

Timed Systems through the Lens of Logic

S. Akshay, Paul Gastin, Vincent Juge, Shankara Narayanan Krishna

▶ **To cite this version:**

S. Akshay, Paul Gastin, Vincent Juge, Shankara Narayanan Krishna. Timed Systems through the Lens of Logic. 2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), Jun 2019, Vancouver, Canada. 10.1109/LICS.2019.8785684 . hal-03185736

HAL Id: hal-03185736
https://hal.science/hal-03185736

Submitted on 30 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL
archives-ouvertes.fr

Timed Systems through the Lens of Logic

S. Akshay, Paul Gastin, Vincent Juge, Shankara Narayanan Krishna

▶ To cite this version:

S. Akshay, Paul Gastin, Vincent Juge, Shankara Narayanan Krishna. Timed Systems through the Lens of Logic. 2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), Jun 2019, Vancouver, Canada. pp.1-13, 10.1109/LICS.2019.8785684 . hal-03185736

HAL Id: hal-03185736
https://hal.archives-ouvertes.fr/hal-03185736

Submitted on 30 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Timed Systems through the Lens of Logic

S. Akshay*, Paul Gastin†, Vincent Jugé‡ and Shankara Narayanan Krishna*
* IIT Bombay {akshayss, krishnas}@cse.iitb.ac.in
† LSV, ENS Paris-Saclay & CNRS, Université Paris-Saclay paul.gastin@lsv.fr
‡ LIGM, Université Paris-Est Marne-la-Vallée, CNRS vincent.juge@u-pem.fr

***Abstract*—In this paper, we analyze timed systems with data structures. We start by describing behaviors of timed systems using graphs with timing constraints. Such a graph is called realizable if we can assign time-stamps to nodes or events so that they are consistent with the timing constraints. The logical definability of several graph properties [20], [10] has been a challenging problem, and we show, using a highly non-trivial argument, that the realizability property for collections of graphs with strict timing constraints is logically definable in a class of propositional dynamic logic (EQ-ICPDL), which is strictly contained in MSO. Using this result, we propose a novel, algorithmically efficient and uniform proof technique for the analysis of timed systems enriched with auxiliary data structures, like stacks and queues. Our technique unravels new results (for emptiness checking as well as model checking) for timed systems with richer features than considered so far, while also recovering existing results.**

***Index Terms*—Timed systems, propositional dynamic logic, Logical definability, Efficient algorithms, graphs**

I. Introduction

The modeling and analysis of complex real-time systems is a challenging and important area, both from theoretical and practical points of view. The challenge often stems from the fact that such models have different sources of infinite behaviors, which makes them highly expressive but difficult to analyze. On one hand, the timing features engender complex constraints between events, which allow (or disallow) infinite sets of timed behaviors (over real numbers) satisfying these constraints. On the other hand, the auxiliary data structures such as multiple stacks allow a rich expressive power often leading to undecidable verification problems, even in the absence of time. Thus, each choice of combining these components of real-time and specific data structures leads to rich models whose analysis is complicated and often intractable.

The analysis of timed systems without any additional data structures has often been done using well-accepted models like timed automata [8], where clocks are real-valued variables that are reset and checked at guards. The classical approach to analyze such timed automata is by abstracting the real-timed system using the so-called region abstraction into a finite-state automaton preserving emptiness. Several variants and extensions of this basic model have been considered over the years, for instance using event-clocks [9] or diagonal constraints, or even by allowing (non-) deterministic updates of clocks. Subsequently, there has been a growing body of work [2], [1], [5], [6], [15], [16], [17], [18], [26] towards adding auxiliary data structures like stacks [28], [4], [3] or queues [3] to such timed automata. In all these, the techniques used to solve the emptiness problem were specific and tailor-made to the choice of the data structure, kind of constraints and updates that are allowed.

Our goal is to introduce a novel and uniform approach for reasoning about such timed systems which allow rich timing features along with several types of auxiliary data structures at the same time. This technique captures the behaviors of the underlying model as graphs (see [3]) and examines the logical definability of certain properties over these graphs.

We start by abstracting a run of a system, be it timed or not, as a sequence of instructions. When the system has a data structure d such as a stack, these instructions may write to d (denoted $w(d)$) or read from d ($r(d)$). The behavior is modeled as a linear graph (the sequence of instructions), with instruction labels and with additional data-structure edges matching writes with corresponding reads, as illustrated in Figure 1. When the system is timed, instructions may also reset clocks ($x := 0$), check guards ($x < 3$), etc. These timing instructions are recorded as additional labels in the linear graph without a priori being interpreted as edges, as shown on Figure 2 left. This allows to decouple the behavior of the underlying untimed system from the timing constraints that should be realized for the run to be feasible.

Our first contribution is to show that non-emptiness of a timed system $\mathcal{T}$ can be reduced to the satisfiability of a formula $\Phi_{\mathcal{T}}$ over such labeled linear graphs, which we call $\mathcal{T}$-graphs. A $\mathcal{T}$-graph G_τ obtained from a sequence of instructions τ, as depicted in Figure 2 (left), is a witness of non-emptiness of $\mathcal{T}$ if it satisfies three properties:
1) The sequence of instructions τ can be generated by $\mathcal{T}$. Since the system $\mathcal{T}$ is usually described with a finite automaton where transitions are labeled with instructions, $\mathcal{T}$ induces a *regular* language of instruction sequences which can easily be captured by (Φ_1) in our logic.
2) The data-structure edges should comply with the sequence of instructions. Intuitively, a node labeled with $w(d)$ (resp. $r(d)$) should have an outgoing (resp. incoming) d-edge. If the data structure d is a stack (resp. queue), then d-edges should be well-nested, i.e., satisfy the LIFO (resp. FIFO) policy. It is known that compliance with stack or queue data-structures can be expressed (Φ_2) in our logics [11].

Partly supported by UMI ReLaX, ANR project TickTac (ANR-18-CE40-0015), DST/INRIA CEFIPRA project EQuaVe and DST/INSPIRE faculty award [IFA12-MA-017].

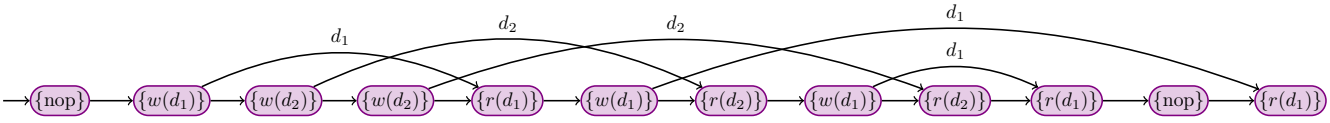


Fig. 1: Labeled linear graph G_σ of a sequence of instructions $\sigma = \text{nop}\ w(d_1)\ w(d_2)\ w(d_2)\ r(d_1)\ w(d_1)\ r(d_2)\ w(d_1)\ r(d_2)\ r(d_1)$ $\text{nop}\ r(d_1)$ from a system having two data structures (a stack d_1 and a queue d_2).

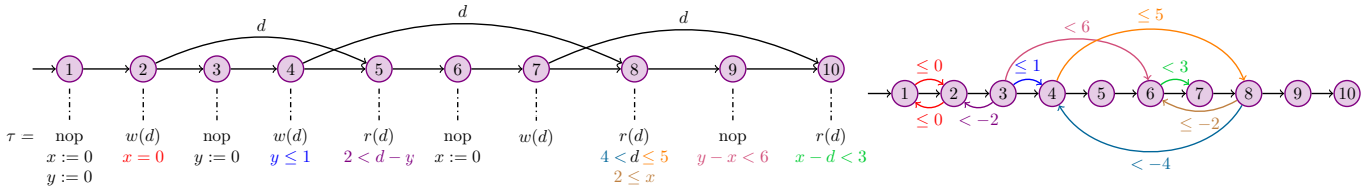


Fig. 2: Left: labeled linear graph G_τ obtained from a sequence of timed instructions τ. For readability, the nodes are numbered and their instruction labels are written below them. Right: the corresponding weighted graph $\mathcal{G}_\tau$.

3) The real-time constraints induced by the timing instructions should be *realizable*, i.e., it is possible to timestamp the nodes of G with some real numbers so that all timing constraints are satisfied. The second main contribution of this paper is to show that realizability can be expressed (Φ_3) in our logic.

We use a light-weight propositional dynamic logic called EQ-ICPDL for the logical definability. Writing formulae for our systems in EQ-ICPDL is rather intuitive and improves readability in several cases compared to the classical MSO. On a technical note, it is known that EQ-ICPDL is a strict fragment of MSO, and gives us a more tractable complexity than MSO (avoiding a non-elementary blowup).

We show that realizability can be expressed in EQ-ICPDL in two steps. First, from the $\mathcal{T}$-graph G_τ, we define a weighted graph $\mathcal{G}_\tau$ which retains only the timing constraints induced by the timed instruction sequence τ. For instance, in Figure 2, the $\mathcal{T}$-graph G_τ is on the left and the associated weighted graph $\mathcal{G}_\tau$ on the right. In $\mathcal{G}_\tau$, an edge from node i to node j labeled < 6 means that the difference $t(j) - t(i)$ between the timestamps assigned to i and j should be less than 6. We prove that the weighted graph $\mathcal{G}_\tau$ can be EQ-ICPDL-interpreted in the graph G_τ. This holds for all timing features that we consider. Second, we prove that realizability of weighted graphs is expressible in EQ-ICPDL, say with Φ_3'. Since weighted graphs $\mathcal{G}_\tau$ can be EQ-ICPDL-interpreted in $\mathcal{T}$-graphs G_τ, we can backward translate Φ_3' into some EQ-ICPDL formula Φ_3 expressing realizability over $\mathcal{T}$-graphs. Finally, non-emptiness of $\mathcal{T}$ is equivalent to satisfiability of $\Phi_\mathcal{T} = \Phi_1 \wedge \Phi_2 \wedge \Phi_3$.

Our logical characterization of realizability for weighted graphs is highly non-trivial. It is easier when the underlying system only has closed guards, but we go beyond this and prove that realizability is also definable in EQ-ICPDL in the presence of both open and closed guards. On the other hand, we show that, without the linear order, realizability is not definable in MSO. In fact, we show that this already holds for graphs with a partial order of width (i.e., size of the largest anti-chain) 2, thus proving a tight characterization.

Our third contribution is to show how the two results above can be combined with *existing* techniques to give an effective algorithm for checking emptiness of several classes of timed systems. First, observe that the above two contributions do not immediately imply that checking emptiness of the system is decidable, as satisfiability of EQ-ICPDL formulae over arbitrary collections of graphs is undecidable. This is expected, since, even in the untimed case, having a single queue or two stacks as data structures leads to undecidability of emptiness. However, we can now consider under-approximations, as classically done for untimed systems. One such under-approximation is to consider collections of $\mathcal{T}$-graphs that have a fixed bound on the tree-width. Such $\mathcal{T}$-graphs can now be interpreted into trees and we can use the fact that checking satisfiability for EQ-ICPDL (with bounded intersection width) over trees is decidable in EXPTIME. This gives us a matching EXPTIME algorithm for checking emptiness of timed systems whose graph behaviors have a bounded tree-width. Using this approach, we retrieve many known results on timed systems with data structures, and also obtain new results. Our approach captures with elan, the intricate flow and exchange of information between data structures and clocks, see Section V.

Related work. Our technique is orthogonal to the theory of timed systems via the region construction as well as to other related approaches. In the untimed setting, the closest work to ours is in [28], [4], where generic approaches for decidability via logic and tree-width have been developed for automata with data structures in the untimed setting. There have been several papers on the decidability of timed systems with a single stack: [12], [2] deal with specific timing constraints, while [16], [17] use the language of timed atoms to specify and analyze an orthogonal but powerful extension to timed registers. In [18], a NEXPTIME bound is shown in this setting by reduction to one-dimensional branching vector addition systems. However, all these works are restricted to a single stack, while we tackle several data structures including multiple stacks, queues. Many recent papers [17], [15], [1] consider

complex constraints between data structures and clocks. In these papers, there are time constraints between data structures d_1, d_2, between clocks, and also between a clock c and a data structure d. All of these can be modeled easily in our case, as can be seen in Section V.

Our work is also related to [5], [6], where the behaviors of timed systems with stacks are modeled as graphs having data-structure edges as well as time constraint edges. The presence of two types of edges necessitates a fresh proof for the the bound on tree-width for each kind of timing feature. On the contrary, we directly inherit the bound on tree-width established in the untimed setting. The other main difference is that [5], [6] directly build tree automata instead of going via logic. Using logic instead of directly building a tree automaton allows us to have a simpler higher level approach which is easier to write and less technical.

The logic we use builds on Propositional Dynamic Logic, a classical logic to reason about programs [23]. The extension with loop, intersection and converse was explored in [25], where complexity bounds were shown for satisfiability and model checking. We inherit these complexity bounds. However, to the best of our knowledge, this is the first time this logic has been used in the analysis of timed systems. Further, even with MSO logic (a strictly more powerful and well-known logic), the characterization of realizability in MSO over graphs of timed systems was open, as mentioned in [5]: we settle this problem in this paper.

Complete proofs of all results can be found in [7].

II. Preliminaries

Node- and edge-labeled graphs. Let Σ and Γ be two alphabets. Nodes will be labeled with Σ and edges with Γ. A (Σ, Γ)-labeled graph is a tuple $G = (V, E, \lambda)$ where V is a finite set of vertices, $\lambda \colon V \to 2^\Sigma$ labels vertices with (sets of) letters from Σ and $E \subseteq V \times \Gamma \times V$ is the set of labeled edges. A vertex may have 0, 1 or several labels from Σ. For $\gamma \in \Gamma$, we let $E_\gamma = \{(u, v) : (u, \gamma, v) \in E\}$ be the set of edges labeled γ. $\mathcal{G}(\Sigma, \Gamma)$ denotes the set of (Σ, Γ)-labeled graphs.

In this paper, graphs model behaviors of sequential systems. Hence, we have a special symbol succ in Γ to define the successor relation E_{succ} of a total order on V. We simply write $u \prec v$ instead of $(u, v) \in E_{\mathsf{succ}}$. We call these graphs *linear*; we let $\preceq = \prec^*$ be the linear order induced by $\prec$ and we note $\prec = \prec^+$ the strict order. The other edges E_γ, with $\gamma \in \Gamma \setminus \{\mathsf{succ}\}$, are used to model other useful relations in the graph, for instance the matching push-pop relation if we are interested in pushdown systems.

Propositional dynamic logic over labeled graphs. We define now the logic that we will use to specify properties of graphs. We use a variant of the propositional dynamic logic [23]. This logic is sufficiently expressive for our purposes and enjoys good complexity for the satisfiability problem, rather than the more expressive monadic second order logic (MSO) which has a much higher complexity. The logic $\mathsf{ICPDL}(\Sigma, \Gamma)$ is defined over Σ (often seen as propositional variables), and Γ (often seen as atomic programs).

Syntax: We have the following, with $p \in \Sigma$ and $\gamma \in \Gamma$:

$$\Phi ::= \mathsf{E}\,\sigma \;:\; \neg\Phi \;:\; \Phi \vee \Phi$$
$$\sigma ::= \top \;:\; p \;:\; \sigma \vee \sigma \;:\; \neg\sigma \;:\; \langle\pi\rangle\sigma \;:\; \mathsf{loop}(\pi)$$
$$\pi ::= \xrightarrow{\gamma} \;:\; \mathsf{test}\{\sigma\} \;:\; \pi + \pi \;:\; \pi \cdot \pi \;:\; \pi^* \;:\; \pi^{-1} \;:\; \pi \cap \pi$$

In ICPDL, C stands for converse (π^{-1}) and I for intersection ($\pi \cap \pi$). We also consider LCPDL which is the fragment with loop but without intersection, since it has better complexity, as stated in Theorem 2. We also write CPDL or PDL with the obvious meaning. In the syntax above, Φ are sentences and E is the existential node quantifier. The universal node quantifier $\mathsf{A}\,\sigma$ is written $\neg\mathsf{E}\neg\sigma$. Formulae σ are called *node* or *state* formulae and have one implicit free first-order variable, while formulae π are called *path* or *program* formulae and have two implicit free first-order variables, the endpoints of the path.

Semantics: Given a (Σ, Γ)-labeled graph $G = (V, E, \lambda)$, we can write the semantics of the formulae. The semantics of a *state formula* σ is a set $[\![\sigma]\!]_G \subseteq V$, while the semantics of a *path formula* π is a binary relation $[\![\pi]\!]_G \subseteq V^2$. Their definitions are mutually inductive. If the graph G is clear from the context, we omit subscripts and simply write $[\![\sigma]\!]$ and $[\![\pi]\!]$.

The base cases for path formulae are $[\![\xrightarrow{\gamma}]\!] = E_\gamma$ and $[\![\mathsf{test}\{\sigma\}]\!] = \{(v, v) : v \in [\![\sigma]\!]\}$. The operations $+, \cap, \cdot, ^*$ correspond to rational expression notations, interpreted respectively as union, intersection, concatenation and Kleene star of the respective relations. Finally, the converse is defined by $[\![\pi^{-1}]\!] = \{(u, v) : (v, u) \in [\![\pi]\!]\}$.

The base cases for state formulae are $[\![\top]\!] = V$ and $[\![p]\!] = \{v \in V : p \in \lambda(v)\}$, where $p \in \Sigma$. Disjunction and negation correspond to union and complement. We let $[\![\mathsf{loop}(\pi)]\!]$ consist of the vertices $v \in E$ from which there is a loop following path π, i.e., such that $(v, v) \in [\![\pi]\!]$. Similarly, we let $[\![\langle\pi\rangle\sigma]\!]$ consist of the vertices $u \in E$ from which it is possible to follow the path π and reach a vertex satisfying σ, i.e., $(u, v) \in [\![\pi]\!]$ for some $v \in [\![\sigma]\!]$. We often write $\langle\pi\rangle$ instead of $\langle\pi\rangle\top$. A sentence $\mathsf{E}\,\sigma$ states that there exists a vertex of G satisfying σ, i.e., $G \models \mathsf{E}\,\sigma$ if $[\![\sigma]\!]_G \neq \emptyset$. Disjunction and negation of sentences are as usual.

While ICPDL allows intersection, loop and converse, we also look at EQ-ICPDL where we allow existential quantification over new propositional variables in a similar spirit as in [27]. Thus, formulae of $\mathsf{EQ\text{-}ICPDL}(\Sigma, \Gamma)$ have the form $\Psi = \exists p_1, \ldots, p_n \; \Phi$ where $\mathsf{AP} = \{p_1, \ldots, p_n\}$ is disjoint from Σ and $\Phi \in \mathsf{ICPDL}(\Sigma \uplus \mathsf{AP}, \Gamma)$. The semantics is defined by $G = (V, E, \lambda) \models \exists p_1, \ldots, p_n \; \Phi$ if there exists $\lambda' \colon V \to 2^{\mathsf{AP}}$ such that $(G, \lambda') = (V, E, \lambda \cup \lambda') \models \Phi$. For formulae Ψ in $\mathsf{ICPDL}(\Sigma, \Gamma)$ or $\mathsf{EQ\text{-}ICPDL}(\Sigma, \Gamma)$, we let $L(\Psi) = \{G \in \mathcal{G}(\Sigma, \Gamma) : G \models \Psi\}$.

Example 1. *We illustrate the semantics of* $\mathsf{ICPDL}(\Sigma, \Gamma)$ *using Figure 3. We have a node- and edge-labeled graph, with node labels* $\Sigma = \{p, q, r, s\}$ *and edge labels* $\Gamma = \{d, e, f, \mathsf{succ}\}$. *In path formulae, we simply write* $\to$ *instead of* $\xrightarrow{\mathsf{succ}}$. *The formula* $\mathsf{E}\,\langle(\mathsf{test}\{p \vee q\} \cdot \to)^*\rangle r$ *evaluates to true on the given graph: the leftmost node is a witness. Likewise, the formula*

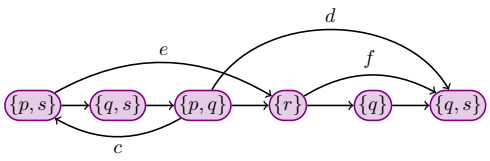


Fig. 3: A node- and edge-labeled graph.

$\neg \mathsf{E}\langle\rightarrow\rangle(p \wedge s)$ *is also true, since there are no nodes in the graph whose successors are labeled both* p *and* s*. Let* $\Delta = \Gamma \setminus \{\mathsf{succ}\}$*. The formula* $\mathsf{E} \bigvee_{(d,d')\in\Delta^2, d\neq d'} \mathsf{loop}(\xrightarrow{d} \cdot \xrightarrow{d'}^{-1})$ *is not true since all the non-successor edges are labeled by a unique symbol. Finally, the formula* $\mathsf{E}\langle \mathsf{test}\{s\} \cdot \xrightarrow{e} \cdot \mathsf{test}\{r\} \cdot \xrightarrow{f} \cdot \mathsf{test}\{s\} \cdot \xrightarrow{d}^{-1} \cdot \xrightarrow{c}\rangle p$ *is true, while* $\mathsf{E}\langle \mathsf{test}\{p\} \cdot \xrightarrow{d}\rangle r$ *is not.*

Satisfiability of propositional dynamic logic. The following definitions and results will be used in Section IV-C. Over arbitrary graphs, the satisfiability problem for PDL is undecidable. On the other hand, when we restrict to graphs of bounded tree-width, then the satisfiability problem becomes decidable with elementary complexity. We explain this now. Tree-width is a well-known measure for graphs [29]. We say that a labeled graph $G = (V, E, \lambda)$ has tree-width k if the underlying unlabeled graph has tree-width k. We will not need the formal definition of tree-width in this paper, so it is omitted. We denote by $\mathcal{G}^k(\Sigma, \Gamma)$ the graphs in $\mathcal{G}(\Sigma, \Gamma)$ having tree-width at most k.

Below is one of the main theorems that we use in this paper. It refers to the intersection width of an EQ-ICPDL formula, which is the maximum of the intersection widths of its path subformulae: the intersection width of path formulae is defined inductively by $\mathsf{iw}(\xrightarrow{\gamma}) = \mathsf{iw}(\mathsf{test}\{\sigma\}) = 1$, $\mathsf{iw}(\pi_1 + \pi_2) = \mathsf{iw}(\pi_1 \cdot \pi_2) = \max(\mathsf{iw}(\pi_1), \mathsf{iw}(\pi_2))$, $\mathsf{iw}(\pi^{-1}) = \mathsf{iw}(\pi^*) = \mathsf{iw}(\pi)$, and $\mathsf{iw}(\pi_1 \cap \pi_2) = \mathsf{iw}(\pi_1) + \mathsf{iw}(\pi_2)$. Hence, a formula in LCPDL has intersection width 1.

Theorem 2 (Satisfiability)**.** *Given* $k \geq 1$ *in unary and a formula* Ψ *in* $\mathsf{EQ\text{-}ICPDL}(\Sigma, \Gamma)$ *of intersection width bounded by a constant, checking whether* $G \models \Psi$ *for some* $G \in \mathcal{G}^k(\Sigma, \Gamma)$ *can be solved in* EXPTIME.

This is a consequence of a similar result over trees due to Göller, Lohrey and Lutz [25, Theorem 3.8]. Indeed, graphs of tree-width at most k can be represented by binary trees which are called k-terms. Moreover, for each formula $\Psi \in \mathsf{ICPDL}(\Sigma, \Gamma)$ we can construct an ICPDL formula $\overline{\Psi}^k$ of size $\mathcal{O}(k^2|\Psi|)$ over k-terms such that, for all k-terms τ, we have $\tau \models \overline{\Psi}^k$ iff $[\![\tau]\!] \models \Psi$, where $[\![\tau]\!]$ is the graph denoted by the k-term τ [11]. Hence, satisfiability of Ψ over $\mathcal{G}^k(\Sigma, \Gamma)$ is reduced to satisfiability of $\overline{\Psi}^k$ over k-terms.

Graph interpretation and backward translation. [21], [11]The following definitions and results will be used in Section IV-B. We consider two signatures (Σ, Γ) and (Σ', Γ'). Intuitively, a graph $G' \in \mathcal{G}(\Sigma', \Gamma')$ is interpreted in a graph $G \in \mathcal{G}(\Sigma, \Gamma)$ if we have formulae over the signature (Σ, Γ) which, when evaluated on G, express nodes, labels and edges of G'. In this paper, we use CPDL interpretations, which means that the formulae for the interpretation are in $\mathsf{CPDL}(\Sigma, \Gamma)$. Also, we only need interpretation when the graphs G and G' have the same set of nodes. In this simple case, an interpretation $\mathcal{I}$ is given by a tuple of state formulae $(\sigma_p)_{p\in\Sigma'}$ and a tuple of path formulae $(\pi_\gamma)_{p\in\Gamma'}$, all in $\mathsf{CPDL}(\Sigma, \Gamma)$. Now, we say that a graph $G' = (V, E', \lambda') \in \mathcal{G}(\Sigma', \Gamma')$ is $\mathcal{I}$-interpreted in the graph $G = (V, E, \lambda) \in \mathcal{G}(\Sigma, \Gamma)$ if, for all $u, v \in V$, all $p \in \Sigma'$ and all $\gamma \in \Gamma'$, we have $p \in \lambda'(u)$ iff $G, u \models \sigma_p$ and $(u, \gamma, v) \in E'$ iff $G, u, v \models \pi_\gamma$. In this case, we write $G' = \mathcal{I}(G)$.

Interpretations allow for a *backward translation* theorem: for each formula $\Psi' \in \mathsf{EQ\text{-}ICPDL}(\Sigma', \Gamma')$, we can construct a formula $\Psi \in \mathsf{EQ\text{-}ICPDL}(\Sigma, \Gamma)$ such that, for all graphs $G \in \mathcal{G}(\Sigma, \Gamma)$, we have $\mathcal{I}(G) \models \Psi'$ iff $G \models \Psi$. The formula Ψ is obtained from Ψ' by replacing the atomic state formulae p with σ_p (for $p \in \Sigma'$) and the atomic path formulae $\xrightarrow{\gamma}$ with π_γ (for $\gamma \in \Gamma'$). Hence, Ψ and Ψ' have same intersection width and $|\Psi| \leq |\Psi'| \cdot \max\{|\sigma_p|, |\pi_\gamma| : p \in \Sigma', \gamma \in \Gamma'\}$.

III. Logical definability of realizability

Weighted graphs. We consider linear weighted graphs where node labels are irrelevant, i.e., $\Sigma = \emptyset$, and edges are labeled with constraints of the form $< \alpha$ or $\leq \alpha$, where $\alpha \in \mathbb{Z}$, i.e., $\Gamma = \{\mathsf{succ}\} \cup (\{<, \leq\} \times \mathbb{Z})$. Since node labels are irrelevant, a linear weighted graph is simply denoted $G = (V, E)$. Often we use a maximal constant $M \in \mathbb{N}$ and let $\Gamma_M = \{\mathsf{succ}\} \cup (\{<, \leq\} \times \{-(M-1), \ldots, 0, \ldots, M-1\})$. A graph $G \in \mathcal{G}(\emptyset, \Gamma_M)$ is called M *weight-bounded*. If we only compare using $\leq$, i.e., if there are no edges of the form $(u, <, \alpha, v)$, then we say that the graph is *closed* or a graph with closed constraints. Otherwise, we call it a *mixed* weighted graph or a graph with mixed constraints.

Realizability. One important property of interest, which is the focus of this paper, is *realizability*. The property of realizability asks whether the constraints defined by the weights can be satisfied in a manner that is consistent with the order.

Definition 3. A weighted graph G is *realizable* if there exists a time-stamp map $\mathsf{ts} : V \to \mathbb{R}$ such that (i) all constraints are satisfied: $\forall (u, \triangleleft, \alpha, v) \in E$, $\mathsf{ts}(v) - \mathsf{ts}(u) \triangleleft \alpha$, and (ii) ts is monotone w.r.t. the linear order: $\forall u, v \in V$, if $u \preceq v$, then $\mathsf{ts}(u) \leq \mathsf{ts}(v)$.

If G is realizable via a map ts, then we say that ts is a *realization* of G. Note that the monotonicity could have been enforced by adding more constraint edges: when $u \prec v$ we could have added an edge $(v, \leq, 0, u)$. With these extra constraints, realizability corresponds to checking the feasibility of the difference constraints. This is a classical problem on graphs which amounts to checking the absence of a negative cycle (see [19] for more details). There are many algorithms to solve this problem, e.g., the Bellman-Ford shortest path algorithm. Finally, as a quick aside, note that if we have reflexive edges $(u, \triangleleft, \alpha, u) \in E$, checking realizability for these constraints is always vacuously true or false for all

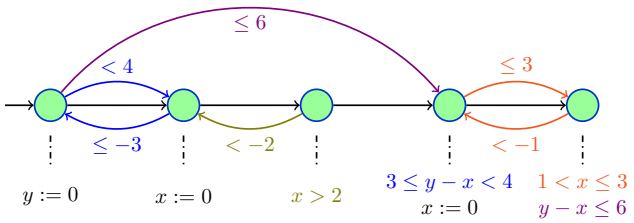


Fig. 4: A realizable linear weighted graph obtained from a sequence of instructions of a timed system. x, y are real-valued variables called *clocks*. $x := 0$ ($y := 0$) denotes reset instructions. Changing the last instruction to $x - y \leq 5$ gives a non-realizable weighted graph. The non-realizability follows from (i) there is a time elapse more than 5 between the first and third nodes, (ii) the time elapse is at most 5 between the first and fourth nodes, and (iii) time is monotone, hence there is at least zero time elapse between the third and fourth nodes. This gives a negative cycle between the first and fourth nodes.

possible time-stamps, and is easy. A realizable linear weighted graph obtained from a sequence of instructions of a timed system is depicted in Figure 4.

A. The first main result: logical definability of realizability

We are interested in properties of (possibly infinite) collections of such graphs, presented in a finite fashion. In particular, we wish to view graphs as being generated by an automaton, i.e., as behaviors of a system, and we wish to reason about this set of graphs. From this automata-theoretic viewpoint, a natural question to ask is whether the properties that we wish to reason about are definable in a certain logic. We focus on the specific property of realizability in weighted graphs and study its definability in EQ-ICPDL in our first main result below. In the next section, we will explain far-reaching consequences of our logical characterization, and in particular its application for checking emptiness of timed systems.

Theorem 4. *Realizability is* EQ-ICPDL *definable on the set of graphs $\mathcal{G}(\emptyset, \Gamma_M)$. The size of the formula is polynomial in M and its intersection width is 2.*

We prove the above theorem in two steps: in Subsection III-A1, we consider closed graphs and show that the logical definition is rather easy for them. Then, in Subsection III-A2, we consider graphs with mixed constraints.

Throughout the proof, given a linear weighted graph $G = (V, E)$ with $|V| = n$, we let $V = \{u_1, \ldots, u_n\}$ with $u_1 \prec u_2 \prec \cdots \prec u_n$. We start with a simple observation regarding the time-stamps witnessing realizability in weighted graphs. Given an M weight-bounded graph $G = (V, E)$, a mapping $\mathsf{ts}\colon V \to \mathbb{R}$ is said to be *slowly monotone* if $0 \leq \lfloor \mathsf{ts}(v) \rfloor - \lfloor \mathsf{ts}(u) \rfloor \leq M - 1$ for all $u \to v$.

Lemma 5. *A graph $G = (V, E)$ in $\mathcal{G}(\emptyset, \Gamma_M)$ is realizable iff there is a slowly monotone map $\mathsf{ts} : V \to \mathbb{R}$ that realizes G.*

Intuitively, if a realization of a graph G is not slowly monotone, then there must exist two consecutive points whose time-stamps are separated by more than $M-1$. But in this case there can be no forward edge (i.e., upper bound) that crosses this point, and hence the time difference between them can be reduced to any value larger than $M - 1$ without affecting realizability. We detail this proof, via an induction, in [7].

Next, we have a crucial definition on general weighted graphs. Given an M weight-bounded linear graph $G = (V, E)$, a *time-stamping modulo* M is a map $\mathsf{tsm}\colon V \to \mathbb{Z}_M = \{0, \ldots M - 1\}$. For all $u, v \in V$, we set $\mathsf{d}_{\mathsf{tsm}}(u, v) = \mathsf{tsm}(v) - \mathsf{tsm}(u) \mod M$. Further, (u, v) is said to be tsm-*big* if there exist $w_1, w_2 \in V$ such that $u \preceq w_1 \prec w_2 \preceq v$ and $\mathsf{d}_{\mathsf{tsm}}(u, w_1) + \mathsf{d}_{\mathsf{tsm}}(w_1, w_2) \geq M$. Observe that, if $v \preceq u$, then (u, v) cannot be tsm-big.

Definition 6. A time-stamping modulo M tsm is said to *weakly satisfy* $G = (V, E)$ if for all $e = (u, \triangleleft, \alpha, v) \in E$,
(a) if $u \preceq v$, then (u, v) is not tsm-big and $\mathsf{d}_{\mathsf{tsm}}(u, v) \leq \alpha$;
(b) if $v \prec u$ then (v, u) is tsm-big or $\mathsf{d}_{\mathsf{tsm}}(v, u) \geq -\alpha$.

Lemma 9 below shows that for linear weighted graphs, existence of such a map is a necessary condition for realizability. But first, we establish some useful facts. Recall that $V = \{u_1, \ldots, u_n\}$ with $u_1 \prec u_2 \prec \cdots \prec u_n$. For $i \leq j$, we also define $\mathsf{d}^+_{\mathsf{tsm}}(u_i, u_j) = \min\{M, \mathsf{d}_{\mathsf{tsm}}(u_i, u_{i+1}) + \cdots + \mathsf{d}_{\mathsf{tsm}}(u_{j-1}, u_j)\}$ and $\mathsf{d}^+_{\mathsf{tsm}}(u_j, u_i) = -\mathsf{d}^+_{\mathsf{tsm}}(u_i, u_j)$. Notice that we have $\mathsf{d}^+_{\mathsf{tsm}}(u_i, u_i) = 0$.

Claim 7. *Let $G = (V, E) \in \mathcal{G}(\emptyset, \Gamma_M)$ and let $\mathsf{ts}\colon V \to \mathbb{R}$ be a slowly monotone map (which need not satisfy the constraints of G). Define $\mathsf{tsm}\colon V \to \mathbb{Z}_M$ by $\mathsf{tsm}(v) = \lfloor \mathsf{ts}(v) \rfloor \mod M$ for all $v \in V$. Then, for all $u, v \in V$ such that $u \preceq v$, we have $\mathsf{d}^+_{\mathsf{tsm}}(u, v) = \min\{\lfloor \mathsf{ts}(v) \rfloor - \lfloor \mathsf{ts}(u) \rfloor, M\}$. Furthermore, we have $\mathsf{d}^+_{\mathsf{tsm}}(u, v) = M$ if (u, v) is* tsm*-big, and $\mathsf{d}^+_{\mathsf{tsm}}(u, v) = \mathsf{d}_{\mathsf{tsm}}(u, v)$ otherwise.*

Given that $|\alpha| < M$ for all edges $e = (u, \triangleleft, \alpha, v) \in E$, Claim 7 provides us with the following, alternative characterization of weak satisfiability. A formal proof of the above claim and of the lemma below can be found in [7].

Lemma 8. *A time-stamping modulo M* tsm *weakly satisfies the graph $G = (V, E)$ if and only if $\mathsf{d}^+_{\mathsf{tsm}}(u, v) \leq \alpha$ for all $(u, \triangleleft, \alpha, v) \in E$.*

Now, we obtain one direction of the characterization, which works both for closed and open constraints.

Lemma 9. *If $G \in \mathcal{G}(\emptyset, \Gamma_M)$ is realizable, then there exists a time-stamping modulo M that weakly satisfies G.*

Proof. Lemma 5 proves that there exists a slowly monotone time-stamping ts that satisfies the constraints G. We define $\mathsf{tsm}\colon V \to \mathbb{Z}_M$ by $\mathsf{tsm}(v) = \lfloor \mathsf{ts}(v) \rfloor \mod M$, and we show below that tsm weakly satisfies G.

Let $(u, \triangleleft, \alpha, v) \in E$. By Lemma 8, it is enough to show that $\mathsf{d}^+_{\mathsf{tsm}}(u, v) \leq \alpha$. According to Claim 7, distinguishing the cases $u \preceq v$ and $v \prec u$, we show easily that $\mathsf{d}^+_{\mathsf{tsm}}(u, v) \leq \lfloor \mathsf{ts}(v) \rfloor - \lfloor \mathsf{ts}(u) \rfloor$ or $\mathsf{d}^+_{\mathsf{tsm}}(u, v) = -M$. In the first case, it follows that $\mathsf{d}^+_{\mathsf{tsm}}(u, v) \leq \lfloor \mathsf{ts}(v) \rfloor - \lfloor \mathsf{ts}(u) \rfloor \leq \mathsf{ts}(v) - \lfloor \mathsf{ts}(u) \rfloor < \mathsf{ts}(v) - \mathsf{ts}(u) + 1 \leq \alpha + 1$, and in the second case, we also

have $\mathsf{d}^+_{\mathsf{tsm}}(u,v) = -M < \alpha + 1$. Hence, in both cases, we have $\mathsf{d}^+_{\mathsf{tsm}}(u,v) < \alpha + 1$. Observing that $\mathsf{d}^+_{\mathsf{tsm}}(u,v)$ and α are integers proves that $\mathsf{d}^+_{\mathsf{tsm}}(u,v) \leq \alpha$. □

The converse of the above lemma does not hold with mixed guards and this will be handled in the next subsection. However, for closed guards it yields the following characterization.

1) Characterizing realizability in closed graphs:

Lemma 10. *A closed graph $G = (V,E)$ in $\mathcal{G}(\emptyset, \Gamma_M)$ is realizable iff there exists a time-stamping modulo M that weakly satisfies G.*

Proof. One direction is Lemma 9. Conversely, suppose that $\mathsf{tsm}\colon V \to \mathbb{Z}_M$ is a time-stamping modulo M that weakly satisfies G. Then, the map $\mathsf{ts}\colon V \to \mathbb{N}$ defined inductively by $\mathsf{ts}(u_1) = 0$ and $\mathsf{ts}(u_{i+1}) = \mathsf{ts}(u_i) + \mathsf{d}_{\mathsf{tsm}}(u_i, u_{i+1})$ is a slowly monotone map.

Let $(u, \triangleleft, \alpha, v) \in E$. By Claim 7, distinguishing the cases $u \preceq v$ and $v \prec u$, we show easily that $\mathsf{d}^+_{\mathsf{tsm}}(u,v) \geq \lfloor \mathsf{ts}(v) \rfloor - \lfloor \mathsf{ts}(u) \rfloor$ or $\mathsf{d}^+_{\mathsf{tsm}}(u,v) = M$. Since tsm weakly satisfies G (i.e., $\mathsf{d}^+_{\mathsf{tsm}}(u,v) \leq \alpha$) and $M > \alpha$, the second case is impossible. It follows that $\mathsf{ts}(v) - \mathsf{ts}(u) = \lfloor \mathsf{ts}(v) \rfloor - \lfloor \mathsf{ts}(u) \rfloor \leq \mathsf{d}^+_{\mathsf{tsm}}(u,v) \leq \alpha$, which shows that ts satisfies the constraints of G. □

It remains to encode the characterization of Lemma 10 in EQ-ICPDL to obtain the logical definability of realizability for linear weighted graphs.

EQ-LCPDL *characterization:* We use existential quantification over atomic propositions $p_0, \ldots, p_{M-1}$ to guess the time-stamping modulo M. Intuitively, a node satisfies p_i iff its tsm value is i. So we define the formula $\exists p_0, \ldots, p_{M-1}\ \mathsf{Partition} \wedge \mathsf{Forward} \wedge \mathsf{Backward}$ where the auxiliary formulae are defined in Table I. The formula Partition states that every vertex satisfies exactly one p_i ($0 \leq i < M$).

For $0 \leq i, j < M$, let $\delta_M(i,j) = (j - i) \mod M$. We use a path formula to characterize pairs of vertices that are tsm-big: a pair (u,v) is tsm-big iff we can go from node u to node v following the path formula BigPath.

Since negation is not allowed at the level of path formulae, we provide another formula, SmallPath, to express that a pair (u,v) of vertices is not tsm-big. There are two cases, depending on whether $\mathsf{tsm}(u) \leq \mathsf{tsm}(v)$ or not. In both cases, $(u,v) \models \mathsf{SmallPath}_{i,j}$ iff $u \preceq v$, (u,v) is not tsm-big, $i = \mathsf{tsm}(u)$ and $j = \mathsf{tsm}(v)$.

Formulae Forward and Backward respectively state the two conditions in Definition 6. The constraint on $\preceq$-forward edges is stated using the loop operator of LCPDL. By excluding the existence of a loop following the path $\mathsf{BigPath} \cdot \xrightarrow{\leq \alpha}{}^{-1}$ we make sure that forward edges $(u,v) \in E_{\leq \alpha}$ are not tsm-big. Now, to ensure that forward edges $(u, \triangleleft, \alpha, v)$ satisfy $\mathsf{d}_{\mathsf{tsm}}(u,v) \leq \alpha$, we exclude the existence of a path violating this property, i.e., a loop following $\mathsf{test}\{p_i\} \cdot \xrightarrow{\leq \alpha} \cdot \mathsf{test}\{p_j\} \cdot (\rightarrow^{-1})^+$ with $\delta_M(i,j) > \alpha$.

2) A characterization with mixed guards: The characterization above is not sufficient when some of the constraints are strict, i.e., E contains edges of the form $(u, <, \alpha, v)$. It turns out that we need an additional condition to make sure that the fractional parts do not violate the realizability.

$$\mathsf{Partition} = \mathsf{A} \bigvee_{0 \leq i < M} [p_i \wedge \bigwedge_{j \neq i} \neg p_j]$$

$$\mathsf{BigPath} = \sum_{\substack{0 \leq i,j,k < M \\ \delta_M(i,j) + \delta_M(j,k) \geq M}} \mathsf{test}\{p_i\} \cdot \rightarrow^+ \cdot \mathsf{test}\{p_j\} \cdot \rightarrow^+ \cdot \mathsf{test}\{p_k\} \cdot \rightarrow^*$$

$$\mathsf{SmallPath}_{i,j} = \mathsf{test}\{p_i\} \cdot \Big(\sum_{i \leq k \leq \ell \leq j} \mathsf{test}\{p_k\} \cdot \rightarrow \cdot \mathsf{test}\{p_\ell\} \Big)^* \cdot \mathsf{test}\{p_j\} \text{ if } i \leq j$$

$$\mathsf{SmallPath}_{i,j} = \sum_{0 \leq \ell \leq j < i \leq k < M} \mathsf{SmallPath}_{i,k} \cdot \rightarrow \cdot \mathsf{SmallPath}_{\ell,j} \qquad \text{if } j < i$$

$$\mathsf{Forward} = \neg \mathsf{E} \bigvee_{-M < \alpha < M} \mathsf{loop}(\mathsf{BigPath} \cdot \xrightarrow{\leq \alpha}{}^{-1}) \wedge \neg \mathsf{E} \bigvee_{\substack{0 \leq i,j < M \\ \delta_M(i,j) > \alpha}} \mathsf{loop}(\mathsf{test}\{p_i\} \cdot \xrightarrow{\leq \alpha} \cdot \mathsf{test}\{p_j\} \cdot (\rightarrow^{-1})^+)$$

$$\mathsf{Backward} = \neg \mathsf{E} \bigvee_{\substack{-M < \alpha < M \\ 0 \leq i,j < M \\ \delta_M(i,j) < -\alpha}} \mathsf{loop}(\mathsf{SmallPath}_{i,j} \cdot \xrightarrow{\leq \alpha})$$

TABLE I: LCPDL for realizability of linear closed graphs

Definition 11. Given a graph $G = (V,E)$ and a time-stamping $\mathsf{tsm} : V \to \mathbb{Z}_M$ modulo M, we define two binary relations $\mathsf{geq}_{\mathsf{Fr}}$ and $\mathsf{gt}_{\mathsf{Fr}}$ on V:

- $(u,v) \in \mathsf{geq}_{\mathsf{Fr}}$ iff one of the following conditions hold:
  1) $u \prec v$, (u,v) is not tsm-big and $\mathsf{d}_{\mathsf{tsm}}(u,v) = \alpha$ for some edge $(u, \triangleleft, \alpha, v) \in E$;
  2) $v \prec u$, (v,u) is not tsm-big and $\mathsf{d}_{\mathsf{tsm}}(v,u) = -\alpha$ for some edge $(u, \triangleleft, \alpha, v) \in E$;
  3) $v \preccurlyeq u$ and $\mathsf{d}_{\mathsf{tsm}}(u,v) = 0$.
- $(u,v) \in \mathsf{gt}_{\mathsf{Fr}}$ iff one of the following conditions hold:
  1) $u \prec v$, (u,v) is not tsm-big and $\mathsf{d}_{\mathsf{tsm}}(u,v) = \alpha$ for some edge $(u, <, \alpha, v) \in E$;
  2) $v \prec u$, (v,u) is not tsm-big and $\mathsf{d}_{\mathsf{tsm}}(v,u) = -\alpha$ for some edge $(u, <, \alpha, v) \in E$.

Notice that $\mathsf{gt}_{\mathsf{Fr}} \subseteq \mathsf{geq}_{\mathsf{Fr}}$. The idea is that these relations give the ordering between the fractional parts. Thus, $(u,v) \in \mathsf{geq}_{\mathsf{Fr}}$ (resp. $\mathsf{gt}_{\mathsf{Fr}}$) means that the fractional part of $\mathsf{ts}(u)$ must be at least (resp. strictly greater than) the fractional part of $\mathsf{ts}(v)$. Once again, since $|\alpha| < M$ for all edges $(u, \triangleleft, \alpha, v) \in E$, Claim 7 provides an alternative characterization of the relations $\mathsf{geq}_{\mathsf{Fr}}$ and $\mathsf{gt}_{\mathsf{Fr}}$.

Lemma 12. *Consider graph $G = (V,E)$, $\mathsf{tsm} : V \to \mathbb{Z}_M$ modulo M and a pair (u,v) of vertices of G. Then,*

- $(u,v) \in \mathsf{geq}_{\mathsf{Fr}}$ *iff there exists an edge $(u, \triangleleft, \alpha, v) \in E$ such that $\mathsf{d}^+_{\mathsf{tsm}}(u,v) = \alpha$, or if $v \prec u$ and $\mathsf{d}^+_{\mathsf{tsm}}(u,v) = 0$;*
- $(u,v) \in \mathsf{gt}_{\mathsf{Fr}}$ *iff there exists an edge $(u, <, \alpha, v) \in E$ such that $\mathsf{d}^+_{\mathsf{tsm}}(u,v) = \alpha$.*

Lemma 13. *Let $G = (V,E)$ be an M weight-bounded graph with a linear order and mixed constraints. G is realizable iff there exists a time-stamping modulo M tsm such that (i) tsm weakly satisfies G and (ii) there do not exist $u, v \in V$ such*

that $(u,v) \in \mathsf{gt}_{\mathsf{Fr}}$ *and* $(v,u) \in \mathsf{geq}_{\mathsf{Fr}}{}^*$*, where* $\mathsf{geq}_{\mathsf{Fr}}{}^*$ *is the reflexive transitive closure of* $\mathsf{geq}_{\mathsf{Fr}}$.

Proof. In the forward direction, let G be realizable. Let $\mathsf{ts}\colon V \to \mathbb{R}$ be a slowly monotone map that realizes G, and let tsm be the time-stamping modulo M defined by $\mathsf{tsm}\colon v \to \lfloor \mathsf{ts}(v) \rfloor \mod M$. Lemma 9 proves that tsm weakly realizes G. We further claim that, if $(u,v) \in \mathsf{geq}_{\mathsf{Fr}}$, then $\{\mathsf{ts}(u)\} \geq \{\mathsf{ts}(v)\}$, and that, if $(u,v) \in \mathsf{gt}_{\mathsf{Fr}}$, then $\{\mathsf{ts}(u)\} > \{\mathsf{ts}(v)\}$. The proof is as follows.

- If $(u,v) \in \mathsf{geq}_{\mathsf{Fr}}$ because $v \prec u$ and $\mathsf{d}^+_{\mathsf{tsm}}(u,v) = 0$, then $\mathsf{ts}(v) \leq \mathsf{ts}(u)$ and $0 = \mathsf{d}^+_{\mathsf{tsm}}(u,v) = \min\{\lfloor \mathsf{ts}(v) \rfloor - \lfloor \mathsf{ts}(u) \rfloor, -M\}$. Hence, $\lfloor \mathsf{ts}(u) \rfloor = \lfloor \mathsf{ts}(v) \rfloor$, and therefore $\{\mathsf{ts}(v)\} \leq \{\mathsf{ts}(u)\}$.
- If $(u,v) \in \mathsf{geq}_{\mathsf{Fr}}$ because there exists an edge $(u, \lhd, \alpha, v) \in E$ such that $\mathsf{d}^+_{\mathsf{tsm}}(u,v) = \alpha$, then $-M < \alpha = \mathsf{d}^+_{\mathsf{tsm}}(u,v) < M$, and Claim 7 proves that $\alpha = \mathsf{d}^+_{\mathsf{tsm}}(u,v) = \lfloor \mathsf{ts}(v) \rfloor - \lfloor \mathsf{ts}(u) \rfloor$. It follows that $\{\mathsf{ts}(v)\} = \mathsf{ts}(v) - \lfloor \mathsf{ts}(v) \rfloor \leq \mathsf{ts}(u) + \alpha - \lfloor \mathsf{ts}(v) \rfloor = \mathsf{ts}(u) - \lfloor \mathsf{ts}(u) \rfloor = \{\mathsf{ts}(u)\}$.
- If $(u,v) \in \mathsf{gt}_{\mathsf{Fr}}$, then the same argument proves that $\alpha = \mathsf{d}^+_{\mathsf{tsm}}(u,v) = \lfloor \mathsf{ts}(v) \rfloor - \lfloor \mathsf{ts}(u) \rfloor$, and it follows that $\{\mathsf{ts}(v)\} = \mathsf{ts}(v) - \lfloor \mathsf{ts}(v) \rfloor < \mathsf{ts}(u) + \alpha - \lfloor \mathsf{ts}(v) \rfloor = \mathsf{ts}(u) - \lfloor \mathsf{ts}(u) \rfloor = \{\mathsf{ts}(u)\}$.

In the reverse direction, let $\mathsf{tsm}\colon V \to \mathbb{Z}_M$ be a time-stamping modulo M that weakly satisfies G and such that (ii) holds. As a direct consequence of (ii), every path in the graph $G_{\mathsf{geq}_{\mathsf{Fr}}} = (V, \mathsf{geq}_{\mathsf{Fr}})$ contains at most $|V|$ edges in $\mathsf{gt}_{\mathsf{Fr}}$. Indeed, otherwize two such edges would start from the same vertex, so that one edge would belong to a cycle of $G_{\mathsf{geq}_{\mathsf{Fr}}}$. Hence, for every vertex $v \in V$, we define the integer $\mathsf{ts}_1(v)$ as the largest number of edges in $\mathsf{gt}_{\mathsf{Fr}}$ that may be used by a path in $G_{\mathsf{geq}_{\mathsf{Fr}}}$ starting from v: observe that $0 \leq \mathsf{ts}_1(v) \leq |V|$.

By construction, for every pair (u,v) in $\mathsf{geq}_{\mathsf{Fr}}$, we have $\mathsf{ts}_1(u) \geq \mathsf{ts}_1(v)$, and we even have $\mathsf{ts}_1(u) > \mathsf{ts}_1(v)$ if $(u,v) \in \mathsf{gt}_{\mathsf{Fr}}$. Then, consider the map $\mathsf{ts}_0\colon V \to \mathbb{N}$ defined inductively by $\mathsf{ts}_0(u_1) = 0$ and $\mathsf{ts}_0(u_{i+1}) = \mathsf{ts}_0(u_i) + \mathsf{d}_{\mathsf{tsm}}(u_i, u_{i+1})$. The proof of Lemma 10 shows that ts_0 is a slowly monotone map and that $\mathsf{ts}_0(v) - \mathsf{ts}_0(u) \leq \alpha$ for all edges $(u, \lhd, \alpha, v) \in E$.

We prove now that the map $\mathsf{ts}\colon V \to \mathbb{R}$ defined by $\mathsf{ts}(v) = \mathsf{ts}_0(v) + \mathsf{ts}_1(v)/(|V|+1)$ is monotone. For all pairs (u,v),

- if $u \prec v$ and $(v,u) \in \mathsf{geq}_{\mathsf{Fr}}$, then $\mathsf{ts}(v) = \mathsf{ts}_0(v) + \mathsf{ts}_1(v)/(|V|+1) \geq \mathsf{ts}_1(u) + \mathsf{ts}_1(u)/(|V|+1) \geq \mathsf{ts}(u)$, because $\mathsf{ts}_0(v) \geq \mathsf{ts}_0(u)$ and $\mathsf{ts}_1(v) \geq \mathsf{ts}_1(u)$;
- if $u \prec v$ and $(v,u) \notin \mathsf{geq}_{\mathsf{Fr}}$, then $\mathsf{d}^+_{\mathsf{tsm}}(v,u) \neq 0$, and therefore $\mathsf{d}^+_{\mathsf{tsm}}(u,v) \geq 1$, which proves that $\mathsf{ts}(v) \geq \mathsf{ts}_0(v) = \mathsf{ts}_0(u) + \mathsf{d}^+_{\mathsf{tsm}}(u) \geq \mathsf{ts}_0(u) + 1 > \mathsf{ts}_0(u) + \mathsf{ts}_1(u)/(|V|+1) = \mathsf{ts}(u)$.

Then, we prove that ts satisfies the constraints of G. Indeed, for every edge $(u, \lhd, \alpha, v) \in E$,

- if $\mathsf{d}^+_{\mathsf{tsm}}(u,v) = \alpha$, then $(u,v) \in \mathsf{geq}_{\mathsf{Fr}}$, and therefore $\mathsf{ts}_1(v) \leq \mathsf{ts}_1(u)$; it follows that $\mathsf{ts}(v) = \mathsf{ts}_0(v) + \mathsf{ts}_1(v)/(|V|+1) \leq \mathsf{ts}_0(u) + \alpha + \mathsf{ts}_1(u)/(|V|+1) = \mathsf{ts}(u) + \alpha$;
- if $\mathsf{d}^+_{\mathsf{tsm}}(u,v) = \alpha$ and, furthermore, $\lhd = <$, then $(u,v) \in \mathsf{gt}_{\mathsf{Fr}}$, hence $\mathsf{ts}_1(v) < \mathsf{ts}_1(u)$; it follows that $\mathsf{ts}(v) = \mathsf{ts}_0(v) + \mathsf{ts}_1(v)/(|V|+1) < \mathsf{ts}_0(u) + \alpha + \mathsf{ts}_1(u)/(|V|+1) = \mathsf{ts}(u) + \alpha$;
- if $\mathsf{d}^+_{\mathsf{tsm}}(u,v) \neq \alpha$, then $\mathsf{d}^+_{\mathsf{tsm}}(u,v) \leq \alpha - 1$, since tsm weakly satisfies G; it follows that $\mathsf{ts}(v) = \mathsf{ts}_0(v) + \mathsf{ts}_1(v)/(|V|+1) < \mathsf{ts}_0(v) + 1 \leq \mathsf{ts}_0(u) + (\alpha - 1) + 1 \leq \mathsf{ts}(u) + \alpha$.

Consequently, in all cases, we have $\mathsf{ts}(v) - \mathsf{ts}(u) \lhd \alpha$, which completes the proof. □

$$\mathsf{geq}_{\mathsf{Fr}} = (\xrightarrow{\leq \alpha} + \xrightarrow{< \alpha}) \cap \Big(\sum_{\substack{0 \leq i,j < M \\ \delta_M(i,j) = \alpha}} \mathsf{SmallPath}_{i,j} + \sum_{\substack{0 \leq i,j < M \\ \delta_M(j,i) = -\alpha}} \mathsf{SmallPath}^{-1}_{j,i} \Big) + \sum_{i < M} \mathsf{test}\{p_i\} \cdot \to^{-1} \cdot \mathsf{test}\{p_i\}$$

$$\mathsf{gt}_{\mathsf{Fr}} = \xrightarrow{< \alpha} \cap \Big(\sum_{\substack{0 \leq i,j < M \\ \delta_M(i,j) = \alpha}} \mathsf{SmallPath}_{i,j} + \sum_{\substack{0 \leq i,j < M \\ \delta_M(j,i) = -\alpha}} \mathsf{SmallPath}^{-1}_{j,i} \Big)$$

TABLE II: ICPDL formulae for capturing strict guards

EQ-ICPDL *characterization:* As before, we use existentially quantified propositional variables $p_0, \ldots, p_{M-1}$ to guess the tsm values. To state weak-realizability, we use the formula $\mathsf{WRealizable} = \mathsf{Partition} \wedge \mathsf{Forward} \wedge \mathsf{Backward}$ where the subformulae have been defined in Table I. In addition, we have to check the absence of a cycle among the fractional parts, which contains at least one strict inequality and other, possibly non-strict, inequalities. By Lemma 13, this suffices to ensure realizability. To capture the ordering among the fractional parts, we use two EQ-ICPDL formulae, $\mathsf{gt}_{\mathsf{Fr}}$ and $\mathsf{geq}_{\mathsf{Fr}}$ respectively for the strict and non-strict parts, formally defined in Table II. The EQ-ICPDL formula Realizable is then:

$$\exists p_0, \ldots p_{M-1}\ \mathsf{WRealizable} \wedge \neg \mathsf{E}\,\mathsf{loop}(\mathsf{gt}_{\mathsf{Fr}} \cdot \mathsf{geq}_{\mathsf{Fr}}{}^*)$$

The intersection width of $\mathsf{gt}_{\mathsf{Fr}}$ and $\mathsf{geq}_{\mathsf{Fr}}$ is 2. Hence, the intersection width of Realizable is also 2. This completes the proof of Theorem 4.

B. Realizability is beyond logical definability in general

Above, we have seen the EQ-ICPDL definability of realizability for linear weighted graphs. In the absence of a linear order, this is no longer true, even if one uses the strictly more expressive MSO logic (an easy example is the property of connectivity which separates EQ-ICPDL from MSO).

Theorem 14. *The property of realizability is not definable in* MSO *for weighted graphs without the linear order.*

We sketch the proof idea here, and leave the technical details to [7]. We consider a family of word structures over $\{a,b\}$ of the form a^*b^* and define an MSO transduction that gives rise to a family of weighted graphs. For a word $a^n b^m$ in the domain, the transduction gives rise to a weighted graph with $n+m$ nodes, with edges from nodes i to $i+1$ (with $1 \leq i \leq n-1$) having weight 1, and edges from node i to $i+1$ (with $n+1 \leq i \leq n+m-1$) having weight -1. Edges of weight 0 go from the node n to the node $n+1$, and from the node $n+m$ to the node 1. The constructed weighted graph has width

2, and it is realizable iff $n \geq m$. If realizability were MSO-definable, then using backwards translation theorem [21], one would obtain a regular language as the pre-image of those realizable graphs. This is not the case with $\{a^n b^m : n \geq m\}$.

IV. Analyzing timed systems with data structures

In this section, we develop a generic technique to analyze timed systems with auxiliary data structures. We start with untimed systems with data structures.

A. Capturing data structure operations as graphs

Let us fix a finite set of data structures DS. Each data structure $d \in \mathsf{DS}$ can be operated via two instructions, either a *write* that writes to the data structure, denoted $w(d)$, or a *read* instruction that reads from the data structure, denoted $r(d)$. The set of instructions from DS is denoted $\Sigma^{\mathsf{DS}} = \{r(d), w(d) : d \in \mathsf{DS}\} \cup \{\mathrm{nop}\}$, where nop is a special operation that does not access the data-structures. For simplicity and ease of exposition, we restrict each $d \in \mathsf{DS}$ to be a stack or a queue. However, the approach described here can be adapted to other structures (such as bags) with minor modifications. When $d \in \mathsf{DS}$ is a stack, $r(d)$ is the pop operation and $w(d)$ is the push operation on stack d. Similarly, if d is a queue, $r(d)$ is the dequeue operation, while $w(d)$ is the enqueue operation on queue d.

A sequence of operations from Σ^{DS} abstracts a run of a system with these data structures. We can then define the system as a generator of (possibly infinitely many) sequences of operations. The mechanism for generating this sequence of operations can be some machine (an automaton), or can be specified by regular expressions. We do not dwell on this detail here, and instead define *a system $\mathcal{S}$ with data structures* as a regular language of sequences of operations over Σ^{DS}. Without loss of generality, we assume that all sequences will start with nop. It is easy to see that standard models such as (multi)pushdown automata, (multi)queue automata, multiset automata and so on generate regular languages of sequences of such operations.

A sequence σ of operations over Σ^{DS} is said to be *valid* if, for every prefix σ' of σ and for every data structure $d \in \mathsf{DS}$, the number of reads $r(d)$ in σ' is at most the number of writes $w(d)$ in σ', and the number of reads and writes in σ are equal. For a system $\mathcal{S}$, we are only interested in *valid* sequences generated by $\mathcal{S}$, and we denote this set by $L(\mathcal{S})$. For instance, a valid behavior of a pushdown system cannot read/pop from a stack before writing/pushing to it. Let $\Gamma^{\mathsf{DS}} = \mathsf{DS} \cup \{\mathsf{succ}\}$. We associate, to any valid sequence σ of operations over Σ^{DS}, a $(\Sigma^{\mathsf{DS}}, \Gamma^{\mathsf{DS}})$ linear graph G_σ.

Definition 15. Let $\sigma = \sigma_1 \dots \sigma_n$ be a valid sequence of operations over Σ^{DS}. We define its $(\Sigma^{\mathsf{DS}}, \Gamma^{\mathsf{DS}})$-graph as $G_\sigma = (V, E, \lambda)$, where $V = \{1, \dots n\}$ and

1) for $1 \leq i \leq n$, $\lambda(i) = \{\sigma_i\}$, and, for $1 \leq i < n$, $i \xrightarrow{\mathsf{succ}} i+1$,
2) $\sigma_i = w(d)$ ($r(d)$) iff there is an outgoing (incoming) edge in E labeled d from (to) i.
3) for each stack (queue) d, edges labeled d satisfy the LIFO (FIFO) property.

As an example, let σ be a sequence of operations from $\mathsf{DS} = \{d_1, d_2\}$, where d_1 is a stack and d_2 is a queue. The graph G_σ corresponding to σ is depicted in Figure 1, where the node labels are exactly the singleton sets of operations $w(d)$ and $r(d)$, for $d \in \{d_1, d_2\}$. We remark that this graph depends crucially on the interpretation of the data structure, as a stack or a queue. Notice that the edges labeled d_1 respect the stack discipline (well-nesting), while the edges labeled d_2 respect FIFO. For a fixed DS, we assume the interpretation of each data structure to be fixed and simply write G_σ.

Given a $(\Sigma, \Gamma^{\mathsf{DS}})$-graph $G = (V, E, \lambda)$, we define its projection $\pi(G)$ as the $(\emptyset, \Gamma^{\mathsf{DS}})$-graph obtained by removing the node labels: $\pi(G) = (V, E)$.

Theorem 16 ([11])**.** *Let $\mathcal{S}$ be a system with data structures from* DS*. We can construct an* EQ-LCPDL$(\emptyset, \Gamma^{\mathsf{DS}})$ *formula $\psi_{\mathcal{S}}$ such that, for all $(\emptyset, \Gamma^{\mathsf{DS}})$-graphs G, $G \models \psi_{\mathcal{S}}$ iff $G = \pi(G_\sigma)$ for some $\sigma \in L(\mathcal{S})$.*

The classical *non-emptiness problem* for a system $\mathcal{S}$ with data structures can be formulated as whether $L(\mathcal{S}) \neq \emptyset$.

Corollary 17. *For system $\mathcal{S}$, $\psi_{\mathcal{S}}$ is satisfiable iff $L(\mathcal{S}) \neq \emptyset$.*

This corollary, along with Theorem 2, and using known bounds on tree-width, provides a "uniform" proof for the decidability of checking non-emptiness for a variety of untimed systems including (multi)pushdown and (multi)queue systems with bounded contexts, scope, or phases in a sequential setting. In many cases, the complexity obtained matches the best known bounds. We extend this approach uniformly to timed systems, using the realizability proof of Section III.

B. Combining timing and data structures

While combining time constraints and data structures, we cannot directly rely on the formula for realizability from Section III in the approach outlined above. The vocabulary of graphs obtained from systems having time constraints and data structures might differ from the (weighted) $(\emptyset, \Gamma^M)$-graphs of Section III and the (unweighted) $(\Sigma, \Gamma^{\mathsf{DS}})$-graphs above, where $\Sigma = \emptyset$ or $\Sigma = \Sigma^{\mathsf{DS}}$. The crucial observation is that, for a large class of timing constraints and data structures that we are interested in, it turns out that the former weighted graphs can be interpreted in the latter unweighted graphs, paving the way to extend the approach for systems having both time constraints and data structures. We now detail this intuition.

1) Timing instructions: In a timed system with data structures, the sequence of instructions generated by the system includes (i) checking time constraints on clocks (encoded as operations on clocks), (ii) checking time constraints on data structures, and (iii) mixing operations on clocks and data structures. Recall that we already have a fixed set of data structures DS consisting of stacks and queues. To be concrete, we also fix a representative set of timing features.

We fix a finite set Clocks of real-valued "clock" variables and a maximal constant $M \in \mathbb{N}$. We also fix notations $\bowtie \in \{\leq, <, =, >, \geq\}$, $\beta \in [0, M) \cap \mathbb{N}$ and use letters $x, y, x_1, \dots$ for clock variables. Atomic timing instructions are as follows:

1) for $x \in \mathsf{Clocks}$, $x{:=}0$ represents *clock resets*, while $x \bowtie \beta$ represent *guards* or *clock constraints*;
2) for $d \in \mathsf{DS}$, $d \bowtie \beta$ represents an *age constraint* checking the "age" of the message read;
3) for $d \in \mathsf{DS}$ and $x, y \in \mathsf{Clocks}$, $(x-y) \bowtie \beta$, $(d-x) \bowtie \beta$ and $(x-d) \bowtie \beta$ represent *diagonal constraints*. The latter two capture mixing clock variables and data structures.

Thus, we define a set of instructions $\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}$ which contains Σ^{DS} with the atomic timing instructions described above. Without loss of generality, we only consider sequences of instruction sets (also called *sequences of instructions* for simplicity) from $\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}$ starting with the set $\{\mathrm{nop}\} \cup \{x{:=}\,0 : x \in \mathsf{Clocks}\}$, i.e., which resets all clocks at start-up. A sequence τ of such instructions is shown in Figure 2. We associate to every such sequence τ a sequence of untimed instructions σ_τ, obtained by ignoring the atomic timing instructions. Now we say τ is valid if σ_τ is valid. Then, for every valid τ, we can immediately associate a $(\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}, \Gamma^{\mathsf{DS}})$-labeled linear graph G_τ by considering G_{σ_τ} and enriching its node labels with the timing instructions.

We define a timed system with data structures $\mathcal{T}$ as a regular language of sequences of instructions over $\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}$. It is easy to see that classical models, such as timed automata, (multi-stack) timed pushdown automata or timed automata with gap order constraints, can be modeled in this formalism. The set of valid sequences generated by $\mathcal{T}$ is denoted $L(\mathcal{T})$. Now, a valid sequence of instructions $\tau = \tau_1 \dots \tau_n$ over $\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}$ is said to be *timed feasible* or just *feasible* if there exists a time-stamping $\mathsf{ts} : \{1, \dots, n\} \to \mathbb{R}^{\geq 0}$ such that all timing constraints engendered by the timing instructions are satisfied. That is, for $\bowtie \in \{\leq, <, =, >, \geq\}$ and $\beta \in \mathbb{N}$:

(C_1) For every guard of the form $x \bowtie \beta$ at position i, if the last reset instruction of the clock x in τ before i was at position j, then $\mathsf{ts}(i) - \mathsf{ts}(j) \bowtie \beta$.

(C_2) For every age constraint of the form $d \bowtie \beta$ at position i, we have an edge $j \xrightarrow{d} i$ in G_τ (which implies $w(d) \in \lambda(j)$ and $r(d) \in \lambda(i)$), and $\mathsf{ts}(i) - \mathsf{ts}(j) \bowtie \beta$.

(C_3) For every diagonal constraint of the form $x - y \bowtie \beta$ at position i, if j and k are the last resets of clocks x and y respectively, then $\mathsf{ts}(k) - \mathsf{ts}(j) \bowtie \beta$.

(C_4) We can similarly define diagonal constraints between clocks and data structures.

Thus, the *non-emptiness problem* for the timed system $\mathcal{T}$ is to check whether there exists a feasible $\tau \in L(\mathcal{T})$.

2) From timing instructions to weighted graphs: We reduce checking non-emptiness of $\mathcal{T}$ to checking satisfiability of an EQ-ICPDL formula over $(\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}, \Gamma^{\mathsf{DS}})$-graphs. Towards this, we first define the weighted graph $\mathcal{G}_\tau$ corresponding to a valid sequence of instructions τ of $\mathcal{T}$ in a natural manner. We extend from Section III, where all timing instructions were simply clock constraints and resets of clocks i.e., corresponding to (C_1) and (C_3) above. In Figure 2, the check of $x = 0$ on node 2 gives two bidirectional weighted edges in the weighted graph $\mathcal{G}_\tau$ depicted on the right, between the last reset point of x and node 2. Similarly, instruction $y \leq 1$ at node 4 gives rise to the forward edge labeled ≤ 1 between last reset of y and node 4. For diagonal constraints (C_3), the edge obtained is between the last reset points. E.g, $y - x < 6$ at node 9 yields the weighted edge from node 3 to node 6 (last resets of clocks y and x).

This construction easily lifts to (C_2) and (C_4) as well. For (C_2), we just observe that each age constraint engenders edges between the source write and target read of that data structure edge. E.g., in Figure 2, the age constraint $4 < d \leq 5$ at node 8 yields two weighted edges between the source of the data structure edge, i.e., node 4 and target, node 8. The upper bound is captured by the forward edge while the lower bound by the backward edge. Similarly the constraint $2 < d - y$ at node 5 yields the backward edge from node 3 (the last reset of clock y) to node 2 (the source of the data structure edge reaching node 5) labeled < -2 (as it is a lower bound constraint).

The main property about the weighted graph is that it captures feasibility of a sequence of instructions as realizability.

Lemma 18. *A valid sequence of instructions τ over $\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}$ is feasible iff $\mathcal{G}_\tau$ is realizable.*

3) Interpreting weighted graphs in unweighted graphs: From the above discussion, given a timed system $\mathcal{T}$, for each valid τ of $\mathcal{T}$, we have a weighted graph $\mathcal{G}_\tau$. A significant contribution of this paper, of possible independent interest, is the following proposition which relates these weighted graphs with unweighted $(\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}, \Gamma^{\mathsf{DS}})$-graphs obtained from τ. Proposition 19 allows us to logically interpret weighted graphs into unweighted ones and, therefore, to decouple the data structure and process edges from the timing constraints.

Proposition 19. *Let τ be a valid sequence of instructions over $\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}$. Then the weighted graph $\mathcal{G}_\tau$ can be* CPDL*-interpreted in the $(\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}, \Gamma^{\mathsf{DS}})$-graph G_τ.*

Proof. Given a valid sequence of instructions τ over $\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}$, let M be the maximal constant appearing in these instructions. We saw in the previous subsection that the weighted graph $\mathcal{G}_\tau = (V, E)$ has successor edges, and weighted edges arising from constraints of type (C_1–C_4). First, we observe that successor edges in $\mathcal{G}_\tau$ are already present as successor edges in G_τ. For weighted edges, let $\triangleleft \in \{<, \leq\}$, and $c \in [0, M) \cap \mathbb{N}$. We assume that equality constraints such as $x = c$ have been replaced by the conjunction of $x \leq c$ and $c \leq x$. For a clock $x \in \mathsf{Clocks}$, we define the path formula
$\mathsf{Reset}_x = \to^{-1} \cdot (\mathsf{test}\{\neg(x := 0)\} \cdot \to^{-1})^* \cdot \mathsf{test}\{(x := 0)\}$
which moves backwards along successor edges up to the last reset of clock x. Then, towards the interpretation of forward edges weighted with $\triangleleft c$, we define the path formula $\Pi_{\triangleleft c}$ as

$$\sum_{x \in \mathsf{Clocks}} \mathsf{Reset}_x^{-1} \cdot \mathsf{test}\{x \triangleleft c\} \qquad (\mathsf{C}_1)$$

$$+ \sum_{d \in \mathsf{DS}} \xrightarrow{d} \cdot \mathsf{test}\{d \triangleleft c\} \qquad (\mathsf{C}_2)$$

$$+ \sum_{x,y \in \mathsf{Clocks}} \mathsf{Reset}_x^{-1} \cdot \mathsf{test}\{x - y \triangleleft c\} \cdot \mathsf{Reset}_y \qquad (\mathsf{C}_3)$$

$$+ \sum_{\substack{x \in \mathsf{Clocks} \\ d \in \mathsf{DS}}} \mathsf{Reset}_x^{-1} \cdot \mathsf{test}\{x - d \triangleleft c\} \cdot \xrightarrow{d}^{-1} + \xrightarrow{d} \cdot \mathsf{test}\{d - x \triangleleft c\} \cdot \mathsf{Reset}_x \qquad (\mathsf{C}_4)$$

Then, for all $u, v \in V$ and $c > 0$ (we will discuss the case $c = 0$ below), we have $(u, \lhd, c, v) \in E$ iff $(G_\tau, u, v) \models \Pi_{\lhd c}$. The four types of *upper* constraints defined in (C_1–C_4) are described by the respective path formulae (C_1–C_4) in $\Pi_{\lhd c}$. As an example, if we refer to the i^{th} node of G_τ(and $\mathcal{G}_\tau$) as u_i in Figure 2, we have the edge $(u_3, \lhd, 6, u_6)$ in $\mathcal{G}_\tau$ because $(G_\tau, u_3, u_6) \models \mathsf{Reset}_y^{-1} \cdot \mathsf{test}\{y - x \lhd 6\} \cdot \mathsf{Reset}_x$. Similarly, the edge $(u_6, <, 3, u_7)$ is present in $\mathcal{G}_\tau$ since $(G_\tau, u_6, u_7) \models \mathsf{Reset}_x^{-1} \cdot \mathsf{test}\{x - d < 3\} \cdot \xrightarrow{d}^{-1}$. Notice that in Reset_x, we walk backward to the *first* node labeled $x := 0$, while, in C_2 and C_4, for checking the age of a data structure, it is sufficient to check the existence of a data structure backward edge from the point where the age is checked.

Similarly, towards the interpretation of backward edges weighted with $\lhd -c$, we define the path formula $\Pi_{\lhd -c}$ as

$$\sum_{x \in \mathsf{Clocks}} \mathsf{test}\{c \lhd x\} \cdot \mathsf{Reset}_x \qquad (\mathsf{C}_1)$$

$$+ \sum_{d \in \mathsf{DS}} \mathsf{test}\{c \lhd d\} \cdot \xrightarrow{d}^{-1} \qquad (\mathsf{C}_2)$$

$$+ \sum_{x,y \in \mathsf{Clocks}} \mathsf{Reset}_y^{-1} \cdot \mathsf{test}\{c \lhd x - y\} \cdot \mathsf{Reset}_x \qquad (\mathsf{C}_3)$$

$$+ \sum_{\substack{x \in \mathsf{Clocks} \\ d \in \mathsf{DS}}} \mathsf{Reset}_x^{-1} \cdot \mathsf{test}\{c \lhd d - x\} \cdot \xrightarrow{d}^{-1} \qquad (\mathsf{C}_4)$$
$$+ \xrightarrow{d} \cdot \mathsf{test}\{c \lhd x - d\} \cdot \mathsf{Reset}_x$$

Then, for all $u, v \in V$ and $c > 0$, we have $(u, \lhd, -c, v) \in E$ iff $(G_\tau, u, v) \models \Pi_{\lhd -c}$. Again, the four types of *lower* constraints defined in (C_1–C_4) are described by the respective path formulae (C_1–C_4) in $\Pi_{\lhd -c}$.

Now, when $c = 0$, an edge weighted $\lhd 0$ may arise from an upper constraint such has $x \lhd 0$ or a lower constraint such as $0 \lhd x$. Therefore, for all $u, v \in V$, we have $(u, \lhd, 0, v) \in E$ iff $(G_\tau, u, v) \models \Pi_{\lhd 0} + \Pi_{\lhd -0}$.

The size of $\Pi_{\lhd \alpha}$ is $\mathcal{O}(|\mathsf{Clocks}|^2 + |\mathsf{DS}| + |\mathsf{Clocks}||DS|)$.

Thus we have described how each edge of the weighted graph $\mathcal{G}_\tau$ can be interpreted in the $(\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}, \Gamma^{\mathsf{DS}})$-graph G_τ by an CPDL-formula, of size $\mathcal{O}(|\mathsf{Clocks}|^2 + |\mathsf{DS}| + |\mathsf{Clocks}||DS|)$, which completes the proof of this proposition. □

Thus, any formula over weighted graphs can be translated into an "*equivalent*" formula over $(\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}, \Gamma^{\mathsf{DS}})$-graphs:

Corollary 20. *Given a formula $\psi \in \mathsf{EQ\text{-}ICPDL}(\emptyset, \Gamma_M)$, we can construct $\psi' \in \mathsf{EQ\text{-}ICPDL}(\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}, \Gamma^{\mathsf{DS}})$ such that, for all valid sequences of instructions τ over $\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}$, we have $\mathcal{G}_\tau \models \psi$ iff $G_\tau \models \psi'$. The size of ψ' is $\mathcal{O}((|\mathsf{Clocks}|^2 + |\mathsf{DS}| + |\mathsf{Clocks}||DS|)|\psi|)$ and its intersection width is same as ψ.*

4) Reducing emptiness of $\mathcal{T}$ to satisfiability of EQ-ICPDL*:* From Theorem 4, we know that there exists a formula capturing realizability on weighted graphs, with signature $(\emptyset, \Gamma_M)$. Combining with Corollary 20 gives us the second main theorem of the paper regarding logical characterization of emptiness checking in timed systems with data structures.

Theorem 21 (Logical characterization of a timed system)**.** *Given a timed system with data structures $\mathcal{T}$, we can construct a formula $\Psi_\mathcal{T} \in \mathsf{EQ\text{-}ICPDL}(\emptyset, \Gamma^{\mathsf{DS}})$ such that for all $(\emptyset, \Gamma^{\mathsf{DS}})$ linear graphs G, we have $G \models \Psi_\mathcal{T}$ iff $G = \pi(G_\tau)$ for some* feasible *$\tau \in L(\mathcal{T})$. The size of $\Psi_\mathcal{T}$ is polynomial in the size of $\mathcal{T}$ and its intersection width is 2.*

Proof sketch. By Theorem 4, we can construct a formula Realizable in $\mathsf{EQ\text{-}ICPDL}(\emptyset, \Gamma_M)$ that captures realizability over weighted graphs $\mathcal{G}(\emptyset, \Gamma_M)$. By Corollary 20, we obtain a formula $\psi_{\mathsf{real}} \in \mathsf{EQ\text{-}ICPDL}(\emptyset, \Gamma^{\mathsf{DS}})$ such that, for all $\tau \in L(\mathcal{T})$, $G_\tau \models \psi_{\mathsf{real}}$ iff $\mathcal{G}_\tau \models$ Realizable. In fact, ψ_{real} is simply obtained from Realizable by replacing every reference to a weighted edge in the formula by its logical interpretation in G_τ. Now, by definition of EQ-ICPDL, we have $\psi_{\mathsf{real}} = \exists p_1 \dots p_r \psi'$ for some $\psi' \in \mathsf{ICPDL}(\{p_1, \dots p_r\}, \Gamma^{\mathsf{DS}})$.

Next, recall that a timed system $\mathcal{T}$ is a regular language of sequences of timed instructions. We consider the automaton that describes this regular collection, denoted by $\mathcal{A} = (Q, i, F, \Delta)$ with Q the set of states, i the initial state and F the final states and Δ the transition function. Then, the accepted sequences of instructions can be captured in EQ-LCPDL, by guessing the states visited along an accepting run, and by checking that consecutive states have a transition between them and start from initial and end at final state.

Set $\Sigma = \Sigma^{\mathsf{DS}}_{\mathsf{Clocks}} \cup Q = \{q_1, \dots, q_n\}$. There exists a formula $\xi = \exists q_1 \dots q_n \xi'$, with $\xi' \in \mathsf{LCPDL}(\Sigma, \Gamma^{\mathsf{DS}})$, such that, for all $(\emptyset, \Gamma^{\mathsf{DS}})$-graphs G, we have $G \models \xi$ iff $G = \pi(G_\tau)$ for some sequence $\tau \in L(\mathcal{T})$. Combining this with the formula above, and define $\psi_\mathcal{T} = \exists p_1 \dots p_r, q_1, \dots q_n (\xi' \wedge \psi')$. Then we have for any $(\emptyset, \Gamma^{\mathsf{DS}})$-graph G, $G \models \psi_\mathcal{T}$ iff $G = \pi(G_\tau)$ for some $\tau \in L(\mathcal{T})$ and τ is feasible, which completes the proof. □

C. Application: deciding emptiness

While we have reduced checking emptiness of timed systems to checking satisfiability of a formula in EQ-ICPDL, this does not immediately give decidability results. This is obvious since systems with multiple data structures (such as stacks or even single queue) are all Turing powerful, even without any timing features. To obtain decidability, one often considers under-approximations, for which we essentially restrict the class of graphs that are considered as behaviors. As mentioned in the preliminaries, graphs of bounded tree-width form a large family of graphs where we regain decidability thanks to Theorem 2. Recall that $\mathcal{G}^k$ denotes graphs of tree-width at most k. Combining Theorems 2 and 21, we have the following corollary about decidability in timed systems.

Corollary 22 (Underapproximations.)**.** *Let $k \in \mathbb{N}$. Let $\mathcal{S}$ be a timed system with data structures that uses clocks from* Clocks *and has maximum constant $M \in \mathbb{N}$. We can check whether there exists a feasible $\tau \in L(\mathcal{S})$ such that $G_\tau \in \mathcal{G}^k(\emptyset, \Gamma^{\mathsf{DS}})$ in time $2^{\mathsf{poly}(k, M, |\mathsf{Clocks}|, |\mathsf{DS}|)} \times |\mathcal{S}|^{\mathsf{poly}(k, |\mathsf{DS}|)}$.*

Thus, if the set $\{G_\tau : \tau \in L(\mathcal{S})\}$ has a bounded tree-width, we obtain the same complexity bounds for checking emptiness of $\mathcal{S}$. As concrete applications, the following models of timed systems all fall in this category of having bounded tree-width, hence we obtain decidability (and efficient algo-

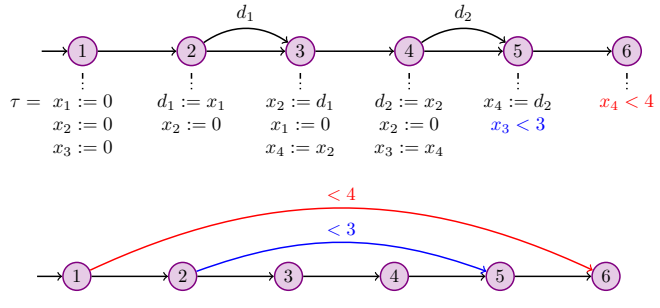


Fig. 5: Intricate flow of information in complex updates.

rithms) for checking emptiness: timed automata [8], dense-timed pushdown automata with a single stack [2], multi-stack dense-timed pushdown automata with bounded rounds [5]. In fact, the complexity obtained for dense-timed pushdown automata with a single stack is even optimal. In addition, by this technique, we also have the following (new, to the best of our knowledge) results on the decidability of the emptiness problem for multi-stack dense-timed pushdown automata with (i) bounded contexts (the tree-width of graphs in the case of p-bounded context systems is $\leq p+1$ [28]), (ii) bounded phase (the tree-width of graphs in the case of p-bounded phase systems is $\leq 2^{p+1}$ [22]), and (iii) bounded scope (the tree-width of graphs in the case of p-bounded scope is $\leq 2(p+2)$ [22]). Further, if one considers timed automata with b-bounded channels (a b-bounded channel is one where the number of unread messages is bounded by $b \in \mathbb{N}$ at any point of time), then the $(\emptyset, \Gamma^{\mathsf{DS}})$-graphs have a tree-width $\leq b+2$ [11]. We expect that many other data structures and various novel combinations (e.g., any combination of the above with multiple stacks and queues) can be handled using our technique, and leave these as routine exercises.

V. Extensions

A. Extending time features - a generic template

We develop a two-step template to add new timing features to our approach above. Step 1 consists in expressing the edges engendered by the new feature in the weighted graph and Step 2 consists in writing a formula in LCPDL to capture this new edge relation. If we can accomplish these steps, then our theorems lift to the setting with these new timing features.

This highlights the robustness of our approach, since we are able to easily and uniformly handle these extra features. That apart, this template is interesting even for timing features which can be simulated by ordinary clocks. A classical instance of this are diagonal guards in timed automata, which do not add expressiveness. Indeed, eliminating diagonal guards incurs an exponential blow-up in the worst-case [13]. This is avoided in our approach by directly expressing their edges in the weighted graph as in Equation C_4.

1) Event clocks: Let us illustrate this template in action via another example of a well-studied model, namely, event predicting clocks [9], [24], which can be simulated by ordinary (non-deterministic) timed automata. We fix a set AP of atomic propositions (events) arising from the system. An event-predicting timing instruction $\mathsf{next}_a \bowtie \alpha$, for $a \in \mathsf{AP}$, $\bowtie \in \{\leq, <, >, \geq\}$ and $\alpha \in [0, M) \cap \mathbb{N}$, entails a constraint between the current point (call it u) and the point at which node label a occurs next (call it v). Consistently with the notations on timing constraints C_1-C_4, in section IV-B1, we call this constraint C_5. Now, Step 1 is that this can be expressed in the weighted graph as an edge between these two vertices u and v. For Step 2, it is easy to write the PDL formula that allows to interpret these edges of the weighted graph as edges in the Γ^{DS}-graph. Specifically, we just have to add to the path formula $\Pi_{\triangleleft\alpha}$ in proof of Proposition 19 the following term:

$$\sum_{a \in \mathsf{AP}} \mathsf{test}\{(\mathsf{next}_a \triangleleft \alpha)\} \cdot \rightarrow \cdot (\mathsf{test}\{\neg a\} \cdot \rightarrow)^* \cdot \mathsf{test}\{a\} \quad (\mathsf{C}_5)$$

We proceed similarly for the path formula $\Pi_{\triangleleft - \alpha}$. It is not difficult to see that we can define similar formulae to capture event recording clocks as well.

2) Clock renaming via tracking: While event clocks are relatively straightforward, for some other timing features, it is not easy to figure out, from the timing instruction, what edges in the weighted graph must be added. This happens for instance in clock renaming: if we assign to x the value of clock y and then check it later with $x \leq \alpha$, the edge to be added is from the last reset of y to the point of checking the constraint. This is the case even if y has been reset in between after the assignment. Figure 5 illustrates this.

We consider a generic class of (deterministic) clock renaming in timed systems. These are a special case of clock updates, again a classical notion in timed automata [14], [13], but have not been studied much for timed systems with single or multiple data structures such as stacks and queues. We divide the features we consider into 4 classes:

(i) the usual reset of a clock x to 0 ($x := 0$),
(ii) assigning to clock x the value of clock x' ($x := x'$),
(iii) assigning to clock x the value associated to data structure $d \in \mathsf{DS}$, while reading from d ($x := d$),
(iv) writing to $d \in \mathsf{DS}$ the value of clock x ($d := x$).

Note that renamings (iii) and (iv), combined with the age and diagonal constraints on data structures, give us a very rich and expressive class of timed systems. This allows us to consider timed systems where we can write to some $d_1 \in \mathsf{DS}$ the value of a clock x_1, then read from d_1 this value (which changes with passage of time) into a clock x_2, write this value of x_2 to some $d_2 \in \mathsf{DS}$, and retrieve the value (after some time elapse) into a clock x_4. This value in x_4 can then be checked with the value read from some $d_4 \in \mathsf{DS}$, or with a clock x_5, or with a constant α. In such a sequence, the clock x_1 has come a long way at this time of checking, and we need to track it, to ensure that the time elapse we are looking at happens from the last reset of x_1 before it was written to d_1. See Figure 5, where the value of clock x_1 flows through d_1, x_2, d_2 and finally x_4, from where it is checked. Likewise, the value of clock x_2 flows through clocks x_4, x_3, and is checked at x_3. Now, x_2 is reset after it flows into x_4; however, when checking x_3, we use the reset of x_2 before x_2 flowed inside x_4.

Inferring such constraints requires us to follow and track the clock reset back to the original event. Rather than writing a formula in CPDL, we find it easier to describe an automaton which "walks" in the graph and performs this tracking. This enables us to express the weighted edges engendered by the constraints using the accepting paths of the automaton. This essentially handles the Step 1 we mentioned earlier. To handle Step 2, which is the logical definability, we write CPDL formulae whose paths π use this automaton. This allows us to interpret the weighted edges.

Formally, we construct an automaton $\mathcal{A}$ with set of states $Q = \{q_x : x \in \mathsf{Clocks}\}$. A run of $\mathcal{A}$ starting from some state q_x will track the name of the clock whose value originates from x. Without loss of generality, we assume that each transition of the timed system $\mathcal{T}$ contains exactly one update for each clock, which could be of the form $x := 0$ (reset), $x := x'$ (deterministic clock update, we use $x := x$ if the clock is unchanged), $x := d$ (x is updated with the value read from $d \in \mathsf{DS}$), or $d := x$. There are two types of transitions:

- (clock update): if there is an update $x' := x$ then we have a transition $q_x \xrightarrow{\mathsf{test}\{x':=x\}\cdot\rightarrow} q_{x'}$,
- (DS update): if there is an update $x' := d$ for some $d \in \mathsf{DS}$, then for all clocks x, we have a transition $q_x \xrightarrow{\mathsf{test}\{d:=x\}\cdot\xrightarrow{d}\cdot\mathsf{test}\{x':=d\}} q_{x'}$. This corresponds to writing the value of clock x to some $d \in \mathsf{DS}$, and, at the time of reading from $d \in \mathsf{DS}$, assign this value to a clock x'.

Consider a run $\rho = q_{x_0} \xrightarrow{\pi_1} q_{x_1} \xrightarrow{\pi_2} q_{x_2} \cdots \xrightarrow{\pi_n} q_{x_n}$ in $\mathcal{A}$. Let $\tau \in L(\mathcal{T})$ be a valid sequence of instructions from the timed system $\mathcal{T}$. Let G_τ be the associated $(\Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}, \Gamma^{\mathsf{DS}})$-graph and let u, v be vertices in G_τ. Then, $G_\tau, u, v \models \mathsf{label}(\rho) = \pi_1 \cdot \pi_2 \cdots \pi_n$ iff the value of clock x_n at v originates from clock x_0 at u. We write $G_\tau, u, v \models \mathcal{A}_{x,x'}$ if there is a run ρ of $\mathcal{A}$ from q_x to $q_{x'}$ such that $G_\tau, u, v \models \mathsf{label}(\rho)$.

Now, we can revisit and generalize the timing constraints above in (C_1–C_4) using $\mathcal{A}$ instead of the paths tracking the last reset of a clock. For instance, the subformulae (C_1–C_3) of $\Pi_{\lhd\alpha}$ in the proof of Proposition 19 should be replaced with

$$\sum_{x,x' \in \mathsf{Clocks}} \mathsf{test}\{(x := 0)\} \cdot \mathcal{A}_{x,x'} \cdot \mathsf{test}\{x' \lhd \alpha\} \qquad (\mathsf{C}_1)$$

$$+ \sum_{\substack{x,x' \in \mathsf{Clocks} \\ d \in \mathsf{DS}}} \mathsf{test}\{(x := 0)\} \cdot \mathcal{A}_{x,x'} \cdot \mathsf{test}\{d := x'\} \cdot \xrightarrow{d} \cdot \mathsf{test}\{d \lhd \alpha\} \qquad (\mathsf{C}_2)$$

$$+ \sum_{x,x',y,y' \in \mathsf{Clocks}} \mathsf{test}\{(x := 0)\} \cdot \mathcal{A}_{x,x'} \cdot \mathsf{test}\{x' - y' \lhd \alpha\} \cdot (\mathcal{A}_{y,y'})^{-1} \cdot \mathsf{test}\{(y := 0)\} \qquad (\mathsf{C}_3)$$

This completes Steps 1 and 2 of our template. Hence, timed systems with data structures whose timing features include updates can be analyzed by our approach, with a complexity blow-up that is polynomial in the size of the input. Even for timed automata without data structures, the presence of clock renamings makes the model exponentially more succinct [13]. Converting timed automata with clock renamings to ordinary timed automata (using the reduction from [14]) and then applying our technique would incur an additional exponential blowup that we avoid by using our template above.

B. Extending to other problems: Model checking

Here, we would like to check whether a system satisfies a specification. As usual, we assume a finite set AP of atomic propositions which are used to link the system and the specification, and thus we will write specifications in the logic $\mathsf{LCPDL}(\mathsf{AP}, \Gamma^{\mathsf{DS}})$. For instance, if $\mathsf{req}, \mathsf{grant} \in \mathsf{AP}$, the formula $\mathsf{A}(\mathsf{req} \implies \langle\rightarrow^+\rangle\mathsf{grant})$ says that every request should eventually be granted. As another example, the formula $\mathsf{A}((a \wedge \langle\rightarrow\cdot\xrightarrow{d}\rangle) \implies \langle\rightarrow\cdot\xrightarrow{d}\cdot\rightarrow\rangle a)$ says that, if some property $a \in \mathsf{AP}$ holds before a message is sent over data structure d, then a still holds after the message is received.

Specifications are evaluated over $(\mathsf{AP}, \Gamma^{\mathsf{DS}})$-graphs. Such graphs are generated by runs of the timed system. Again, we consider valid sequences $\tau = \tau_1 \cdots \tau_n$ of instructions over $\mathsf{AP} \cup \Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}$. An instruction $\tau_i \subseteq \mathsf{AP} \cup \Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}$ defines the atomic propositions $\tau_i \cap \mathsf{AP}$ which hold on the i^{th} event, together with the set of operations $\tau_i \cap \Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}$ which are executed at the i^{th} event. Let $G_\tau = (V, E, \lambda)$ be the $(\mathsf{AP} \cup \Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}, \Gamma^{\mathsf{DS}})$-graph associated with τ. When $\Sigma' \subseteq \Sigma$, we note $\pi_{\Sigma'}$ the projection on Σ': if $G = (V, E, \lambda)$ is a (Σ, Γ)-graph, then $\pi_{\Sigma'}(G) = (V, E, \lambda')$, where $\lambda'(u) = \lambda(u) \cap \Sigma'$ for all $u \in V$.

Let $\mathcal{T}$ be a timed system with data structures DS and let $\Phi \in \mathsf{LCPDL}(\mathsf{AP}, \Gamma^{\mathsf{DS}})$ be a specification. Recall that, in Theorem 21, we define the formula $\Psi_{\mathcal{T}} = \exists p_1, \ldots, p_n \ \Psi'_{\mathcal{T}}$. Consider $\Psi = \exists p_1, \ldots, p_n \ (\Psi'_{\mathcal{T}} \wedge \neg\Phi)$. Let $G = (V, E)$ be an $(\emptyset, \Gamma^{\mathsf{DS}})$-graph. By Theorem 21, if $G \models \Psi$ then $G_\tau \models \Psi$ and there exists a feasible $\tau \in L(\mathcal{T})$ such that $G = \pi_\emptyset(G_\tau)$. Then $G_\tau \models \neg\Phi$, and since the specification uses AP only, we deduce that $\pi_{\mathsf{AP}}(G_\tau) \models \neg\Phi$. Thus, as a corollary of Theorem 21, we can construct a formula $\Psi \in \mathsf{EQ\text{-}ICPDL}(\emptyset, \Gamma^{\mathsf{DS}})$ which is satisfiable over $(\emptyset, \Gamma^{\mathsf{DS}})$-linear graphs iff there is a run of the system which violates the specification Φ.

Corollary 23. *Let $\mathcal{T}$ be a timed system with data structures DS and let $\Phi \in \mathsf{LCPDL}(\mathsf{AP}, \Gamma^{\mathsf{DS}})$ be a specification. For all $(\emptyset, \Gamma^{\mathsf{DS}})$-linear graphs G, we can construct a formula Ψ such that $G \models \Psi$ iff there exists a* feasible *$\tau \in L(\mathcal{T})$ such that $G = \pi_\emptyset(G_\tau)$ and $\pi_{\mathsf{AP}}(G_\tau) \not\models \Phi$. The size of Ψ is polynomial in the size of $\mathcal{T}$ and Φ, and its intersection width is 2.*

VI. Conclusion

We studied timed systems via their behaviors depicted as graphs and reasoned about them via logic EQ-ICPDL. This gave rise to a problem of independent and basic interest: logical definability of realizability of weighted graphs. We showed that realizability is definable in EQ-ICPDL over sequential graphs but not definable, even in MSO, over non-sequential graphs. We developed a new logic based technique to analyze and model-check timed systems having a complex interplay of time and data structures. Potential future work would be to generalize this approach to handle larger classes of timed systems. In light of the negative result for non-sequential systems, an intriguing question is to come up with classes of concurrent systems that can be analyzed.

REFERENCES

[1] P. Abdulla, M. F. Atig, and S. Krishna. Perfect timed communication is hard. In *FORMATS Proceedings*, pages 91–107, 2018.

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

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

[4] C. Aiswarya, P. Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *ATVA Proceedings*, pages 1–17, 2014.

[5] S. Akshay, P. Gastin, and S. Krishna. Analyzing timed systems using tree automata. In *CONCUR Proceedings*, 2016.

[6] S. Akshay, P. Gastin, S. Krishna, and I. Sarkar. Towards an efficient tree automata based technique for timed systems. In *CONCUR Proceedings*, pages 39:1–39:15, 2017.

[7] S. Akshay, Paul Gastin, Vincent Jugé, and Shankara Narayanan Krishna. Timed systems through the lens of logic. *CoRR*, abs/1903.03773, 2019.

[8] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[9] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211(1-2):253–273, 1999.

[10] A. Blumensath and B. Courcelle. Monadic second-order definable graph orderings. *Logical Methods in Computer Science*, 10(1), 2014.

[11] B. Bollig and P. Gastin. Non-sequential theory of distributed systems. *CoRR*, abs/1904.06942, 2019.

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

[13] Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 10(4):393–405, 2005.

[14] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theor. Comput. Sci.*, 321(2-3):291–345, 2004.

[15] L. Clemente. Decidability of timed communicating automata. *CoRR*, abs/1804.07815, 2018.

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

[17] L. Clemente and S. Lasota. Binary reachability of timed pushdown automata via quantifier elimination and cyclic order atoms. In *ICALP Proceedings*, pages 118:1–118:14, 2018.

[18] L. Clemente, S. Lasota, R. Lazic, and F. Mazowiecki. Timed pushdown automata and branching vector addition systems. In *LICS Proceedings*, pages 1–12, 2017.

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

[20] B. Courcelle. Regularity equals monadic second-order definability for quasi-trees. In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, volume 9300 of *Lecture Notes in Computer Science*, pages 129–141. Springer, 2015.

[21] 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.

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

[23] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[24] Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. On regions and zones for event-clock automata. *Formal Methods in System Design*, 45(3):330–380, 2014.

[25] S. Göller, M. Lohrey, and C. Lutz. PDL with intersection and converse: satisfiability and infinite-state model checking. *Journal of Symbolic Logic*, 74(1):279–314, 2009.

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

[27] F. Laroussinie and N. Markey. Quantified CTL: expressiveness and complexity. *Logical Methods in Computer Science*, 10(4), 2014.

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

[29] Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.