1) \sqcup\!\sqcup_x (q_2, \mathsf{wt}_2)$ for some $x \in \{\ell, r\}^*$. Now, the induced ordering $\dashrightarrow$ on the blocks of $[\![\tau]\!]$ corresponds to the same shuffle of blocks, i.e., $([\![\tau]\!], \dashrightarrow) = ([\![\tau_1]\!], \dashrightarrow_1) \sqcup\!\sqcup_x ([\![\tau_2]\!], \dashrightarrow_2)$. Now, applying the congruence Lemma 27, 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
  - $q = (P, <, \to) \xrightarrow{\mathsf{Forget}_i} q' = (P', <', \to')$ is a transition of $\mathcal{A}^{K,M}_{\mathsf{valid}}$ (in particular, i is not an endpoint),
  - i is not part of a negative cycle of length 2: for all $j \neq i$ we have $\mathsf{wt}(j,i) + \mathsf{wt}(i,j) \geq 0$,
  - 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.

Notice that 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 have

Claim 28. *If $\tau' = \mathsf{Forget}_i\, \tau$ and $W = (q, \mathsf{wt})$ is a sound abstraction of τ, then $W' = (q', \mathsf{wt}')$ is a sound abstraction of τ'.*

Proof. We will first show that $W \sim_M W'$. Notice that since i is not an endpoint, W and W' have the same number of blocks. Let $C \in \mathsf{TCW}$ be an M-context with $\mathsf{width}(C) = 1 + \mathsf{width}(W) = 1 + \mathsf{width}(W')$. We show that $C \circ W$ is realizable iff $C \circ W'$ is realizable. Suppose $C \circ W$ has a negative cycle Q. Wlog we may assume that Q is simple.

1. If this cycle Q does not pass through i, then the same cycle Q is present in $C \circ W'$, and by definition of wt', the weight of Q in $C \circ W'$ is at most the weight of Q in $C \circ W$.
2. Let the cycle Q pass through i. Since i is not part of a negative cycle of length 2, there are points $j \neq i \neq k$ such that j, i, k is a part of the cycle. Removing i, we obtain a cycle Q' containing j, k in $C \circ W'$. By definition, $\mathsf{wt}'(j,k) \leq \mathsf{wt}(j,i) + \mathsf{wt}(i,k)$, and hence $\mathsf{wt}'(Q') \leq \mathsf{wt}(Q)$.

In both cases, we obtain a negative cycle Q' in $C \circ W'$.
Conversely, assume that there is a negative cycle Q' in $C \circ W'$. We obtain a cycle Q in $C \circ W$ by inserting i between each pair of consecutive points j, k in Q' such that $\mathsf{wt}'(j,k) < \mathsf{wt}(j,k)$. Notice that in this case $\mathsf{wt}'(j,k) = \mathsf{wt}(j,i) + \mathsf{wt}(i,k)$, and otherwise $\mathsf{wt}'(j,k) = \mathsf{wt}(j,k)$. Therefore, $\mathsf{wt}(Q) = \mathsf{wt}'(Q')$ and we have a negative cycle in $C \circ W$ as well. This completes the proof that $W' \sim_M W$.
Now, we know by hypothesis that $W \sim_M ([\![\tau]\!], \dashrightarrow)$. Further, we also have $([\![\tau]\!], \dashrightarrow) \sim_M ([\![\tau']\!], \dashrightarrow')$ since τ' differs from τ only in the coloring; the remaining part is the same and in particular $\dashrightarrow' = \dashrightarrow$. Hence the paths (and cycles) are exactly the same. Thus, combining these, we obtain $W' \sim_M ([\![\tau']\!], \dashrightarrow)$, which completes the proof of the claim. ◻

- Accepting condition: Finally, we define a state (q, wt) to be accepting if
  - q is an accepting state of $\mathcal{A}^{K,M}_{\mathsf{valid}}$, hence it consists of a single block with left endpoint i and right endpoint j (possibly $i = j$),

- the pair (q, wt) is realizable, i.e., $\mathsf{wt}(i,j) + \mathsf{wt}(j,i) \geq 0$.

This completes the construction of $\mathcal{A}_{\mathsf{inf}}^{K,M}$, whose correctness follows from the claim below and thus, completes the proof of Proposition 12.

Claim 29. $\mathcal{L}(\mathcal{A}_{\mathsf{inf}}^{K,M}) = \{\tau \in \mathcal{L}(\mathcal{A}_{\mathsf{valid}}^{K,M}) \mid [\![\tau]\!]$ *is realizable*$\}$.

Proof. Suppose $\mathcal{A}_{\mathsf{inf}}^{K,M}$ has an accepting run reading τ reaching state (q, wt) at the root. Then τ is accepted by $\mathcal{A}_{\mathsf{valid}}^{K,M}$. Moreover, (q, wt) is a sound abstraction of τ, which implies $([\![\tau]\!], \dashrightarrow) \sim_M (q, \mathsf{wt})$. Now, since q has a single block, so does $[\![\tau]\!]$ and the hole relation is empty. Therefore, $[\![\tau]\!] \sim_M (q, \mathsf{wt})$. Since the latter is realizable by definition of accepting states, we deduce that $[\![\tau]\!]$ is realizable.

Conversely, let $\tau \in \mathcal{L}(\mathcal{A}_{\mathsf{valid}}^{K,M})$ be a (K, M)-STT such that $[\![\tau]\!]$ is realizable. Each accepting run of $\mathcal{A}_{\mathsf{valid}}^{K,M}$ on τ defines a unique run of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ on τ. Indeed,

- (existence) the only additional precondition for a transition to be enabled in $\mathcal{A}_{\mathsf{inf}}^{K,M}$ is the fact that i should not be part of a negative cycle of length 2 when reading Forget_i. Assume that there is a subterm $\mathsf{Forget}_i\, \tau'$ of τ and that in the state (q', wt') associated with τ', point i is part of a negative cycle of length 2. Then (q', wt') is not realizable and since it is a sound abstraction of τ' it follows that $([\![\tau']\!], \dashrightarrow')$ is not realizable either. But $[\![\tau']\!]$ is a subgraph of $[\![\tau]\!]$. This would imply that $[\![\tau]\!]$ is not realizable, a contradiction.
- (uniqueness) the wt component in the states of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ is always computed deterministically by its transitions.

Finally, let (q, wt) be the state reached at root of τ in this run of $\mathcal{A}_{\mathsf{inf}}^{K,M}$. Since we started from an accepting run of $\mathcal{A}_{\mathsf{valid}}^{K,M}$, state q is accepting in $\mathcal{A}_{\mathsf{valid}}^{K,M}$. Moreover, since $[\![\tau]\!]$ is realizable and (q, wt) is a sound abstraction of τ, we also have (q, wt) realizable. Therefore, (q, wt) is accepting in $\mathcal{A}_{\mathsf{inf}}^{K,M}$. ◻

Observe that the number of states of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ may grow unboundedly, since the constants in wt' can grow in the automaton (due to taking sum during *forget transitions*). A backward edge $j > k$ with $j > i > k$ may grow in absolute value with the update $\mathsf{wt}'(j,k) = \min(\mathsf{wt}(j,k), \mathsf{wt}(j,i) + \mathsf{wt}(i,k))$. On the other hand, a forward edge $j < k$ may get a big value due to the formula above only if $\mathsf{wt}(j,k) = \infty$. After having a finite value, the weight of a forward edge may only decrease due to the $\min$ operation.

A first question we ask is if there are classes where this does not happen. A simple solution is to consider time-bounded classes where all behaviors must occur within some global time bound T. The idea here is to check if some backward edge grows above T in absolute value after a forget move, in which case we reject it; while if the same happens with a forward edge, then replace it with ∞.

Corollary 30. *If the system is time-bounded by some constant T, then there exists a finite tree automaton $\mathcal{A}_{\mathsf{real}}^{K,M}$ of size at most $T^{\mathcal{O}(k^2)} \cdot 2^{\mathcal{O}(k^2 \lg k)}$ for checking realizability.*

However, in general, when we do not assume a global time bound, the constants in the states of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ may grow unboundedly. We next show how to modify the above construction so that we can make sure that the constants are always bounded and hence obtain a finite tree automaton for realizability. Thus, this generalizes the above corollary with a better complexity.

C.2 Bounding the constants

We now prove Proposition 7, by constructing a finite tree automaton $\mathcal{A}_{\mathsf{real}}^{K,M}$ to check realizability. The set of states of $\mathcal{A}_{\mathsf{real}}^{K,M}$ will be a finite subset of the states of $\mathcal{A}_{\mathsf{inf}}^{K,M}$. More precisely, a state (q, wt)

of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ with $q = (P, <, \rightarrow)$ is a state of $\mathcal{A}_{\mathsf{real}}^{K,M}$ if for all $i, j \in P$ we have $\mathsf{wt}(i, j) = +\infty$ or $|\mathsf{wt}(i, j)| \leq 8KM$, where K and M are constants defined earlier.

As mentioned earlier, the weights on back and forward edges may grow (due to taking sums) after each *forget* transition. We will define three transformations σ, β, γ which change the weights of a state without affecting its realizability under contexts[1]. Each transformation maps a state (q, wt) of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ to another state (q, wt') of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ such that $(q, \mathsf{wt}') \sim_M (q, \mathsf{wt})$.

For instance, the simplest of these transformations is the map σ defined by $\sigma(q, \mathsf{wt}) = (q, \mathsf{wt}')$ where for all $i < j$, $\mathsf{wt}'(i, j) = \mathsf{wt}(i, j)$ and $\mathsf{wt}'(j, i) = \min\{\mathsf{wt}(j', i') \mid i \leq i' \leq j' \leq j\}$. In other words, the transformation σ strengthens the constraints of backward edges. It is then easy to check that σ preserves realizability under contexts:

Claim 31. *For all states (q, wt) of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ we have $\sigma(q, \mathsf{wt}) \sim_M (q, \mathsf{wt})$.*

Proof. Let $W = (q, \mathsf{wt})$ and $W' = \sigma(W) = (q, \mathsf{wt}')$. Observe that since we apply only min operations, the weights of edges given by wt' in $\sigma(W)$ can only be lower than the weights of edges given by wt in W. Let C be a context. If Q is a negative cycle in $C \circ W$, then the same cycle taken in $\sigma(W)$ will have either the same or a lower total weight and hence be negative. In the other direction, suppose $C \circ \sigma(W)$ has a negative weight cycle Q: $i_1 i_2 \ldots i_r = i_1$. Consider the weight of the same cycle in W. The weights of the forward edges are the same. For each backedge $e = (i_j, i_{j+1})$, by definition of σ, there exists a backedge e' contained in (or possibly equal to) e such that $\mathsf{wt}'(e) = \mathsf{wt}(e)$. That is, $e' = (k, \ell)$ such that $i_{j+1} \leq \ell < k \leq i_j$. Then we can replace in Q, each such backedge e by the backward path segment $i_j\, k\, \ell\, i_{j+1}$, whose weight is lesser than or equal to $\mathsf{wt}'(e) = \mathsf{wt}(e')$. Thus, we have a negative cycle in $C \circ W$. ☐

The backward abstraction β Next, to bound back edges we define a transformation β which reduces the weight of a back edge when it goes above a certain constant in absolute value, while *preserving realizability under all contexts*. In fact, we define it on back edges across a block. Let (q, wt) be a state of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ 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 a 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 $i < j$,

$$\mathsf{wt}'(i, j) = \mathsf{wt}(i, j) + \sum_{e \in \mathcal{B}(i,j)} (M + \mathsf{wt}(e)) \tag{1}$$

$$\mathsf{wt}'(j, i) = \mathsf{wt}(j, i) - \sum_{e \in \mathcal{B}(i,j)} (M + \mathsf{wt}(e)) \tag{2}$$

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).

[1] To be precise, β, γ will preserve realizability under contexts only under additional hypotheses, which we will define later and preserve throughout.

Lemma 32. *For all states $W = (q, \mathsf{wt})$ of $\mathcal{A}_{\mathit{inf}}^{K,M}$ with $q = (P, <, \rightarrow)$ such that all points are endpoints $P = \mathsf{EP}(W)$, we have $W \sim_M \beta(W)$.*

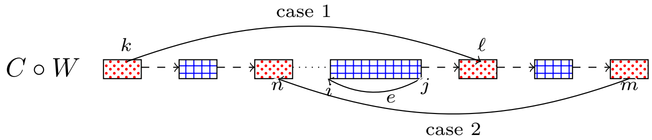


Figure 10 Backward Abstraction

Proof. Let C be an M-context. Recall that constants in C have absolute value less than M.
Case 1. Assume that there is forward edge $k \curvearrowright \ell$ in C going over a BBBE $e = i \curvearrowleft j$ in W. In $C \circ W$, we have the cycle Q consisting of $i \rightsquigarrow k, \ell \rightsquigarrow j, i$, Here, $\rightsquigarrow$ represent backward paths that lead from one point to another. The cycle Q is negative in both W and $\beta(W)$. This follows since $\mathsf{wt}(Q) = \mathsf{wt}(i \rightsquigarrow k) + \mathsf{wt}(k, \ell) + \mathsf{wt}(\ell \rightsquigarrow j) + \mathsf{wt}(j, i) \leq 0 + \mathsf{wt}(k, \ell) + 0 + \mathsf{wt}(j, i) < M + 0 - M + 0 = 0$. The strict inequality follows from the fact that C is an M-context and (k, ℓ) belongs to C. Thus in this case $W \sim_M \beta(W)$.
Case 2. Assume below that there is no forward edges in C going over a BBBE of W.

Let Q be a path in $C \circ W$. Assume Q uses a backedge $n \curvearrowleft m$ in C going over a BBBE $e = i \curvearrowleft j$ in W. Then we replace in Q the edge $n \curvearrowleft m$ by the path $m \rightsquigarrow j, i \rightsquigarrow n$. We then have $\mathsf{wt}(m \rightsquigarrow j) + \mathsf{wt}(j, i) + \mathsf{wt}(i \rightsquigarrow n) \leq 0 + \mathsf{wt}(j, i) + 0 < \mathsf{wt}(m, n)$ since C is an M-context. Thus we obtain a path Q' in $C \circ W$ with $\mathsf{wt}(Q') < \mathsf{wt}(Q)$. We deduce that $C \circ W$ has a negative cycle Q iff $C \circ W$ has a negative cycle Q' using no edges in C crossing over a BBBE of W, forward or backward.

Similarly, we can show that $C \circ \beta(W)$ has a negative cycle Q iff $C \circ \beta(W)$ has a negative cycle Q' using no edges in C crossing over a BBBE of $\beta(W)$. Recall that a BBE is big in W iff it is big in $\beta(W)$.

Therefore, in order to prove that $C \circ W$ is realizable iff $C \circ \beta(W)$, it suffices to consider cycles Q using no edges in C crossing over a BBBE of W. Now, for such a cycle, we can show that the weight $\mathsf{wt}(Q)$ in $C \circ W$ equals the weight $\mathsf{wt}'(Q)$ in $C \circ \beta(W)$. Indeed, for each BBBE $e = i \curvearrowleft j$ in W, the number of forward edges $k \curvearrowright \ell$ in Q going accross e (i.e., $k \leq i < j \leq \ell$) equals the number of backward edges $k' \curvearrowleft \ell'$ going across e (i.e., $k' \leq i < j \leq \ell'$). This uses the fact that all points of W are endpoints. Moreover, by the hypothesis on Q, none of these edges are from C. We deduce that $\mathsf{wt}(Q) = \mathsf{wt}'(Q)$. ◻

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. For this, we need to define another property, that we will inductively maintain as an invariant in the tree automaton.

Definition 33 (Backward-edge property (BEP))**.** A split-TCW W is said to satisfy the *backward-edge property* (BEP) if for all $i \leq j \leq k \leq \ell$ with either $j \dashrightarrow k$ or $j = k$, we have $\mathsf{wt}(\ell, i) > \mathsf{wt}(\ell, k) - M + \mathsf{wt}(j, i)$.

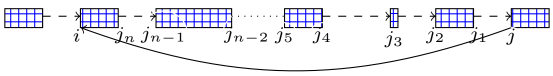
With this, we have our second and crucial invariant.

(I2) $\mathcal{A}_{\mathsf{real}}^{K,M}$ always satisfies BEP.

Note that, by definition, any M-context C satisfies BEP. Preserving this invariant requires a slight transformation of the shuffle operation (at a $\oplus$ node), which we will discuss later in this appendix. However, assuming that Invariant (I2) holds, we now show that all back edges are bounded.

Lemma 34. *Let $W = (q, \mathsf{wt})$ be a state of $\mathcal{A}_{\mathit{inf}}^{K,M}$ 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 $2KM$.*

Proof. Assume that $W' = \beta(W) = (q, \mathsf{wt}')$ satisfies BEP. Let $i \curvearrowleft j$ be some backward edge in W', i.e., $i < j$. Then there exist some $n \geq 0$ points $i = j_{n+1} \lessdot j_n \lessdot j_{n-1} \lessdot \ldots \lessdot j_1 \lessdot j_0 = j$. Notice that in W' also, all points are endpoints. Hence, for all $0 \leq k \leq n$ we have either $j_{k+1} \dashrightarrow j_k$ or $j_{k+1} \rightarrow j_k$ and in the latter case, (j_{k+1}, j_k) is a block.



We show by induction on $m-k$ that for $0 \leq k < m \leq n+1$ we have $\mathsf{wt}'(j_k, j_m) \geq -(m-k)M$. In particular, we deduce that $\mathsf{wt}'(j, i) \geq -(n+1)M$.

If $m = k+1$ then either $j_{k+1} \dashrightarrow j_k$ is a hole and applying BEP we get $\mathsf{wt}'(j_k, j_{k+1}) > \mathsf{wt}'(j_k, j_k) - M + \mathsf{wt}'(j_{k+1}, j_{k+1}) = -M$, or $j_{k+1} \rightarrow j_k$ is a block and by definition of β we get $\mathsf{wt}'(j_k, j_{k+1}) \geq -M$.

Now, assume that $m > k+1$. Since we only have endpoints, there must be a hole $j_{\ell+1} \dashrightarrow j_\ell$ for some $k \leq \ell < m$. Then applying BEP we get $\mathsf{wt}'(j_k, j_m) > \mathsf{wt}'(j_k, j_\ell) - M + \mathsf{wt}'(j_{\ell+1}, j_m)$. Applying the induction hypothesis we obtain $\mathsf{wt}'(j_k, j_\ell) \geq -(\ell - k)M$ and $\mathsf{wt}'(j_{\ell+1}, j_m) \geq -(m - (\ell+1))M$. We deduce that $\mathsf{wt}'(j_k, j_m) \geq -(m-k)M$ as desired.

Applying the above property to $k = 0$ and $m = n+1$, we deduce that $\mathsf{wt}'(j, i) \geq -(n+1)M$. Now the number of points in P is at most $2K$, therefore, $n+1 \leq 2K-1$, which concludes the proof of the lemma. $\square$

Forward abstraction γ. Finally, we turn to forward edges. We apply a *forward abstraction* γ that 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 $\mathsf{wt}(i, j) + \sum_{e \in \mathsf{BBE}(i,j)} \mathsf{wt}(e) \geq (3K-1)M$. Note that as e is a (block) back edge, its weight is negative. Then we define $\gamma(q, \mathsf{wt}) = (q, \mathsf{wt}')$ where, for any $i < j$, $\mathsf{wt}'(j, i) = \mathsf{wt}(j, i)$ and

$$\mathsf{wt}'(i, j) = \begin{cases} \infty & \text{if } (i, j) \text{ is } \textit{big} \\ \mathsf{wt}(i, j) & \text{otherwise.} \end{cases}$$

While the definition of this abstraction is simple, it turns out that showing that it is sound (i.e., it preserves realizability under all contexts) is rather tricky, even with the additional BEP assumption (which will be maintained inductively on the tree automaton).

Lemma 35. *If $W = (q, \mathsf{wt})$ is a state of $\mathcal{A}_{\mathit{inf}}^{K,M}$ which satisfies BEP, then we have $W \sim_M \gamma(W)$.*

Proof. Let $W = (q, \mathsf{wt})$ be a state of $\mathcal{A}_{\mathsf{inf}}^{K,M}$ with $q = (P, <, \rightarrow)$. Assume that W satisfies BEP. Let C be an M-context. Note that $C \circ W$ and $C \circ \gamma(W)$ have the same paths and the same cycles. Moreover, if Q is such a cycle, its weight $\mathsf{wt}'(Q)$ in $C \circ \gamma(W)$ is at most its weight $\mathsf{wt}(Q)$ in $C \circ W$. Therefore, if there is a negative cycle in $C \circ \gamma(W)$ then the same cycle is negative in $C \circ W$.

Conversely, we have to prove now that if there is a negative cycle in $C \circ W$ then there is also a negative cycle in $C \circ \gamma(W)$. So we assume that $C \circ W$ has a negative cycle. Let Q be a shortest (in terms of path length) negative cycle in $C \circ W$. We will show that Q must be also a negative cycle in $C \circ \gamma(W)$, i.e., $\mathsf{wt}'(Q) < 0$.

If Q does not pass through any *big* forward edge then $\mathsf{wt}'(Q) = \mathsf{wt}(Q)$ and we are done.

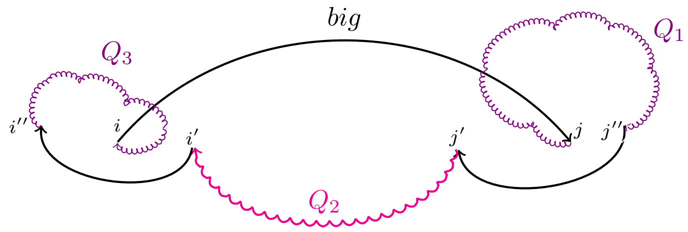


Figure 11 $i \lessdot j_n \lessdot j_{n-1} \lessdot \cdots \lessdot j_1 \lessdot j_0 = j$. Also, $i \le i' \le j' \le j$

Else, suppose Q passed through $i \curvearrowright j$, a *big* edge in W. Then, we can always break Q into $ijQ_1j''j'Q_2i'i''Q_3i$ as depicted in Figure 11, where $j' \curvearrowleft j''$ is the last time Q crosses j backward and $i'' \curvearrowleft i'$ is the first time that Q crosses i backward (both these must happen since the cycle has to return from j to i). As a result, we have that Q_2 lies completely within i and j, i.e., for all $\ell \in Q_2$, $i \le \ell \le j$.

Now, let the points within i to j be $i \lessdot j_n \lessdot j_{n-1} \dots j_1 \lessdot j_0 = j$ for some $n \ge 0$. Applying BEP on backedge $j' \curvearrowleft j''$ with respect to point $j_0 = j$, we obtain $\mathsf{wt}(j'', j') \ge \mathsf{wt}(j'', j_0) - M + \mathsf{wt}(j_0, j')$. Similarly we apply BEP on $i'' \curvearrowleft i'$ with respect to i. Denoting by R_0 the path $j_0j'Q_2i'i$, we maintain the invariant that R_0 lies to the left of j_0 (and right of i). We have:

$$
\begin{aligned}
&0 > \mathsf{wt}(Q) \\
&= \mathsf{wt}(i,j) + \mathsf{wt}(Q_1) + \mathsf{wt}(j'',j') + \mathsf{wt}(Q_2) + \mathsf{wt}(i',i'') + \mathsf{wt}(Q_3) \\
&\ge \mathsf{wt}(i,j) + \mathsf{wt}(Q_1) + \mathsf{wt}(j'',j_0) - M + \mathsf{wt}(R_0) - M + \mathsf{wt}(i,i'') + \mathsf{wt}(Q_3) \\
&\ge \mathsf{wt}(i,j) + 0 - M + \mathsf{wt}(R_0) - M + 0 \\
&= \mathsf{wt}(i,j) - 2M + \mathsf{wt}(R_0)
\end{aligned}
$$

In the above we have $\mathsf{wt}(Q_1) + \mathsf{wt}(j'', j_0) \ge 0$, else we have a shorter negative cycle $jQ_1j''j$. Similarly $\mathsf{wt}(i, i'') + \mathsf{wt}(Q_3) \ge 0$. Now, the sketch of the proof is as follows: We consider the points between i and j and process them inductively, from the right. We start from $j = j_0$ and for each point, we use BEP to move left while maintaining the invariant that the resulting path cannot go right (this is true when we begin since as we observed above R_0 indeed stays within $[i, j]$). Finally, since the number of points in R_0 can at most be $2K$ we terminate and are left with a lower bound on $\mathsf{wt}(R_0)$. This in conjuction with the equation above gives the theorem.

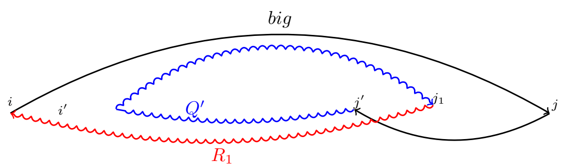


Figure 12 R_0 passes through j_1. The blue path Q' originates in j' and ends in j_1. The red path R_1 originates in j_1 and reaches i, and stays between i and j_1.

Let us start with $j_1 \lessdot j_0$. We have two cases:

- Assume that R_0 passes through j_1 (see Figure 12). Then $R_0 = j_0j'Q'j_1R_1i$ where Q' is the part of R_0 going from j' to j_1 and R_1 is the part of R_0 going from j_1 to i. Thus R_1 satisfies the invariant property that it stays between i and j_1. Applying BEP on backedge $j' \curvearrowleft j$ wrt the point

j_1 we have:

$$\begin{aligned}\mathsf{wt}(R_0) &= \mathsf{wt}(j_0, j') + \mathsf{wt}(Q') + \mathsf{wt}(R_1)\\ &\geq \mathsf{wt}(j_0, j_1) - M + \mathsf{wt}(j_1, j') + \mathsf{wt}(Q') + \mathsf{wt}(R_1)\\ &\geq \mathsf{wt}(j_0, j_1) - M + 0 + \mathsf{wt}(R_1)\end{aligned}$$

Again, we have $\mathsf{wt}(j_1, j') + \mathsf{wt}(Q') \geq 0$ by minimality of the negative cycle Q.

- Assume R_0 does not pass through j_1. Then it cannot pass through anything on the right of j_1, i.e., in this case j_0. Thus, R_0 (after the first backedge $j' \curvearrowleft j_0$) stays on the left of j_1, i.e., we write $R_0 = j_0 j' Q'$ for some Q' which stays to left of j_1. We then define $R_1 = j_1 j' Q'$, which satisfies the invariant property that it lies in $[j_1, i]$. Applying BEP on backedge $j' \curvearrowleft j_1$ wrt the point j_1 we have:

$$\begin{aligned}\mathsf{wt}(R_0) &= \mathsf{wt}(j_0, j') + \mathsf{wt}(Q')\\ &\geq \mathsf{wt}(j_0, j_1) - M + \mathsf{wt}(j_1, j') + \mathsf{wt}(Q')\\ &\geq \mathsf{wt}(j_0, j_1) - M + \mathsf{wt}(R_1)\end{aligned}$$

Inductively, we can apply the same argument for $j_2, \ldots, j_n$ and obtain segments $R_2, \ldots R_n$ (notice that R_n must consist of the single backedge $i \curvearrowleft j_n$ since $i \lessdot j_n$) such that finally, we have:

$$\mathsf{wt}(R_0) \geq \mathsf{wt}(j_0, j_1) - M + \mathsf{wt}(j_1, j_2) - M + \ldots + \mathsf{wt}(j_{n-1}, j_n) - M + \mathsf{wt}(j_n, i)$$

If any $j_{i+1} \curvearrowleft j_i$ is a hole, we apply BEP on the backedge to obtain $\mathsf{wt}(j_i, j_{i+1}) \geq -M$. Now, since $W = (q, \mathsf{wt})$ where q is a state of $\mathcal{A}^{K,M}_{\mathsf{valid}}$, there are at most K blocks in W, hence at most $K-1$ holes. Moreover, there are at most $2K$ points and $n \leq 2K-2$. We deduce that,

$$\begin{aligned}\mathsf{wt}(R_0) &\geq \sum_{e \in \mathsf{BBE}(i,j)} \mathsf{wt}(e) - (K-1)M - (2K-2)M\\ &= \sum_{e \in \mathsf{BBE}(i,j)} \mathsf{wt}(e) - (3K-3)M\end{aligned}$$

This gives:

$$\begin{aligned}0 > \mathsf{wt}(Q) &\geq \mathsf{wt}(i,j) - 2M + \sum_{e \in \mathsf{BBE}(i,j)} \mathsf{wt}(e) - (3K-3)M\\ &\geq (3K-1)M - (3K-1)M = 0\end{aligned}$$

which is a contradiction. The last inequality follows by the definition of a big edge. Thus, no big edge can be part of the shortest negative weight cycle. This shows that the *existence* of negative cycles is preserved by the forward abstraction. □

Construction of $\mathcal{A}^{K,M}_{\mathsf{real}}$ The finite tree automaton $\mathcal{A}^{K,M}_{\mathsf{real}}$ is derived from $\mathcal{A}^{K,M}_{\mathsf{inf}}$ by applying the abstraction σ (strengthening) at $\oplus$ nodes and β (backward) and γ (forward) at Forget_i nodes. More precisely, $(q_1, \mathsf{wt}_1), (q_2, \mathsf{wt}_2) \xrightarrow{\oplus} \sigma(q, \mathsf{wt})$ is in $\mathcal{A}^{K,M}_{\mathsf{real}}$ if $(q_1, \mathsf{wt}_1), (q_2, \mathsf{wt}_2) \xrightarrow{\oplus} (q, \mathsf{wt})$ is in $\mathcal{A}^{K,M}_{\mathsf{inf}}$. Similarly, if $(q, \mathsf{wt}) \xrightarrow{\mathsf{Forget}_i} (q', \mathsf{wt}')$ is a transition in $\mathcal{A}^{K,M}_{\mathsf{inf}}$ then $(q, \mathsf{wt}) \xrightarrow{\mathsf{Forget}_i} (q'', \mathsf{wt}'')$ is in $\mathcal{A}^{K,M}_{\mathsf{real}}$ 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. The reason for performing the σ abstraction on $\oplus$

nodes is to preserve BEP, which is needed for these Lemmas to work. The proof of preservation of BEP is in the next subsection.

The remaining construction of $\mathcal{A}^{K,M}_{\mathsf{real}}$ is directly from $\mathcal{A}^{K,M}_{\mathsf{inf}}$. Atomic STTs are handled as before. Now, if $(q,\mathsf{wt}) \xrightarrow{\alpha} (q',\mathsf{wt}')$ is a transition in $\mathcal{A}^{K,M}_{\mathsf{inf}}$ for $\alpha \in \{\mathsf{Rename}_{i,j}, \mathsf{Add}^{\rightarrow}_{i,j}\}$, then we define $(q,\mathsf{wt}) \xrightarrow{\alpha} (q',\mathsf{wt}')$ to be a transition in $\mathcal{A}^{K,M}_{\mathsf{real}}$ as well. Notice that these transitions do not change the weights, hence if (q,wt) is a state of $\mathcal{A}^{K,M}_{\mathsf{real}}$ then so is (q',wt').

Let us explain why the constants never exceed $8KM$ in states of $\mathcal{A}^{K,M}_{\mathsf{real}}$. By Claim 26, all reachable states of $\mathcal{A}^{K,M}_{\mathsf{valid}}$ (and hence $\mathcal{A}^{K,M}_{\mathsf{real}}$) have the property that they can have at most two internal points. We show that along a run, if a state (q,wt) has no internal points, then the weights/constants are bounded by $4KM$, otherwise, the constants are bounded by $8KM$.

To see this, observe first that after an atomic STT the constants are at most M. Now, transitions $\mathsf{Rename}_{i,j}$, $\mathsf{Add}^{\rightarrow}_{i,j}$ and $\oplus$ do not increase the constants. At $\oplus$ transitions, we apply the σ-abstraction but this does not increase the largest constant occurring in the state. As remarked earlier Claim 26), a transition $\mathsf{Add}^{\rightarrow}_{i,j}$ may create (at most 2) internal points, but it can be applied only when starting from a state with no internal points, hence having constants at most $4KM$. Then, only $\mathsf{Rename}_{i,j}$ and Forget_i transitions may be applied. The first Forget_i eliminates one internal point and may double the constants if an internal point remains. That is why the constants may grow up to $8KM$. After the second Forget_j transition, there are no internal points left, hence the backwared and forward abstractions are applied resulting in a state whose weights are bounded by $4KM$.

Recall that the backward abstraction makes sure that the weight of a back edge $i \curvearrowleft j$ is bounded: $\mathsf{wt}'(j,i) \geq -2KM$ (Lemma 16). And the forward abstraction makes sure that $\mathsf{wt}'(i,j) = +\infty$ or $\mathsf{wt}'(i,j) \leq 4KM$. Therefore, for all states (q,wt) of $\mathcal{A}^{K,M}_{\mathsf{real}}$ are bounded by $8KM$ in absolute value.

Since the transformations σ, β, γ preserve realizability under contexts (Lemma 14 and Lemma 17) we deduce that the key invariant defined earlier is preserved by all transitions of $\mathcal{A}^{K,M}_{\mathsf{real}}$, i.e.,

(I1) $\mathcal{A}^{K,M}_{\mathsf{real}}$ always computes a sound abstraction of the given STT.

The acceptance condition of $\mathcal{A}^{K,M}_{\mathsf{real}}$ is the same as for $\mathcal{A}^{K,M}_{\mathsf{inf}}$. The correctness of the construction now follows on the exact same lines as for $\mathcal{A}^{K,M}_{\mathsf{inf}}$ since the same key invariant is preserved. This completes the proof of Proposition 7.

C.3 Preservation of BEP

It remains to prove that the second key invariant, the BEP property is preserved throughout the tree automaton and for this we will use the fact that at every $\oplus$ node we perform a σ strengthening.

Lemma 36. *$\mathcal{A}^{K,M}_{\mathsf{real}}$ preserves* BEP*, i.e,* BEP *is satisfied when we start and it continues to be satisfied after each transition.*

Proof. First, note that the atomic STTs trivially satisfy BEP since for all back edge $i \curvearrowleft \ell$ we have $\mathsf{wt}(\ell,i) > -M$. Then, consider the transitions one by one.

- $\mathsf{Rename}_{i,j}$: preservation of BEP is trivial since only the names of the points change.
- $\mathsf{Add}^{\rightarrow}_{i,j}$: preservation of BEP is also trivial since we simply removed a hole, hence we have fewer cases to consider for the backward edge property.
- $\oplus$: consider a transition $(q_1,\mathsf{wt}_1),(q_2,\mathsf{wt}_2) \xrightarrow{\oplus} \sigma(q,\mathsf{wt}) = (q,\mathsf{wt}')$ of $\mathcal{A}^{K,M}_{\mathsf{real}}$ coming from a transition $(q_1,\mathsf{wt}_1),(q_2,\mathsf{wt}_2) \xrightarrow{\oplus} (q,\mathsf{wt})$ in $\mathcal{A}^{K,M}_{\mathsf{inf}}$. By induction hypothesis, (q_1,wt_1) and (q_2,wt_2) satisfy BEP. We show that (q,wt') also does. Let $i \leq j \leq k \leq \ell$ in q be such that either $j = k$ or $j \dashrightarrow k$. We have $\mathsf{wt}'(\ell,i) = \mathsf{wt}(n,m)$ for some $i \leq m \leq n \leq \ell$. We have three cases:

- if m and n are not from the same q_i, then $\mathsf{wt}'(\ell,i) = \mathsf{wt}(n,m) = 0 > -M \geq \mathsf{wt}'(\ell,k) - M + \mathsf{wt}'(j,i)$.
  So we assume below that m and n are both from the same state, say q_1.
- if $m \curvearrowleft n$ does not cross j,k. Wlog, let us say that it remains entirely to right of k, i.e., $k \leq m < n$. Then, $\mathsf{wt}'(\ell,i) = \mathsf{wt}(m,n) \geq \mathsf{wt}'(\ell,k) > \mathsf{wt}'(\ell,k) - M + \mathsf{wt}'(j,i)$ since last two terms add a negative quantity.
- if $m \curvearrowleft n$ crosses j,k, i.e., $m \leq j \leq k \leq n$. Let k' be the least point in q_1 which is above k: we have $k \leq k' \leq n$. Similarly, let j' be the greatest point in q_1 below j: $m \leq j' \leq j$. We must have $j' = k'$ or $j' \dashrightarrow k'$. In this case, applying BEP on edge (m,n) of (q_1, wt_1), we get $\mathsf{wt}'(\ell,i) = \mathsf{wt}_1(n,m) \geq \mathsf{wt}_1(n,k') - M + \mathsf{wt}_1(j',m) \geq \mathsf{wt}'(\ell,k) - M + \mathsf{wt}'(j,i)$.

- Forget_c: Let $(q,\mathsf{wt}) \xrightarrow{\mathsf{Forget}_i} (q',\mathsf{wt}')$ be a transition of $\mathcal{A}_{\mathsf{inf}}^{K,M}$. We assume that (q,wt) satisfies BEP. We show that (q',wt') also satisfies BEP. So let $i \leq j \leq k \leq \ell$ be points in q' such that $j \dashrightarrow k$ or $j = k$. Notice that also in q we have $j \dashrightarrow k$ or $j = k$ since a forget transition only removes an internal point. We have $\mathsf{wt}(\ell,i) > \mathsf{wt}(\ell,k) - M + \mathsf{wt}(j,i)$. There are two cases.
  - Case 1: $\mathsf{wt}'(\ell,i) = \mathsf{wt}(\ell,i)$. This case is easy. We have $\mathsf{wt}'(\ell,i) = \mathsf{wt}(\ell,i) > \mathsf{wt}(\ell,k) - M + \mathsf{wt}(j,i) \geq \mathsf{wt}'(\ell,k) - M + \mathsf{wt}'(j,i)$.
  - Case 2: $\mathsf{wt}'(\ell,i) < \mathsf{wt}(\ell,i)$. In this case, by definition of the transition, we must have $\mathsf{wt}'(\ell,i) = \mathsf{wt}(\ell,c) + \mathsf{wt}(c,i)$. First, note that $c \notin \{i,j,k,\ell\}$, since these points are in P' but by definition $c \notin P'$. Now we have two cases, either $k < c$ or $c < j$. Indeed, since $j \dashrightarrow k$ or $j = k$, we cannot have $j < c < k$. Wlog let $k < c$ (the other case is symmetric and follows by similar arguments).

$$\begin{aligned}
\mathsf{wt}'(\ell,i) &= \mathsf{wt}(\ell,c) + \mathsf{wt}(c,i) \\
&\geq \mathsf{wt}(\ell,c) + \big(\mathsf{wt}(c,k) - M + \mathsf{wt}(j,i)\big) && \text{(by BEP on edge } (c,i) \text{ of } (q,\mathsf{wt})) \\
&= \mathsf{wt}(\ell,c) + \mathsf{wt}(c,k) - M + \mathsf{wt}(j,i) \\
&\geq \mathsf{wt}'(\ell,k) - M + \mathsf{wt}'(j,i) && \text{(by defn on wt}')
\end{aligned}$$

- β: Let (q,wt) be a state satisfying BEP and let $\beta(q,\mathsf{wt}) = (q,\mathsf{wt}')$ be its backward abstraction. We will show that (q,wt') also satisfies BEP. Let $i \leq j \leq k \leq \ell$ be such that $j \dashrightarrow k$ or $j = k$. Notice that the BBBE between i and ℓ are either between i and j or between k and ℓ: $\mathcal{B}(i,\ell) = \mathcal{B}(i,j) \uplus \mathcal{B}(k,\ell)$. Hence, we have

$$\begin{aligned}
\mathsf{wt}'(\ell,i) &= \mathsf{wt}(\ell,i) - \sum_{e \in \mathcal{B}(i,\ell)} (\mathsf{wt}(e) + M) \\
&\geq \mathsf{wt}(\ell,k) - M + \mathsf{wt}(j,i) - \sum_{e \in \mathcal{B}(i,\ell)} (\mathsf{wt}(e) + M) && \text{(by BEP on } W) \\
&= \Big(\mathsf{wt}(\ell,k) - \sum_{e \in \mathcal{B}(k,\ell)} (\mathsf{wt}(e) + M)\Big) - M + \Big(\mathsf{wt}(j,i) - \sum_{e \in B(i,j)} (\mathsf{wt}(e) + M)\Big) \\
&= \mathsf{wt}'(\ell,k) - M + \mathsf{wt}'(j,i)
\end{aligned}$$

- γ trivially preserves BEP as it does not change the weights of backedges. □

D Tree automata for timed systems

The goal of this section is to build a tree automaton which accepts the STTs denoting TCWs accepted by a TPDA. We recall the formal statement given in Section 4.

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

Again, we could prove the existence of a tree automaton by arguing that the existence of a run of $\mathcal{S}$ on a simple TCW is MSO definable and appealing to Courcelle's theorem [9]. But, as for the automaton $\mathcal{A}_{\mathsf{valid}}^{K,M}$ in Appendix B we choose to directly construct the tree automaton $\mathcal{A}_{\mathcal{S}}^{K,M}$, allowing in particular to analyse its size which is stated above.

Let us explain first how the automaton would work for an *untimed* system with no stack. At a leaf of the STT of the form (i, a) the tree automaton guesses a transition δ from $\mathcal{S}$ which could be executed reading action a. It keeps the transition in its state, paired with color i. After reading a subterm τ, the tree automaton stores in its state a map Δ which assigns with each active (free) color k of τ the transition $\Delta(k)$ guessed at the corresponding leaf. The map Δ can be easily updated at nodes labelled $\oplus$ or Forget_i or $\mathsf{Rename}_{i,j}$. When, reading a node labelled $\mathsf{Add}_{i,j}^{\rightarrow}$, the tree automaton checks that the target state of the transition $\Delta(i)$ equals the source state of the transition $\Delta(j)$. This ensures that the transitions guessed at leaves form a run when taken in the total order induced by the word denoted by the final term. At the root, we check that at most two colors remain free: i and j for the leftmost (resp. rightmost) endpoint of the word. Then, the tree automaton accepts if the source state of $\Delta(i)$ is initial in $\mathcal{S}$ and the target state of $\Delta(j)$ is final in $\mathcal{S}$.

The situation is a bit more complicated for timed systems. First, we are interested in the *simple* TCW semantics in which each event is blown-up in several micro-events. Following this idea, each transition $(s, \gamma, a, \mathsf{op}, R, s')$ of the timed system $\mathcal{S}$ is blown-up in micro-transitions as explained in Section 2.2, assuming that $\gamma = \gamma_1 \wedge \cdots \wedge \gamma_n$ has n conjuncts of the form $x \in I$, and that $R = \{x_1, x_2, \ldots, x_m\}$:

$$\begin{tikzcd} s \arrow[r, "\{\zeta\}"] & \delta_0 \arrow[r, "\gamma_1"] & \delta_1 \cdots \delta_{n-1} \arrow[r, "\gamma_n"] & \delta_n \arrow[r, "{a, \mathsf{op}}"] & \delta_{x_1} \arrow[loop above, "\{x_1\}"] \arrow[r, "\varepsilon"] & \cdots \arrow[r, "\varepsilon"] & \delta_{x_m} \arrow[loop above, "\{x_m\}"] \arrow[r, "\zeta = 0"] & s' \end{tikzcd}$$

Notice the reset part at the end which allows an arbitrary number (possibly zero) resets of clock x_1 (depending on how many timing constraints for clock x_1 will originate from this reset), followed by an arbitrary number (possibly zero) resets of clock x_2, etc.

The second difficulty is to make sure that, when a guard of the form $x \in I$ is checked in some transition, then the source point of the timing constraint in the simple TCW indeed corresponds to the latest transition resetting clock x. To check this property, the tree automaton $\mathcal{A}_{\mathcal{S}}^{K,M}$ stores,

- for each color k of the left endpoint of a block, the set $C(k)$ of clocks reset by some transition whose *middle* point (the one corresponding to a, op micro-transition) occurs in the block.
- for each pair (i, j) of colors of left endpoints of *distinct* blocks, set $D(i, j)$ of clocks that are reset in block i and checked in j.

Finally, the last property to be checked is that the push-pop edges are well-nested. To this end, the tree automaton $\mathcal{A}_{\mathcal{S}}^{K,M}$ stores the set E of pairs (i, j) of left endpoints of *distinct* blocks such that there is at least one push-pop edge from block i to block j.

Formally, a state of $\mathcal{A}_{\mathcal{S}}^{K,M}$ is a tuple (q, Δ, C, D, E) where $q = (P, <, \rightarrow)$ is a state of $\mathcal{A}_{\mathsf{valid}}^{K,M}$, the map Δ assigns to each color $k \in P$ the pair of states of the micro-transition $\Delta(k)$ guessed at the leaf corresponding to color k, and the maps C, D and the set E are as described above. We describe now the transitions of $\mathcal{A}_{\mathcal{S}}^{K,M}$.

Atomic STTs When reading an atomic STT $\tau = \mathsf{Add}_{1,2}^{c,d}((1, \varepsilon) \oplus (2, \varepsilon))$, the tree automaton $\mathcal{A}_{\mathcal{S}}^{K,M}$ may guess that it encodes a clock constraint $x \in [c, d]$ induced by two transitions $\delta^1 =$

$(s^1, \gamma^1, a^1, \mathsf{op}^1, R^1, s'^1)$ and $\delta^2 = (s^2, \gamma^2, a^2, \mathsf{op}^2, R^2, s'^2)$ of $\mathcal{S}$, i.e., $x \in R^1$ and some conjunct of γ^2, say the k-th, is $x \in [c, d]$. Then it moves to state $(q_{1,2}, \Delta, C, D, E)$ where $q_{1,2} = (\{1, 2\}, 1 < 2, \emptyset)$ is the state reached by $\mathcal{A}^{K,M}_{\mathsf{valid}}$ on τ, and $\Delta(1) = (\delta^1_x, \delta^1_x)$ is the reset micro-transition for clock x of δ^1, $\Delta(2) = (\delta^2_{k-1}, \delta^2_k)$ is the micro-transition checking the k-th conjunct of γ^2, $C(1) = C(2) = \emptyset$, $D(1, 2) = \{x\}$ and $E = \emptyset$.

When reading an atomic STT $\tau = \mathsf{Add}^{c,d}_{1,2}((1, a^1) \oplus (2, a^2))$, the tree automaton $\mathcal{A}^{K,M}_{\mathcal{S}}$ may also guess two matching push-pop transitions of $\mathcal{S}$: $\delta^1 = (s^1, \gamma^1, a^1, \downarrow_b, R^1, s'^1)$ and $\delta^2 = (s^2, \gamma^2, a^2, \uparrow^{c,d}_b, R^2, s'^2)$. Then it moves to state $(q_{1,2}, \Delta, C, D, E)$ where $q_{1,2} = (\{1, 2\}, 1 < 2, \emptyset)$ is the state reached by $\mathcal{A}^{K,M}_{\mathsf{valid}}$ on τ, and $\Delta(1) = (\delta^1_n, \delta^1_x)$ is the *middle* micro-transition of δ^1, $\Delta(2) = (\delta^2_m, \delta^2_y)$ is the *middle* micro-transition of δ^2, $C(1) = R^1$, $C(2) = R^2$, $D(1, 2) = \emptyset$ and $E = \{(1, 2)\}$.

When reading an atomic STT $\tau = \mathsf{Add}^{0,0}_{1,2}((1, \varepsilon) \oplus (2, \varepsilon))$, the tree automaton $\mathcal{A}^{K,M}_{\mathcal{S}}$ may guess that it encodes the ζ clock constraint of some transition $\delta = (s, \gamma, a, \mathsf{op}, R, s')$ of $\mathcal{S}$. Then, it moves to state $(q_{1,2}, \Delta, C, D, E)$ where $q_{1,2} = (\{1, 2\}, 1 < 2, \emptyset)$ is the state reached by $\mathcal{A}^{K,M}_{\mathsf{valid}}$ on τ, and $\Delta(1) = (s, \delta_0)$, $\Delta(2) = (\delta_x, s')$ are the first and last micro-transitions from δ, $C(1) = C(2) = \emptyset$, $D(1, 2) = \emptyset$ and $E = \emptyset$.

Finally, when reading the atomic STT $\tau = (1, a)$ with $a \in \Sigma$, the tree automaton $\mathcal{A}^{K,M}_{\mathcal{S}}$ guesses a transition $\delta = (s, \gamma, a, \mathsf{nop}, R, s')$ of $\mathcal{S}$ and moves to state (q_1, Δ, C, D, E) where $q_1 = (\{1\}, \emptyset, \emptyset)$ is the state reached by $\mathcal{A}^{K,M}_{\mathsf{valid}}$ on $(1, a)$, and $\Delta(1) = (\delta_n, \delta_x)$ is the *middle* micro-transition of δ (assuming that γ has n conjuncts), $C(1) = R$ since the set of clocks reset by δ is R, and D is undefined and $E = \emptyset$ since we have a single block.

$\mathsf{Rename}_{i,j}$ We simply exchange the colors i and j in the current state.

Forget_i We define $(q, \Delta, C, D, E) \xrightarrow{\mathsf{Forget}_i} (q', \Delta', C', D', E')$ to be a transition if

- $q \xrightarrow{\mathsf{Forget}_i} q'$ is a transition of $\mathcal{A}^{K,M}_{\mathsf{valid}}$. In particular, i is not an endpoint.
- We obtain Δ' from Δ by forgetting the entry of color i. Since i is not a left endpoint, the components C, D and E are not affected: $C' = C$, $D' = D$ and $E' = E$.

$\mathsf{Add}^{\rightarrow}_{i,j}$ We define $(q, \Delta, C, D, E) \xrightarrow{\mathsf{Add}^{\rightarrow}_{i,j}} (q', \Delta', C', D', E')$ to be a transition if

- $q \xrightarrow{\mathsf{Add}^{\rightarrow}_{i,j}} q'$ is a transition of $\mathcal{A}^{K,M}_{\mathsf{valid}}$. In particular, i is a right endpoint, j is a left endpoint, and $i \lessdot j$.
- Either the target state of $\Delta(i)$ equals the source state of $\Delta(j)$, or there is an ε-path of micro-transitions $\xrightarrow{\varepsilon} \delta_{x_k} \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \delta_{x_\ell} \xrightarrow{\varepsilon}$ between the target state of $\Delta(i)$ and the source state of $\Delta(j)$: indeed, adding $\rightarrow$ between i and j means that these points are consecutive in the final TCW, hence it should be possible to concatenate the micro-transition taken at i and j.
- $\Delta' = \Delta$.
- Let k be the left end-point of the block of i. The blocks of k and j are merged, hence we set $C'(k) = C(k) \cup C(j)$ and $C'(\ell) = C(\ell)$ if $\ell \notin \{k, j\}$ is another left endpoint. Also, if $\ell, \ell' \notin \{k, j\}$ are other left endpoints, we set $D'(\ell, \ell') = D(\ell, \ell')$ if $\ell < \ell'$, $D'(\ell, k) = D(\ell, k) \cup D(\ell, j)$ if $\ell < k$, and $D'(k, \ell) = D(k, \ell) \cup D(j, \ell)$ if $j < \ell$. Finally, E' is the set of pairs (ℓ, ℓ') of left endpoints of distinct blocks such that either $\ell \neq j \neq \ell'$ and $(\ell, \ell') \in E$, or $\ell = k$ and $(j, \ell') \in E$, or $\ell < k = \ell'$ and $(\ell, j) \in E$.

$\oplus$ We define $(q_1, \Delta_1, C_1, D_1, E_1), (q_2, \Delta_2, C_2, D_2, E_2) \xrightarrow{\oplus} (q, \Delta, C, D, E)$ to be a transition if

- $q_1, q_2 \xrightarrow{\oplus} q$ is a transition of $\mathcal{A}^{K,M}_{\mathsf{valid}}$.
- The other components of are inherited: $\Delta = \Delta_1 \cup \Delta_2$, $C = C_1 \cup C_2$, $D = D_1 \cup D_2$ and $E = E_1 \cup E_2$.
- For all $i < k < j$ left endpoints in $q = (P, <, \to)$, we have $C(k) \cap D(i,j) = \emptyset$. This is the crucial condition which ensures that a timing constraint always refers to the last transition resetting the clock being checked. If some clock $x \in D(i,j)$ is reset in block i and checked in block j, it is not possible to insert a block k between blocks i and j if clock x is reset by a transition in block k.
- For all $(i,j) \in E$ and $(k,\ell) \in E$, if $i < k < j$ then $\ell < j$. This ensures that the push-pop edges are well-nested.

Accepting condition A state (q, Δ, C, D, E) is accepting if

- q is an accepting state of $\mathcal{A}^{K,M}_{\mathsf{valid}}$, hence it consists of a single block with left endpoint i and right endpoing j (possibly $i = j$),
- the source state of $\Delta(i)$ is an initial state of $\mathcal{S}$, and the target state of $\Delta(j)$ is a final state of $\mathcal{S}$, or there is an ε-path of micro-transitions from the target state of $\Delta(j)$ to some final state of $\mathcal{S}$.

If the underlying system is a timed automata, it is sufficient to store the maps C, D in the state, since there are no stack operations.

E Dense-Time Multi-stack Pushdown Automata

In this section, we consider the model of dense-timed multistack push down automata (dtMPDA). The reachability problem for untimed MPDA is already undecidable. Several restrictions have been studied on MPDA like bounded phase, bounded context, bounded scope and so on to regain decidability. We look at dtMPDA restricted to a bounded number of rounds.

A dtMPDA is a tuple $\mathcal{S} = (S, \Sigma, \Gamma, X, s_0, F, \Delta)$ where S is a finite set of states, Σ is a finite alphabet, Γ is the stack alphabet, s_0 is the initial state, X is a finite set of clocks, $F \subseteq S$ is a finite set of final states, and Δ is a finite set of transitions of the form $(s, \gamma, a, \mathsf{op}, R, s')$ 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 $I \in \mathcal{I}$, $R \subseteq X$ is the set of clocks reset, and op is a stack operation of one of the following kinds:

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

A sequence $\sigma = \mathsf{op}_1 \cdots \mathsf{op}_m$ of operations is a round if it can be decomposed in $\sigma = \sigma_1 \cdots \sigma_n$ where each factor σ_i is a possibly empty sequence of operations of the form nop, $\downarrow^i_c$, $\uparrow^i_{c \in I}$.

E.1 Simple TC-word semantics for dtMPDA

We define the semantics for dtMPDA in terms of simple TCWs. Let n denote the number of stacks. A simple TCW $\mathcal{V} = (P, \to, \lambda, \triangleright, \theta)$ is said to be generated or accepted by a dtMPDA $\mathcal{S}$ if there is an accepting abstract run $\rho = (s_0, \gamma_1, a_1, \mathsf{op}_1, R_1, s_1)\ (s_1, \gamma_2, a_2, \mathsf{op}_2, R_2, s_2) \cdots (s_{m-1}, \gamma_m, a_m, \mathsf{op}_m, R_m, s_m)$ of $\mathcal{S}$ such that $s_m \in F$ is a final state and

- the sequence of push-pop operations of any stack $1 \leq y \leq n$ is well-nested. In each prefix $\mathsf{op}_1 \cdots \mathsf{op}_o$ with $1 \leq o \leq m$, the number of pops of stack y is at most the number of pushes on stack y, and in the full sequence $\mathsf{op}_1 \cdots \mathsf{op}_m$ the number of pops equals the number of pushes.
- the sequence $\mathsf{op}_1 \cdots \mathsf{op}_m$ of stack operations has (at most) k-rounds.
- We have $P = P_0 \uplus P_1 \uplus \cdots \uplus P_m$ with $P_i \times P_j \subseteq \to^+$ for $0 \leq i < j \leq n$.
  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 simple TCW. op_i is an operation on some stack y. Intuitively, the transition δ_i is simulated by a sequence of micro-transitions

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

  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-transtions, corresponding to the reset of a new clock ζ and checking the constraint $\zeta = 0$ make sure that all micro-transtions in the sequence occur simultaneously. We have a point in P_i for each micro-transition (excluding the ε-micro-transitions between the $\delta_i^{x_j}$). Hence, P_i consists in a sequence
$$\ell_i \to \ell_i^1 \to \cdots \to \ell_i^{h_i} \to p_i \to r_i^1 \to \ldots \to r_i^{g_i} \to r_i$$
  where g_i is the number of timing constraints corresponding to clocks reset during transition i and checked afterwards. Similarly, h_i is the the number of timing constraints checked 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 being reset. So we let $R_0 = X$ and P_0 consists of a sequence
$$\ell_0 \to r_0^1 \to \ldots \to r_0^{g_0} \to r_0 \,.$$
- the relation for timing constraints can be partitionned as $\rhd = \biguplus_{1 \leq y \leq n} \rhd^y \uplus \biguplus_{x \in X \cup \{\zeta\}} \rhd^x$ where
  - $\rhd^\zeta = \{(\ell_i, r_i) \mid 0 \leq i \leq m\}$ and we set $\theta(\ell_i, r_i) = [0, 0]$ for all $0 \leq i \leq n$.
  - We have $p_i \rhd^y p_j$ if $\mathsf{op}_i = \downarrow_b^y$ is a push and $\mathsf{op}_j^y = \uparrow_{b \in I}^y$ is the matching pop for stack y (same number of pushes and pops of stack y in $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1}$), and we set $\theta(p_i, p_j) = I$.
  - for each $0 \leq i < j \leq n$ 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 $1 \leq s \leq g_i$ and $\theta(r_i^s, \ell_j^t) = I$. Therefore, every point ℓ_i^t with $1 \leq t \leq h_i$ is the target of a timing constraint. Moreover, every reset point r_i^s for $1 \leq s \leq g_i$ 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 $r_i^1 \lessdot \cdots \lessdot r_i^{g_i}$ are grouped by clocks (as suggested by the sequence of micro-transitions simulating δ_i): if $1 \leq s < u < t \leq g_i$ 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 request 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 $\mathcal{S}$. The language of $\mathcal{L}(\mathcal{S})$ is the set of *realizable* simple TCWs in $\mathsf{STCW}(\mathcal{S})$. Given a bound k on the number of rounds, we denote by $\mathsf{STCW}(\mathcal{S}, k)$ the set of simple TCWs generated by runs of $\mathcal{S}$ using at most k rounds. We let $\mathcal{L}(\mathcal{S}, k)$ be the corresponding language.

Given a dtMPDA $\mathcal{S}$, we show that all simple TCWs in $\mathsf{STCW}(\mathcal{S}, k)$ have bounded split-width. Actually, we will prove a slightly more general result. We first identify some properties satisfied by all simple TCWs generated by a dtMPDA, then we show that all simple TCWs satisfying these properties have bounded split-width.

Let $\mathcal{V} = (P, \to, \lambda, \rhd, \theta)$ be a simple TCW. Recall that $< \, = \, \to^+$ is the transitive closure of the successor relation. We say that $\mathcal{V}$ is k-round *well timed* with respect to a set of clocks Y and stacks $1 \leq s \leq n$ if the $\rhd$ relation for timing constraints can be partitionned as $\rhd = \biguplus_{1 \leq s \leq n} \rhd^s \uplus \biguplus_{x \in Y} \rhd^x$ where

(T_1') for each $1 \leq s \leq n$, the relation $\rhd^s$ corresponds to the matching push-pop events of stack s, hence it is well-nested: for all $i \rhd^s j$ and $i' \rhd^s j'$, if $i < i' < j$ then $i' < j' < j$, see Figure 4. Moreover, $\mathcal{V}$ consists of at most k rounds, i.e., we have $P = P_1 \uplus \cdots \uplus P_k$ with $P_i \times P_j \subseteq <$ for all $1 \leq i < j \leq k$. And each P_ℓ is a round, i.e., $P_\ell = P_\ell^1 \uplus \cdots \uplus P_\ell^n$ with $P_\ell^s \times P_\ell^t \subseteq <$ for $1 \leq s < t \leq n$ and push pop events of P_ℓ^s are all on stack s (for all $i \rhd^t j$ with $t \neq s$ we have $i, j \notin P_\ell^s$).

(T_2') An x-reset block is a maximal consecutive sequence $i_1 \lessdot \cdots \lessdot i_n$ of positions in the domain of the relation $\rhd^x$. 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, then $i < i' < j' < j$. 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 easy to check that the simple TCWs in $\mathsf{STCW}(\mathcal{S}, k)$ where $\mathcal{S}$ is a dtMPDA using set of clocks X are well-timed for clocks in $Y = X \cup \{\zeta\}$, i.e., satisfy the properties above. The first condition (T_1') is satisfied by $\mathsf{STCW}(\mathcal{S}, k)$ by definition. Consider (T_2'). Let $i \rhd^x j$ and $i' \rhd^x j'$ for some clock $x \in X$. If i, i' are points in the same x-reset block for some $x \in X$, then by construction of $\mathsf{STCW}(\mathcal{S})$, if $i < i'$ then $i' < j' < j$ which gives well nesting. Similarly, if $i < i'$ are points in different x-reset blocks, then by definition of $\mathsf{STCW}(\mathcal{S})$, we have $j < i'$. It is clear that the new clock ζ also satisfies (T_2').

Lemma 37. *A k-round well-timed simple* TCW *has split-width at most $(4nk + 4)(|Y| + 1)$, where n is the number of stacks.*

Again, we play the split-game between *Adam* and *Eve*. *Eve* should have a strategy to disconnect the word without introducing more than $(4nk + 4)(|Y| + 1)$ blocks. The strategy of *Eve* is as follows: Given the k-round word w, *Eve* first breaks this into two words. The first word only has stack 1 edges, and the second word has stack edges corresponding to stacks $2, \ldots, n$. The first word can now be dealt with as we did in the case of TPDA. *Eve* then breaks the second word into two words, the first of which has only stack 2 edges, while the second word has edges of stacks 3,4 $\ldots, n$, and so on. Finally, we obtain n split-STCW's, each having edges corresponding to only one stack. Once this is achieved, these words can be processed as was done in the case of TPDA. The only thing to calculate is the number of cuts required in isolating each word.

Obtaining a word containing only stack i edges Since we are dealing with k-round TCWs, we know that the stack operations follow a nice order : stacks $1, \ldots, n$ are operated in order k times. More precisely, by (T_1'), the set P of points of the simple TCW $\mathcal{V}$ can be partitionned in $P = P_1 \cup P_1' \cup \cdots \cup P_k \cup P_k'$ such that $P_1 < P_1' < \cdots < P_k < P_k'$ and $P_1 \cup \cdots \cup P_k$ contains all and only stack operations from stack 1. *Eve*'s strategy is to separate all these sets, i.e., cut just after $P_1, P_1', \ldots, P_k$. This results in $2k - 1$ cuts. This will not disconnect the word if there are edges with clock timing constraints across blocks, i.e., from P_i or P_i' to some other block on the right.

Consider the case when we have timing constraints on some clock x which is reset in some block P_i and checked in some block P_j' with $i \leq j$. All these timing constraints come from the last reset block R_x of clock x which lies in P_i. With two cuts, we detach the consecutive sequence of resets in R_x which are checked in P_j'. So with at most $2|Y|$ cuts, we detach all the resets that come from blocks P_i with $i \leq j$ and are checked in block P_j'. Similarly, with at most $2|Y|$ cuts, we detach all the resets that come from blocks P_i' with $i < j$ and are checked in block P_j.

In total, including the $2k - 1$ cuts used to separate the blocks $P_1, P_1', \ldots, P_k, P_k'$ we have used $(2|Y| + 1)(2k - 1) \leq 4k(|Y| + 1)$ cuts.

The split-TCW is now disconnected, and we obtain 2 words, one containing only stack 1 edges, and the other containing only stack $2, \ldots, n$ edges. Each word has at most $k|Y|$ holes after the disconnect: at most $|Y|$ holes in each P_i of the first word, and at most $|Y|$ holes in each P'_i of the second word. If $Y = \{x_1, \ldots, x_m\}$, then the first and second words look like

$$\overline{P}_1 \qquad H_{x_{g_1}} \ldots H_{x_{g_m}} \qquad \overline{P}_2 \qquad H_{x_{d_1}} \ldots H_{x_{d_m}} \qquad \ldots \qquad \overline{P}_k$$
$$G_{x_{i_1}} \ldots G_{x_{i_m}} \qquad \overline{P}'_1 \qquad G_{x_{h_1}} \ldots G_{x_{h_m}} \qquad \overline{P}'_2 \qquad \ldots \qquad G_{x_{j_1}} \ldots G_{x_{j_m}} \qquad \overline{P}'_k$$

Here, $G_{x_{i_j}}$ consists of the bunches of edges coming from the last x_{i_j}-reset block in P_1, while $H_{x_{g_j}}$ consists of the bunches of edges coming from the last x_{g_j}-reset block in P'_1 and so on. $\overline{P}_1$ is the word containing the holes which earlier contained $G_{x_{i_1}} \ldots G_{x_{i_m}}$ in P_1 and so on. Likewise, $\overline{P}'_1$ is the word containing the holes which earlier contained $H_{x_{g_1}} \ldots H_{x_{g_m}}$ in P'_1 and so on.

Starting with the second word having at most $k|Y|$ holes, we repeat the process to separate out stack 2 edges. This will result again in at most $4k(|Y| + 1)$ edges to be cut, and will result in two new words, each having at most $2k|Y|$ holes. Continuing this, after the last separation, when we isolate stack n edges, we will obtain two words each having at most $nk|Y|$ holes.

Thus, by cutting at most $4nk(|Y| + 1)$ edges, we can separate out the starting word into n words, each having only one kind of stack edge. Applying the TPDA game on each of these words, we obtain a split bound of $(4nk + 4)(|Y| + 1)$.

E.2 Tree Automata Construction for MultiStack and Complexity

Having established a bound on the split-width for dtMPDA restricted to k rounds, we now discuss the construction of a tree automaton that checks ***ValCoRe*** when the underlying system is a dtMPDA.

Given a dtMPDA $\mathcal{S} = (S, \Sigma, \Gamma, X, s_0, F, \Delta)$, we first construct a dtMPDA $\mathcal{S}'$ that only accepts runs using at most k-rounds. The tree automaton that checks ***ValCoRe*** is for this dtMPDA $\mathcal{S}'$. The idea behind constructing $\mathcal{S}'$ is to easily keep track of the k-rounds by remembering in the finite control of $\mathcal{S}'$, the current round number and context number. The initial states of $\mathcal{S}'$ is $(s_0, 1, 1)$. Here 1,1 signifies that we are in round 1, and context 1 in which operations on stack 1 are allowed without changing context. The states of $\mathcal{S}'$ are $\{(s, i, j) \mid 1 \le i \le k, 1 \le j \le n, s \in L\}$.

Assuming that $(s, \gamma, a, \mathsf{op}, R, s') \in \Delta$ is a transition of $\mathcal{S}$, then the transitions Δ' of $\mathcal{S}'$ are as follows:

1. $((s, i, j), \gamma, a, \mathsf{op}, R, (s', i, j)) \in \Delta'$ if op is one of nop, $\downarrow^j_c$ or $\uparrow^j_{c \in I}$,
2. $((s, i, j), \gamma, a, \mathsf{op}, R, (s', i, h)) \in \Delta'$ if $j < h$ and op is one of $\downarrow^h_c$ or $\uparrow^h_{c \in I}$,
3. $((s, i, j), \gamma, a, \mathsf{op}, R, (s', i+1, h)) \in \Delta'$ if $h < j$ and op is one of $\downarrow^h_c$ or $\uparrow^h_{c \in I}$.

The final states of $\mathcal{S}'$ are of the form $\{(s, i, j) \mid s \in F\}$. It can be shown easily that accepting runs of $\mathcal{S}'$ correspond to accepting k-round bounded runs of $\mathcal{S}$.

Now, given the dtMPDA $\mathcal{S}'$, we discuss the tree automaton that checks ***ValCoRe***. The validity and realizability checks (Val and Re parts) are as discussed in Appendices B and C. The only change pertains to the automaton that checks correctness of the underlying run. The tree automaton $\mathcal{A}^{K,M}_{\mathcal{S}}$ stores the set E^s of pairs (i, j) of left endpoints of *distinct* blocks such that there is at least one push-pop edge pertaining to stack s from block i to block j. Thus, instead of one set E as in the case of TPDA, we require n distinct sets to ensure well-nesting for each stack edge. Secondly, we need to ensure that the k-round property is satisfied. Rather than doing this at the tree automaton level, we have done it at the dtMPDA level itself, by checking this in $\mathcal{S}'$. This blows up the number of locations by nk, the number of stacks and rounds. Thus, the number of states of the tree automaton $\mathcal{A}^{K,M}_{\mathcal{S}}$ that checks correctness when the underlying system is a k-round dtMPDA is $(nk|\mathcal{S}|)^{\mathcal{O}(K^2)} \cdot 2^{\mathcal{O}(nK^2(|X|+1))}$, where $K = (4nk + 4)(|X| + 2)$.

Proposition 38. Let $\mathcal{S}$ be a k-round multistack timed automaton of size $|\mathcal{S}|$ (constants encoded in unary) with n stacks and set of clocks X. Then, we can build a tree automaton $\mathcal{A}^{K,M}_{\mathcal{S}}$ of size $(nk|\mathcal{S}|)^{\mathcal{O}(K^2)} \cdot 2^{\mathcal{O}(nK^2(|X|+1))}$ such that $\mathcal{L}(\mathcal{A}^{K,M}_{\mathcal{S}}) = \{\tau \in \mathcal{L}(\mathcal{A}^{K,M}_{\mathsf{valid}}) \mid [\![\tau]\!] \in \mathsf{STCW}(\mathcal{S}')\}$.