Advanced topics in timed automata

Project report

Dhruv Nevatia

Chennai Mathematical Institute, Chennai

Contents

Part I. Timed systems through the lens of logic

1 Introduction

The emptiness problem for timed systems has been studied extensively using timed automata without any additional data structures, by checking for reachability in the untimed system and realizability of reachable paths to ensure that the constraints are satisified, using region abstraction. Many variants and extensions of this model have been considered such as introducing diagonal constraints and event clocks, or data structures like stacks and queues to the timed automata. The techniques and algorithms for solving the emptiness problems for each of these variants were specifically tailor-made for that instance and hence resulted in a vast theory for the decidability of the problem. The goal of this paper is to introduce a uniform novel approach that can be used while introducing several timing features and data structures. The technique tries to reason with the underlying graph of the model and the logical definability of appropriate properties on this graph. This shall be explained in greater detail later.

The authors of the paper have used the logic EQ-ICPDL being less powerful than MSO (proof unknown to me) whence complexity-wise being less costly than the same, is sufficient for our purposes.

Node and edge labelled graphs Let Σ and Γ be two alphabets (which can also be seen as function symbols). We consider the language with signature (Σ, Γ). A structure over this language would be a (Σ, Γ)-labelled graph with the 2-sorted universe (V, E) which intuitively mean vertices and edges where the interpretation of Σ would be a labelling relation on V and Γ would be a labelling relation on E. The vertex labelling relation can be summarised in a function $\lambda : V \to 2^{\Sigma}$, which may also be added in the universe of the (Σ, Γ)-structure. In the paper a graph has been defined to be a (Σ, Γ)-language structure (V, E, λ). E_γ is the set of all γ labelled edges and $\mathcal{G}(\Sigma, \Gamma)$ denotes the set of all (Σ, Γ)-labelled graphs.

In this paper, the graph mostly models the behaviour of sequential systems so Γ contains a symbol succ which defines a total order on V. We say $u \prec v$ instead of $(u, v) \in E_{\mathsf{succ}}$.

Propositional dynamic logic over labelled graphs We now introduce an extension of PDL called ICPDL where I and C stand for intersection and converse resp. The logic is parametrized by (Σ, Γ) denoted as ICPDL(Σ, Γ).

Syntax Let $p \in \Sigma$ and $\gamma \in \Gamma$:

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

We also consider the fragment LCPDL with loop but without intersection, which has a better complexity than ICPDL. In the above syntax Φ are sentences and E is the existensial node quantifier. Note E is different from $\exists$, the existensial quantification for elements of the structure and will be introduced in EQ-ICPDL. σ are *nodes* or *state* formulae and have one implicit free first order variable and π are *path* formulae having two free variables (the end-points of the path).

Semantics Given a (Σ, Γ)-structure $G = (V, E, \lambda)$ we now define the semantics of the logic. The semantics of a state formula σ is $[\![\sigma]\!]_G \subseteq V$ and for a path formula π is $[\![\pi]\!]_G \subseteq V^2$. Now the semantics for $\vee, \neg, +, \cdot, ^*, ^{-1}$ and $\cap$ is pretty clear. The rest have been summarised below,

$$\begin{aligned}
\mathsf{E}\sigma &= \{G \in \mathcal{M}(\Sigma, \Gamma) \mid \exists v \in V_G \cap [\![\sigma]\!]_G\} \\
\top &= V \\
p &= \{v \in V \mid p \in \lambda(v)\} \\
\langle\pi\rangle\sigma &= \{v \in V \mid \exists u \in [\![\sigma]\!].\, (v, u) \in [\![\pi]\!]\} \\
\mathsf{loop}(\pi) &= \{v \in V \mid (v, v) \in [\![\pi]\!]\} \\
\xrightarrow{\gamma} &= E_\gamma \\
\mathsf{test}\{\sigma\} &= \{(v, v) \in V^2 \mid v \in [\![\sigma]\!]\}
\end{aligned}$$

The logic used in this paper is an extension of ICPDL, namely EQ-ICPDL which consists of an additional rule for Φ, $\exists p.\, \Phi$, for $p \in \mathsf{AP}$, such that $\mathsf{AP} \cap \Sigma = \emptyset$, which denotes existensial quantification over the node alphabet. As prompted before, it is immediately seen that this is different from E.

Let's have a look at an example,

▶ **Example 1.** Let $\Sigma = \{p, q, r, s\}$ and $\Gamma = \{c, d, e, f, \mathsf{succ}\}$. We will denote $\xrightarrow{\mathsf{succ}}$ by $\rightarrow$. Now consider the graph in the following figure,

$$\begin{tikzcd}
\{p,s\} \arrow[r] \arrow[rrr, bend left, "e"] & \{q,s\} \arrow[r] & \{p,q\} \arrow[r] \arrow[rrr, bend left, "d"] \arrow[ll, bend left, "c"] & \{r\} \arrow[r] \arrow[rr, bend left, "f"] & \{q\} \arrow[r] & \{q,s\}
\end{tikzcd}$$

Fig. 1: A node and edge-leballed graph G

Following the definitions above we have $G \in \mathsf{E}\langle(\mathsf{test}\{p \vee q\} \cdot \to)^*\rangle r$ as $\{p, s\} \in \langle(\mathsf{test}\{p \vee q\} \cdot \to)^*\rangle r$. But the formula $\mathsf{E} \bigvee_{(d,d') \in \Gamma^2, d \neq d'} \mathsf{loop}(\xrightarrow{d} \cdot \xrightarrow{d'}^{-1})$ is not satisfied by G since every node has a unique non-successor edge.

Intuitively, the $\mathsf{test}\{\sigma\}$ operator allows us to reason about the intermediate nodes visited by a model for the path formula. This is done by considering an empty path from a node satisfying σ to itself. Additionally we can also check for intermittent loops satisfying some path formula in a bigger path.

We could have very well used MSO for our purposes but we will see that while EQ-ICPDL is strictly less powerful than MSO and at least as powerful as FO [4], it is sufficient for our work.

Contributions of this paper Now we are sufficiently equipped to discuss the results of this paper. The paper has three main contributions:

1. The first contribution in this paper shows that the non-emptiness problem of a timed system can be reduced to the satisfiability problem of an EQ-ICPDL formula over linear labelled graphs.
2. The second contribution shows that if the condition of linearity is removed from the models then the reduction fails and hence, it gives a tight reduction. This would explained in more detail later.
3. Lastly, they combine their results to give an agorithm to check the emptiness of several classes of timed systems.

Classical approach Before moving forward it is suggested to revisit the classical approach to the emptiness problem of classical timed automata. The main idea is to abstract the infinite space of clock valuations into finitely many equivalence classes so that there are finitely many configurations. These equivalences should be such that they preserve the corresponding runs in the automata. Such abstractions are called regions. See [9] for further details.

2 Some more preliminaries

Weighted graphs The main idea is to think of a run of the system as a linear graph (and hence only sequential systems are considered) which would then act as a (Σ, Γ)-structure for an appropriate choice of (Σ, Γ)-interpretation in the structure. Intuitively, Γ would be the set of constraints whence, $\Gamma = \{\mathsf{succ}\} \cup (\{<, \leq\} \times \mathbb{Z})$. A *linear weighted graph* is a $(\Sigma = \emptyset, \Gamma)$-structure. We denote by Γ_M, the set $\{\mathsf{succ}\} \cup (\{<, \leq\} \times \mathbb{Z}_M)$, particularly useful when we consider automata with M-bounded constraints. An adge (u, v) labelled $\triangleleft\alpha$ is denoted by (u, α, v). See [1] for an example.

▶ **Definition 2** (Realizability)**.** A weighted graph G is realizable if there exists a time-stamp map $ts : V \to \mathbb{R}$ such that

1. all constraints are satisfied: $\forall(u, \alpha, v) \in E, ts(v) - ts(u) \lhd \alpha$, and
2. ts is monotone w.r.t. the linear order: $\forall u, v \in V$, if $u \preceq v$, then $ts(u) \leq ts(v)$.

Current approach As discussed briefly in one of the contributions we aim towards constructing a formula ϕ that defines the existence of an accepting timed run viewed as a graph, given a timed system. The problem is broken down into 3 parts,

1. First, we construct a formula ϕ_1 to capture the untimed regular language of the underlying finite automata of the timed system.
2. Next, we construct a formula ϕ_2 to capture the data structure edges in the automata while also ensuring that the data flow follows the rules of the corresponding data structure, e.g., for a stack the data flow is LIFO and for a queue its FIFO. The first part follows through a similar reasoning as for ϕ_1 and the second part has been explained in detail in [2].
3. Lastly, we construct a formula ϕ_3 which captures the realizablity of the timed system. This is the part that the paper mainly focuses on.

Finally we define $\phi = \phi_1 \wedge \phi_2 \wedge \phi_3$.

3 Back to the logic

Over graphs of arbitrary tree-width, the satisfiability problem of PDL is undecidable. However, it is decidable for bounded tree-width. Let $\mathcal{G}^k(\Sigma, \Gamma)$ denote the set of all (Σ, Γ)-graphs of tree-width at most k. We next define a notion of *intersection-width* (iw) defined as, $\mathsf{iw}(\mathsf{test}\{\sigma\}) = \mathsf{iw}(\xrightarrow{\gamma}) = 1$, $\mathsf{iw}(\pi_1 + \pi_2) = \mathsf{iw}(\pi_1 \cdot \pi_2) = \max(\mathsf{iw}(\pi_1), \mathsf{iw}(\pi_2))$, $\mathsf{iw}(\pi^{-1}) = \mathsf{iw}(\pi^*) = \mathsf{iw}(\pi)$, $\mathsf{iw}(\pi_1 \cap \pi_2) = \mathsf{iw}(\pi_1) + \mathsf{iw}(\pi_2)$. An important observation is that the iw of any formula in LCPDL is 1.

Earlier we saw how the graphs could also be seen as structures for a language. This paves way for language interpretation. In terms of this paper, we shall call them *graph interpretations,* which are interpretations specific to the language $(\Sigma, \Gamma) \oplus$ EQ-ICPDL. Further details can be found in [7, 1].

▶ **Theorem 3** (Satisfiability). *Given $k \geq 1$ in unary and a formula Ψ in EQ-ICPDL(Σ, Γ) of intersection width bounded by a constant, checking whether $G \models \Psi$ for some $G \in G^k(\Sigma, \Gamma)$ can be solved in EXPTIME.*

The above result is a consequence of Theorem 3.8 in [5] which can be used for satisfiability of a formula w.r.t. the complete set of graphs (without a bound on the tree-width) given a language. However, the current statement mentions a bound on the tree-width of the graphs. This can be achieved by expanding the langauge to be able to describe $k-SST$ as defined in [2]. Given a formula ϕ we then consider the $k-SST$ representation of the models of ϕ and hence construct the corresponding formula in the larger language, ϕ' which now encodes the tree-width within the formula.

Next we convert this formula ϕ' into a *two-way alternating parity tree automata (TWAPTA)* which results in an exponential blow-up [5] and use the following result by Vardi to decide the emptiness of the resulting TWAPTA in EXPTIME, effectively giving us a 2-EXPTIME time algorithm to decide satisfiability of ϕ. *Taking ϕ as $\phi_1 \wedge \phi_2 \wedge \phi_3$ as described earlier decides the emptiness problem for timed systems.*

▶ **Theorem 4** (Vardi[10]). *For a given TWAPTA $\mathscr{T}$ with transition function S, it can be checked in time $exp(|\mathscr{T}| + i(\mathscr{T})) \cdot |\delta|^{O(1)}$ whether $L(\mathscr{T}) \neq \emptyset$.*

4 Logical definability of realizability

The goal is to view graphs as the finite behaviours (/runs) being generated by a system and reason about sets of graphs to capture the realizability of the system. In this section we discuss the construction of ϕ_3. Consequently we will have the following,

▶ **Theorem 5.** *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.*

The proof is divided into two cases. We first consider only closed guards only and then consider strict guards as well.

Closed guards If the graph contains only closed guards with the maximum constraint constant being $M-1$, then intuitively, we consider only the integral parts of the time-stamps upto a thershold $M-1$ in any input timed word to figure out whether it generates an accepting run or not.

For the following results we are given a linear weighted graph $G = (V, E)$ with $|V| = n$, we let $V = \{u_1, \ldots, u_n\}$ with $u_1 \prec \ldots \prec u_n$. If G is M-weight bounded then a time stamp mapping $\mathsf{ts} : V \to \mathbb{R}$ is called *slowly monotone* if $0 \leq \lfloor \mathsf{ts}(v) \rfloor - \lfloor \mathsf{ts}(v) \rfloor \leq M-1$ whenever $u \prec v$. The following results guide us to the first step of Theorem 5 for closed graphs.

▶ **Lemma 6.** *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.*

A *time-stamping modulo M*is a map $\mathsf{tsm} : V \to \mathbb{Z}_M$. For all $u, v \in V$, we define $d_{\mathsf{tsm}}(u, v) = (\mathsf{tsm}(v) - \mathsf{tsm}(u)) \bmod M$ and we say $(u, v) \in \mathsf{tsm}$-big if there exist $w_1, w_2 \in V$ such that $u \preceq w_1 \prec w_2 \preceq v$ and $d_{\mathsf{tsm}}(u, w_1) + d_{\mathsf{tsm}}(w_1, w_2) \geq M$.

▶ **Definition 7.** A *time-stamping modulo M* tsm is said to weakly satisfy $G = (V, E)$ if for all $e = (u, \lhd, \alpha, v) \in E$,

(a) if $u \preceq v$ then (u, v) is not tsm-big and $d_{\mathsf{tsm}}(u, v) \leq \alpha$;

(b) if $v \prec u$ then (v, u) is tsm-big or $d_{\mathsf{tsm}}(v, u) \geq -\alpha$.

▶ **Lemma 8.** *If $G \in \mathcal{G}(\emptyset, \Gamma_M)$ is realizable then there exists a time-stamping modulo M that weakly satisfies G.*

Finally,

▶ **Lemma 9.** *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.*

▶ **Corollary 10.** *For a classical timed automata $\mathcal{T}$ with only closed guards and integer constraints, if the language of the automata is non-empty then there is a timed word with only integer time stamps.*

Proof. Given that $\mathcal{T}$ is non-empty, we know there exists a timed word $\sigma = \sigma_1 \dots \sigma_n$ which has an accepting run $\tau = \tau_1 \dots \tau_n$ where $\sigma_i = (a_i, t_i)$ for $a_i \in \Sigma$ and $t_i \in \mathbb{R}^{\geq 0}$, and $\tau_i \in \Delta_{\mathcal{T}}^{\mathsf{C}}$ where the C denotes that the transitions in the system have been enriched with the corresponding constraints (and resets). We say $c \in \tau \iff$ (c occurs in the constraints for some τ_i), where c is a constraint.

Claim 11. There exists a timed word $\sigma' = \sigma'_1 \dots \sigma'_n$ with the same run τ where $\sigma'_i = (a_i, \lfloor t_i \rfloor)$.

Proof of claim 11. Suppose we are currently undergoing transition τ_i and we see a constraint of the form $x - y \leq k \in \tau_i$ where $k \in \mathbb{Z}$ and $x, y \in \mathsf{Clocks}$. Then this constraint can be translated to the constraint $t_l - t_m \leq k$ where τ_l is the reset point of clock y and τ_m that of x. For any constraint of the form $x \leq k \in \tau_i$, we can translate it to $t_i - t_l \leq k$ where τ_l is the reset point of x. Hence every constraint can be reduced to a constraint on the time stamps. Now,

$$\begin{aligned} & t_l - t_m \leq k \\ \iff & \lfloor t_l \rfloor - \lfloor t_m \rfloor + (\{t_l\} - \{t_m\}) \leq k \\ \iff & \lfloor t_l \rfloor - \lfloor t_m \rfloor \leq k + (\{t_m\} - \{t_l\}) \\ \implies & \lfloor t_l \rfloor - \lfloor t_m \rfloor < k + 1 && (\because \{t_m\} - \{t_l\} < 1) \\ \implies & \lfloor t_l \rfloor - \lfloor t_m \rfloor \leq k \end{aligned}$$

Hence, we know that σ' will satisfy all the constraints satisfied by σ. Since the letters of the timed word are identical as well, both the words will execute the same rub τ. ◀

From the above claim we know that if τ is an accepting run then σ' is also a word in the language of $\mathcal{T}$. ◀

Lemma 9 also provides an equivalent characterization of realizability of a weighted graph and we shall use this to construct the formula ϕ_3.

EQ-ICPDL characterization (Closed guards) We use existensial 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 as defined in the figure below. Let $\delta_M(i,j) = (j-i) \bmod M$. Other than that the formulae are self explanatory (refer [1] for a brief discussion).

$$
\begin{aligned}
\mathsf{Partition} &= \mathsf{A} \bigvee_{0 \le i < M} [p_i \wedge \bigwedge_{j \ne i} \neg p_j] \\
\mathsf{BigPath} &= \sum_{\substack{0 \le i,j,k < M \\ \delta_M(i,j)+\delta_M(j,k) \ge M}} \mathsf{test}\{p_i\} \cdot \rightarrow^{+} \cdot \mathsf{test}\{p_j\} \cdot \rightarrow^{+} \cdot \mathsf{test}\{p_j\} \cdot \rightarrow^{*} \\
\mathsf{SmallPath}_{i,j} &= \mathsf{test}\{p_i\} \cdot \left(\sum_{i \le k < l \le j} \mathsf{test}\{p_k\} \cdot \rightarrow \cdot \mathsf{test}\{p_l\} \right)^{*} \cdot \mathsf{test}\{p_j\} \text{ if } i \le j \\
\mathsf{SmallPath}_{i,j} &= \sum_{0 \le l \le j < i \le k < M} \mathsf{SmallPath}_{i,k} \cdot \rightarrow \cdot \mathsf{SmallPath}_{l,j} \text{ if } j < i \\
\mathsf{Forward} &= \neg \mathsf{E} \bigvee_{-M < \alpha < M} \mathsf{loop}(\mathsf{BigPath} \cdot \xrightarrow{\le \alpha}^{-1}) \\
&\quad \wedge \neg \mathsf{E} \bigvee_{\substack{0 \le i,j < M \\ \delta_M(i,j) > \alpha}} \mathsf{loop}(\mathsf{test}\{p_i\} \cdot \xrightarrow{\le \alpha} \cdot \mathsf{test}\{p_j\} \cdot (\rightarrow^{-1})^{+}) \\
\mathsf{Backward} &= \neg \mathsf{E} \bigvee_{\substack{-M < \alpha < M \\ 0 \le i,j < M \\ \delta_M(i,j) < -\alpha}} \mathsf{loop}(\mathsf{SmallPath}_{i,j} \cdot \xrightarrow{\le \alpha})
\end{aligned}
$$

Fig. 2: LCPDL for realizability of linear closed graphs

Strict guards We now introdue some necessary tools for the mixed guard characterization.

▶ **Definition 12.** Given a graph $G = (V, E)$ and a time-stamping $\mathsf{tsm} : V \to \mathbb{Z}_M$ mod 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 holds:
  1. $u \prec v$, (u, v) is not tsm-big and $\mathsf{d}_{\mathsf{tsm}}(u, v) = \alpha$ for some edge $(u, \lhd, \alpha, v) \in E$;
  2. $v \prec u$, (v, u) is not tsm-big and $\mathsf{d}_{\mathsf{tsm}}(v, u) = -\alpha$ for some edge $(u, \lhd, \alpha, v) \in E$;
  3. $v \prec u$ and $\mathsf{d}_{\mathsf{tsm}}(u, v) = 0$.
- $(u, v) \in \mathsf{gt}_{\mathsf{Fr}}$ iff one of the following holds:

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 $(u,<,\alpha,v) \in E$.

The idea is that the 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 $\mathsf{ts}(u) \geq$ (resp. $>$) the fractional part of $\mathsf{ts}(v)$. This definition leads to the final result for the complete logical characterization of realizability of weighted graphs. The proof has been omitted below and largely follows from definitions.

▶ **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 $\mathsf{geq}_{\mathsf{Fr}}$.*

IQ-ICPDL characterization (Mixed guards) To state weak realizability, we use the formula WRealizable = Partition $\wedge$ Forward $\wedge$ Backward where the subformulae have been defined in fig. 2. In addition, we need to check the absence of cycle among the fractional parts, which contains at least one strict inequality. By lemma 13, this suffices to ensure realizability.

$$\mathsf{Realizable} = \exists p_0, \ldots, p_{M-1} \mathsf{WRealizable} \wedge \neg \mathsf{Eloop}(\mathsf{gt}_{\mathsf{Fr}}.{\mathsf{geq}_{\mathsf{Fr}}}^*)$$

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 5.

Importance of linearity So far we have seen logical characterization of realizability of weighted graphs having a unique linear order in our beloved EQ-ICPDL logic. However the paper also displays the tightness of the approach by showing the undefinability of realizability for non-linear weighted graphs in general. This crucially relies on the MSO-undefinability of non-regular languages like $L = \{a^n b^n \mid n \geq 0\}$.

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

5 Reducing emptiness of $\mathcal{T}$ to satisfiability of EQ-ICPDL

Timing instructions In a timed system $\mathcal{T}$ with data structures, the sequence of instructions generated by the system includes (i) checking constraints on clocks (ii) checking constraints on data structures and (iii) mixing operations on clocks and data structures. This sequence of instructions can be defined as a regular language of constraints in $\mathcal{T}$. The set of valid sequences generated by $\mathcal{T}$ is denoted by $L(\mathcal{T})$. A valid sequence of constraints $\tau = \tau_1 \ldots \tau_n$ is said to be *feasible* if there exists a time-stamping $\mathsf{ts} : \{1, \ldots, n\} \to \mathbb{R}^{\geq 0}$ such that all timing constraints given by τ are satisfied. For every such sequence we can construct a weighted graph denoting the run (we may forget the actual transitions) and retaining the constraints as weighted edges; the nodes enriched with resets and data operations, denoted by G_τ. Moreover, 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 constraint of the form $d \bowtie \beta$ at position i, we have an edge $j \xrightarrow{d} i$ in G_τ, 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 $\tau \in L(\mathcal{T})$. This problem can be reduced to the problem of checking satisfiability of an EQ-ICPDL formula over the enriched weighted graphs from before.

▶ **Proposition 15.** *Let τ be a valid sequence of instructions. Then the weighted graph $\mathcal{G}_\tau$ can be* CPDL*-interpreted in G_τ.*

From timing instructions to weighted graphs Any formula over weighted graphs can thus be translated into an "equivalent" formula over the enriched graphs,

▶ **Corollary 16.** *Given a formula $\psi \in$ EQ-ICPDL$(\emptyset, \Gamma_M)$, we can construct $\psi' \in$ EQ-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 $\mathcal{O}((|\mathsf{Clocks}|^2 + |\mathsf{DS}| + |\mathsf{Clocks}||\mathsf{DS}|)|\psi|)$ and its intersection width is same as ψ.*

Finally combining Theorem 5 and corollary 16 we obtain the complete logical characterization of the non-emptiness problem of timed systems.

▶ **Theorem 17** (Logical characterization of a timed system)**.** *Given a timed system with data structures $\mathcal{T}$, we can construct a formula $\Psi_{\mathcal{T}} \in$ EQ-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.*

6 Extending to other problems : Model checking

Apart from several non-trivial extensions in regards to more complicated timing and data features, a rather crucial extension of the current characterization of the non-emptiness problem would be towards *model checking*. In this problem, 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.

Let $G_\tau = (V, E, \lambda)$ be the $(\mathsf{AP} \cup \Sigma^{\mathsf{DS}}_{\mathsf{Clocks}}, \Gamma^{\mathsf{DS}})$-graph associated with a run 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 17, we define the formula $\Psi_{\mathcal{T}} = \exists p_1, \ldots, p_n \Psi'_{\mathcal{T}}$. Consider $\Psi = \exists p_1, \ldots, p_n(\Psi'_T \wedge \neg\Phi)$. Let $G = (V, E)$ be an $(\emptyset, \Gamma^{\mathsf{DS}})$-graph. By Theorem 17, 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 17, 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 18.** *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. We can construct a formula Ψ such that, for all $(\emptyset, \Gamma^{\mathsf{DS}})$-linear graphs G, $G \models \Psi$ 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.*

Part II. Relating timed and register automata

1 Introduction

In this paper we study the decidability and complexity of some common decision problems : *non-emptiness*, *universality* and *inclusion* of register and timed automata, in particular the general model we consider in this paper is *alternating timed and register automata*. Both models and there corresponding submodels have similar complexity results for the decision problems stated above. However, for the register models we consider the data domain to be *totally ordered*, which is a necessary extension to simulate the inherent order in timed stamps; this will be crucial to prove the tight equivalence of runs of timed and register automata. Additionally we preserve the number of registers and clocks in the *transformation* to and from the register and timed context.

In this paper we shall only concern ourselves with finite timed and data words.

Some preliminaries Let $\mathbb{R}_+$ denote the set of non-negative real numbers. Let $\mathscr{B}^+(X)$ denote the set of all boolean combinations over the set of propositions X. Every formula is generated using the following grammar:

$$\phi ::= \qquad x \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \qquad (x \in X)$$

We also fix a finite alphabet Σ.

Alternating timed automata A *timed word* over Σ is a finite sequence

$$w = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$$

where $a_i \in \Sigma$ and $t_i \in \mathbb{R}_+$ such that $t_i \leq t_{i+1}$ for $i \in \{1, \dots n-1\}$.

For set of clock variables $\mathscr{C}$, the set of clock constraints denoted by $\mathsf{Constr}(\mathscr{C})$ will be generated using the following grammar:

$$\sigma ::= c < k \mid c \leq k \mid \sigma_1 \wedge \sigma_2 \mid \neg\sigma$$

where $c \in \mathscr{C}$ and $k \in \mathbb{R}_+ \cap \mathbb{Z}$. Note that diagonal constraints can be reduced to the above grammar so we don't explicitly consider diagonal constraints in our framework. A *valuation* of the clocks will be denoted by $\boldsymbol{v} \in (\mathbb{R}_+)^{\mathscr{C}}$. Let $[\sigma] \subseteq (\mathbb{R}_+)^{\mathscr{C}}$ denote the set of clock valuations satisfying σ.

▶ **Definition 19** (ATA). An *alternating timed automata* over Σ consists of a finite set of states Q, an initial state $q_0 \in Q$, a set of accepting states $F \subseteq Q$, a finite set of clocks $\mathscr{C}$ a transition relation,

$$\delta : Q \times \Sigma \times \mathsf{Constr}(\mathscr{C}) \dot{\to} \mathscr{B}^+(Q \times \mathscr{P}(\mathscr{C}))$$

with the restriction that for every state and input letter the corresponding σ's form a partition of $(\mathbb{R}_+)^{\mathscr{C}}$. $\boldsymbol{v} + t$ denotes an increment of all the clocks by t units and $\boldsymbol{v}[X := 0]$ denotes reset of all clocks in X.

For an ATA $\mathcal{A}$ and a word w we play a game between Adam and Eve to accept or reject the word. The game has $|w|$ phases i.e., for every transition we play a game between Adam and Eve and then chose the next state and valuation of clocks. We start from the configuration $(q_0, \boldsymbol{v}_0)$ where all the clocks are assigned 0. At any point of time we have a unique choice of a boolean formula ϕ due to our initial assumptions on partition of the valuation space, we then proceed as follows,

- $\phi = \phi_1 \wedge \phi_2$; Adam chooses one of ϕ_1 and ϕ_2.
- $\phi = \phi_1 \vee \phi_2$; Eve chooses one of ϕ_1 and ϕ_2.
- $\phi = (q, X)$; the phase which started with the configuration $(q', \boldsymbol{v}')$ ends with the configuration $(q, \boldsymbol{v}'[X])$.

We accept w if Eve has a strategy such that the game for w ends in a final state. We then say $w \in L(\mathcal{A})$.

Alternating register automata We first fix an infinite data domain $\mathbb{D}$ with a total order $\preceq$. *Data words* over Σ are finite sequences of the form

$$w = (a_1, d_1) \ldots (a_n, d_n)$$

where $a_i \in \Sigma$ and $d_i \in \mathbb{D}$ for $i \in \{1, \ldots n-1\}$.

For set of register variables $\mathscr{R}$, the set of register tests denoted by $\mathsf{Tests}(\mathscr{R})$ will be generated using the following grammar:

$$\sigma ::= \prec r \mid \preceq r \mid \sigma_1 \wedge \sigma_2 \mid \neg\sigma$$

where $r \in \mathscr{R}$. A *valuation* of the registers will be denoted by $\boldsymbol{v} \in (\mathbb{D})^{\mathscr{R}}$. Let $[\sigma] \subseteq \mathbb{D}^{\mathscr{R}} \times \mathbb{D}$ denote the set of register valuations satisfying σ.

▶ **Definition 20** (ARA). An *alternating register automata* over Σ consists of a finite set of states Q, an initial state $q_0 \in Q$, a set of accepting states $F \subseteq Q$, a finite set of registers $\mathscr{R}$ and a transition relation,

$$\delta : Q \times \Sigma \times \mathsf{Tests}(\mathscr{R}) \dot{\rightarrow} \mathscr{B}^+(Q \times \mathscr{P}(\mathscr{R}))$$

with the restriction that for every state and input letter the corresponding σ's form a partition of $\mathbb{D}^{\mathscr{C}} \times \mathbb{D}$. $\boldsymbol{v}[X]$ denotes loads of all registers in X. If all the constraints in the automaton are boolean combinations of $= r$ and $\neq r$ then we call such an automaton an *order-blind automaton.*

For an ARA $\mathcal{A}$ we have a similar acceptance scheme as in ATA. Hence, we similarly define the language $L(\mathcal{A})$.

Isomorphism The following result is the most crucial in the register$\leftrightarrow$timed transformation. When transforming from one domain to the other we don't remember the exact data values but instead the mutual order between them. It turns out that this is sufficient to preserve acceptance of the corresponding runs in the tranformed and pre-transformed automata.

A *time isomorphism* is an isomorphism $f : ([0,1), \leq) \rightarrow ([0,1), \leq)$ and a *data isomorphism* is an isomorphism $f : (\mathbb{D}, \preceq) \rightarrow (\mathbb{D}, \preceq)$ between the two respective algebras. Note that any timed isomorphism f can be extended to an isomorphism $f' : x \mapsto \lfloor x \rfloor + f(\{x\})$. We abuse our notation by labelling these extended morphisms as timed isomorphisms as well.

▶ **Theorem 21.** *Languages recognized by alternating timed (or register) automata is closed under timed (or data) isomorphisms.*

2 Braids

The *data projection* of a data word $w = (a_1, d_1) \dots (a_n, d_n)$ is the projection of a data word onto the data values $d_1 \dots d_n$. Any data projection can be partitioned further into maximal increasing subwords called *ordered partition*, i.e. we consider the factorization $w = w_1 \dots w_n$ where $w_{(i,1)} \prec w_{(i-1,|w_{i-1}|)}$ for $i \geq 2$. Likewise can be done for data words as well.

Data braid A *data braid* is a data word satisfying the following conditions,

- minimum data value occurs in the first position.
- each factor w_i is a substring of w_{i+1}.
- We can partition the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ so that a data value $d_i = d_1$ iff $a_i \in \Sigma_2$. Σ_2-labelled positions are called marked positions.

Timed braid A *timed braid* is a timed word satisfying the following conditions,

- first time stamp is 0.
- for all $i < n$, if $t_i < \lfloor t_n \rfloor$ then $t_i + 1$ appears among $t_{i+1} \dots t_n$.
- the alphabet can be partitioned into $\Sigma = \Sigma_1 + \Sigma_2$ so that the marked positions are precisely those with integer time-stamps.

Look at Table 1 for some examples and non-examples of braids.

	Word	Braid (?)
1	$w = (c, 1) \cdot (d, 1)(a, 4)(b, 8) \cdot (c, 1)(b, 2)(a, 4)(a, 8)(b, 9) \cdot (c, 1)$	No
2	$w_1 = (c, 1) \cdot (d, 1)(a, 4)(b, 8) \cdot (c, 1)(b, 2)(a, 4)(a, 8)(b, 9)$	Data braid
3	$v = (c, 3) \cdot (d, 2)(a, 3)(b, 8) \cdot (c, 2)(b, 3)(a, 5)(a, 8)$	No
4	$v_1 = (c, 2) \cdot (d, 2)(a, 3)(b, 8) \cdot (c, 2)(b, 3)(a, 5)(a, 8)$	Data braid
5	$w' = (\bar{b}, 2)(a, 4) \cdot (\bar{a}, 2)(b, 4)(b, 8) \cdot (\bar{b}, 2)(b, 3)(a, 4)(a, 8)(b, 9)$	Data braid
6	$v' = (\bar{b}, 0.0)(a, 0.5) \cdot (\bar{a}, 1.0)(b, 1.5)(b, 1.6) \cdot (\bar{b}, 2.0)(b, 2.3)(a, 2.5)(a, 2.6)(b, 2.9)$	Timed braid

Tab. 1: Examples and non-examples of braids

In example 1, w is a non-example because it does not satisfy the sub-string property. If you remove the $(c, 1)$ it then satisfies the substring property; hence, w_1 is a data braid. v is a non-example because the first data value of the word is not the least in the whole word. Changing the value from 3 to 2 would then make it a data braid; hence, v_1 is a data braid.

Encoding It's possible to convert timed (data) words to timed (data) braids. Moreover, timed and data braids are interconvertible. We demonstrate this using an example.

▶ **Example 22.**

$(a, 4) \cdot (b, 1)(a, 4)(b, 8) \cdot (a, 1)(a, 5)(a, 8)$	w
$(\bar{\checkmark}, 1)(a, 4) \cdot (\bar{b}, 1)(a, 4)(b, 8) \cdot (\bar{a}, 1)(\checkmark, 4)(a, 5)(a, 8)$	$db(w)$

$(a, 0.0)(a, 0.7) \cdot (b, 1.5) \cdot (b, 2.0)$	v
$(\bar{a}, 0.0)(a, 0.7) \cdot (\bar{\checkmark}, 1.0)(b, 1.5)(\checkmark, 1.7) \cdot (\bar{b}, 2.0)(\checkmark, 2.5)(\checkmark, 2.7)$	$tb(v)$

$(\bar{\checkmark}, 0.0)(a, 0.1) \cdot (\bar{b}, 1)(a, 1.1)(b, 1.3) \cdot (\bar{a}, 2)(\checkmark, 2.1)(a, 2.2)(a, 2.3)$	$tb(w)$

$(\bar{a}, 1)(a, 3) \cdot (\bar{\checkmark}, 1)(b, 2)(\checkmark, 3) \cdot (\bar{b}, 1)(\checkmark, 2)(\checkmark, 3)$	$db(v)$

For conversion from words to the corresponding braids the idea is to first use $\checkmark$ to fill the gaps so as to satisfy the substring property. Then we consider our alphabet as $\Sigma \cup \bar{\Sigma}$ where $\bar{\Sigma}$ contains a copy of Σ but with bars above them to denote marked alphabet. We use $\bar{\Sigma}$ to mark the appropriate letters in the word.

Now note that in the bottom two rows we have converted the corresponding braids to the other braid structure. This is where we use the total on the data domain $\mathbb{D}$ to construct an injection from the fractional parts of the time-stamps into $(\mathbb{D}, \prec)$. Observe that this is an interconversion. While converting from a data braid to a timed braid we use the count of the number of factors to discertain the integral part of the time stamp and while converting from the timed braid to the data braid we use the fractional part of the time stamps to get the data values.

3 Transformation

In this section, we will show that upto appropriate encoding the languages recognized by timed automata are recognized by register automata as well and vice versa, all this while preserving the number of clocks and registers.

Timed to register

▶ **Theorem 23.** *Given an alternating timed automaton $\mathscr{A}$ one can compute in exponential time an order-blind register automaton $\mathscr{B}$ such that for any timed word w, $\mathscr{A}$ accepts w if an only if $\mathscr{B}$ accepts $db(w)$. The number of registers of $\mathscr{B}$ equals the number of clocks of $\mathscr{A}$. Moreover, $\mathscr{B}$ is deterministic (resp. nondeterministic, alternating) if $\mathscr{A}$ is so.*

Proof sketch. Intuitively, we remember the integral part of the clock values in the the states upto the maximum threshold needed. For the transitions, we guess which clocks are being reset and accordingly compute the integral parts of the state to be reached. These give it the exponential blow-up. As for the transitions themselves consider the following transition in the timed automaton,

$$q, a, \sigma \mapsto \phi$$

and include the following transition in the transformed automaton,

$$(q, v), a, (\bigwedge_{c \in X} = c \wedge \bigwedge_{c \notin X} \neq c) \mapsto \phi_v^X \text{ for } a \in \Sigma \cup \bar{\Sigma}.$$

Where ϕ_v^X is constructed appropriately using ϕ containing the information on the registers to be loaded and the integral parts to be updated. It is noted that the structure of ϕ_v^X is the same as that of ϕ and so the mode of computation of the automaton does not change. ◀

▶ **Lemma 24.** *The complement of the language of data braids is recognized by a nondeterministic one register automaton.*

▶ **Theorem 25.** *The following decision problems for timed automata: language inclusion, language equality, nonemptiness and universality, reduce to the analogous problems for register automata. The reductions keep the number of registers equal to the number of clocks, and preserve the mode of computation (nondeterministic, alternating) of the input automaton.*

Proof sketch. We prove the following statement,

$$L(\mathscr{A}) \subseteq L(\mathscr{B}) \iff L(\mathscr{A}') \subseteq L(\mathscr{B}') \cup L(\mathscr{A}_{\neg db})$$

where $\mathscr{A}$ and $\mathscr{B}$ are the timed automata and the primed ones are the transformed automata as in Theorem 23.

For the forward implication note that the conversion of timed braids to data braids is not a surjective operation i.e. every data braid is not the data braid of some timed word. Hence, when trying to prove $L(\mathscr{A}') \subseteq L(\mathscr{B}') \cup L(\mathscr{A}_{\neg db})$ and considering a data braid $w \in L(\mathscr{A})$ we first remove all the useless $\checkmark$'s to make it into the data braid image of a timed word i.e. retaining the acceptance. And then proceed as needed. ◀

Register to timed Likewise, we have similar results when converting from register to timed. The procedure is very similar in most cases as well (Refer [3] for detailed proofs).

▶ **Theorem 26.** *Given an alternating register automaton $\mathscr{A}$ one can compute in exponential time a timed automaton $\mathscr{B}$ such that for any data word w, $\mathscr{A}$ accepts w if an only if $\mathscr{B}$ accepts $tb(w)$. The number of clocks of $\mathscr{B}$ equals the number of registers of $\mathscr{A}$. Moreover, $\mathscr{B}$ is deterministic (resp. nondeterministic, alternating) if $\mathscr{A}$ is so.*

Proof sketch. The proof heavily depends on the structure of a data braid Essentially, we remember whether for every register the last marked position was seen before or after the last load or repeat of the data value in the register. This information is encoded in the language of clocks and the construction is carried out. One important thing to note is that the constraints in transformed automaton are only equality checks. ◀

▶ **Lemma 27.** *The complement of the language of timed braids is recognized by a nondeterministic one clock automaton.*

▶ **Theorem 28.** *The following decision problems for register automata: language inclusion, language equality, nonemptiness and universality, reduce to the analogous problems for timed automata. The reductions keep the number of clocks equal the number of registers, and preserve the mode of computation (nondeterministic, alternating) of the input automata.*

Applications We state some examples showing the tight equivalence of the two automaton models.
Applying, Theorem 28 we have the following,

▶ **Theorem 29.** *The emptiness problem for alternating one register automata is decidable.*

The following result is immediate as the analogous problem is decidable and non-primitive recursive for timed automata. Decidability follows from Theorem 28 while the lower bound follows from Theorem 25.

▶ **Theorem 30.** *Consider the following inclusion problem: Given a nondeterministic register automaton $\mathscr{A}$ (with possibly many registers) and an alternating one register automaton $\mathscr{B}$, is every word accepted by $\mathscr{A}$ also accepted by $\mathscr{B}$? This problem is decidable and of non-primitive recursive complexity.*

4 Additional observations

Recently in [8], the authors proved that the containment problem for unambiguous register automata is decidable.

▶ **Theorem 31.** *[8] The containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$, where $\mathcal{A}$ is a non-deterministic register automaton and $\mathcal{B}$ is an unambiguous register automaton, is in* 2-EXPSPACE. *If the number of registers of $\mathcal{B}$ is fixed, the problem is in* EXPSPACE.

We should note that the register automata model considered here is inherently order-blind since the data domain was considered to be unordered. However, since our construction yields an order-blind register automaton even though the data domain is completely ordered we shall not talk about it explicitly.

An optimistic reader would hope that all our constructions in the case of timed-register preserve unambiguity of the automaton model and then using Theorem 25 in fell swoop we prove the following,

▶ **Conjecture 32.** *The containment problem for unambiguous timed automata is decidable.*

Unfortunately, though we can't come to a conclusion using this approach. *Even though, our construction in Theorem 23 does preserve unambiguity in the register automaton for data braids*, it doesn't say anything for non-data braids. Moreover, Lemma 24 introduces outright non-determinism in the resultant automaton for the language $L(\mathscr{B}') \cup L(\mathscr{A}_{\neg db})$ as denoted in the proof of Theorem 25 because we take disjunct union of register automaton in its proof.

Possible extensions Another possible extension would be to consider alternating timed pushdown automata as well as the analogous model of alternating register pushdown automata and investigate the extendability of the current approach to this larger class of automata to study the possible tight equivalence their subclasses. There are no known results in this direction to the best of my knowledge and seems to be an interesting take on the current ideas that can be explored further.

Unfortunately, I did not have enough time and resources to make this a part of my reading project.

References

[1] S Akshay, Paul Gastin, Vincent Jugé, and Shankara Narayanan Krishna. Timed systems through the lens of logic. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2019.

[2] Benedikt Bollig and Paul Gastin. Non-sequential theory of distributed systems. *arXiv preprint arXiv:1904.06942*, 2019.

[3] Diego Figueira, Piotr Hofman, and Sławomir Lasota. Relating timed and register automata. *arXiv preprint arXiv:1011.6432*, 2010.

[4] Stefan Göller, Markus Lohrey, and Carsten Lutz. Pdl with intersection and converse is 2exp-complete. In *International Conference on Foundations of Software Science and Computational Structures*, pages 198–212. Springer, 2007.

[5] Stefan Göller, Markus Lohrey, and Carsten Lutz. Pdl with intersection and converse: satisfiability and infinite-state model checking. *The Journal of Symbolic Logic*, 74(1):279–314, 2009.

[6] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[7] David Marker. *Model theory: an introduction*, volume 217. Springer Science & Business Media, 2006.

[8] Antoine Mottet and Karin Quaas. The containment problem for unambiguous register automata and unambiguous timed automata. *Theory of Computing Systems*, pages 1–30, 2020.

[9] B Srivathsan. Language emptiness for timed automata. `https://www.cmi.ac.in/~madhavan/courses/qath-2015/slides/lecture8-regions.pdf`, 2015.

[10] Moshe Y Vardi. Reasoning about the past with two-way automata. In *International Colloquium on Automata, Languages, and Programming*, pages 628–641. Springer, 1998.