ANR-NRF ProMiS

Deliverable 2

Final results on a formalism to solve information through parametric verification

Author: Étienne André

Information leakage can have dramatic consequences on systems security. Among harmful information leaks, the timing information leakage occurs whenever an attacker successfully deduces confidential internal information. In a first line of works [AS19; And+22], we consider that the attacker has access (only) to the system execution time. We address the following timed opacity problem: given a timed system, a private location and a final location, synthesize the execution times from the initial location to the final location for which one cannot deduce whether the system went through the private location. We also consider the full timed opacity problem, asking whether the system is opaque for all execution times. We show that these problems are decidable for timed automata (TAs) [AD94] but become undecidable when one adds parameters, yielding parametric timed automata (PTAs) [AHV93]. We identify a subclass with some decidability results. We then devise an algorithm for synthesizing PTAs parameter valuations guaranteeing that the resulting TA is opaque. We finally show that our method can also apply to program analysis.

In a second direction, we considered a notion of non-interference for timed automata (TAs) [AD94] that allows to quantify the frequency of an attack; that is, we infer values of the minimal time between two consecutive actions of the attacker, so that (s)he disturbs the set of reachable locations. We also synthesize valuations for the timing constants of the TA (seen as parameters) guaranteeing non-interference. We show that this can reduce to reachability synthesis in parametric timed automata. We apply our method to a model of the Fischer mutual exclusion protocol and obtain preliminary results [AK20].

Document completed by Étienne André on March 14, 2022.

References

[And+22] Étienne André, Didier Lime, Dylan Marinho, and Jun Sun. "Guaranteeing timed opacity using parametric timed model checking". In: *ACM Transactions on Software Engineering and Methodology* (2022). To appear. (cit. on p. 1).

[AK20] Étienne André and Aleksander Kryukov. "Parametric non-interference in timed automata". In: *ICECCS* (Mar. 4–6, 2021). Ed. by Yi Li and Alan Liew. CORE Rank A. Singapore, 2020, pp. 37–42. DOI: 10.1109/ICECCS51672.2020.00012 (cit. on p. 1).

[AS19] Étienne André and Jun Sun. “Parametric Timed Model Checking for Guaranteeing Timed Opacity”. In: *ATVA* (Oct. 28–31, 2019). Ed. by Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza. Vol. 11781. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, 2019, pp. 115–130. DOI: 10.1007/978-3-030-31784-3_7 (cit. on p. 1).

[AD94] Rajeev Alur and David L. Dill. “A theory of timed automata”. In: *Theoretical Computer Science* 126.2 (Apr. 1994), pp. 183–235. DOI: 10.1016/0304-3975(94)90010-8 (cit. on p. 1).

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. “Parametric real-time reasoning”. In: *STOC* (May 16–18, 1993). Ed. by S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal. San Diego, California, United States: ACM, 1993, pp. 592–601. DOI: 10.1145/167088.167242 (cit. on p. 1).

Parametric non-interference in timed automata

Étienne André
Université de Lorraine, CNRS, Inria, LORIA
Nancy, France
0000-0001-8473-9555

Aleksander Kryukov
Université de Lorraine, CNRS, Inria, LORIA
Nancy, France

***Abstract*—We consider a notion of non-interference for timed automata (TAs) that allows to quantify the frequency of an attack; that is, we infer values of the minimal time between two consecutive actions of the attacker, so that (s)he disturbs the set of reachable locations. We also synthesize valuations for the timing constants of the TA (seen as parameters) guaranteeing non-interference. We show that this can reduce to reachability synthesis in parametric timed automata. We apply our method to a model of the Fischer mutual exclusion protocol and obtain preliminary results.**

***Index Terms*—security, non-interference, parametric timed automata**

I. Introduction

Timed automata (TAs) [AD94] are a powerful formalism using which one can reason about complex systems involving time and concurrency. Among various security aspects, *non-interference* addresses the problem of deciding whether an intruder (or attacker) can disturb some aspects of the system.

In [BT03], a decidable notion of non-interference is proposed to detect whether an intruder with a given frequency of the actions (s)he can perform is able or not to disturb the set of *discrete* reachable behaviors (locations); that is, this notion can quantify the *frequency* of an attack In this paper, we extend that definition in two different ways: first, by allowing some free parameters within the model—that becomes a parametric timed automaton (PTA) [AHV93]. Second, by *synthesizing* the admissible frequency for which the system remains secure, i. e., for which the actions of the intruder cannot modify the set of reachable locations.

a) Contribution: In this work, we propose a parametric notion of non-interference in timed automata that allows to *quantify* the speed of the attacker necessary to disturb the model. Our contribution is threefold: 1) we define a notion of n-location-non-interference for timed automata; 2) we show that checking this notion can reduce to reachability synthesis in PTAs; 3) we model a benchmark from the literature, spot and correct an error in the original model, and we automatically infer using IMITATOR [And+12] parameter valuations for which the system is n-location-non-interfering.

This is the author version of the manuscript of the same name published in the proceedings of the 25th International Conference on Engineering of Complex Computer Systems (ICECCS 2020). This work is partially supported by the ANR-NRF French-Singaporean research program ProMiS (ANR-19-CE25-0015).

b) Related work: It is well-known (see e. g., [Koc96; Ben+15]) that time is a potential attack vector against secure systems. That is, it is possible that a non-interferent (secure) system can become interferent (insecure) when timing constraints are added [GMR07]. In [Bar+02; BT03], a first notion of *timed* non-interference is proposed, based on traces and locations. The latter is decidable as it reduces to the reachability problem for TAs [AD94]. In [GMR07], Gardey *et al.* define timed strong non-deterministic non-interference (SNNI) based on timed language equivalence between the automaton with hidden low-level actions and the automaton with removed low-level actions.

In [Cas09], the problem of checking opacity for timed automata is considered: it is undecidable whether a system is opaque, i. e., whether an attacker can deduce whether some set of actions was performed, by only observing a given set of observable actions (with their timing). In [AS19], we proposed an alternative (and decidable) notion of opacity for timed automata, in which the intruder can only observe the *execution time* of the system. We also extend this notion to PTAs, and propose a procedure to automatically synthesize internal timings and admissible execution times for which the system remains opaque.

In [VNN18], Vasilikos *et al.* define the security of timed automata in term of information flow using a bisimulation relation and develop an algorithm for deriving a sound constraint for satisfying the information flow property locally based on relevant transitions.

In [Ben+15], Benattar *et al.* study the control synthesis problem of timed automata for SNNI. That is, given a timed automaton, they propose a method to automatically generate a (largest) sub-systems such that it is non-interferent if possible. Different from the above-mentioned work, our work considers parametric timed automata, i. e., timed systems with unknown design parameters, and focuses on synthesizing parameter valuations which guarantee non-interference. In [NNV17], the authors propose a type system dealing with non-determinism and (continuous) real-time, the adequacy of which is ensured using non-interference. We share the common formalism of TA; however, we mainly focus on non-interference seen as the set of reachable locations, and we *synthesize* internal parts of the system (clock guards), in contrast to [NNV17] where the system is fixed.

c) Outline: In Section II, we recall the necessary preliminaries, including non-interference for TAs. In Section III,

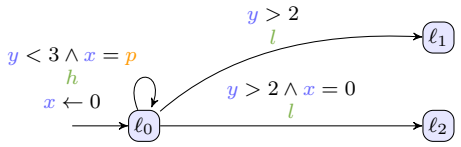


Figure 1: A PTA example

we define the problem of parametric location-non-interference for (P)TAs, and we draft a solution reducing to reachability synthesis. In Section IV, we propose a new model for the Fischer mutual exclusion protocol. In Section V, we apply IMITATOR to this model, and obtain preliminary results ensuring non-interference. We sketch future directions of research in Section VI.

II. Preliminaries

We assume a set $\mathbb{X} = \{x_1, \ldots, x_H\}$ of *clocks*, i. e., real-valued variables that evolve at the same rate. A clock valuation is $\mu : \mathbb{X} \to \mathbb{R}_{\geq 0}$. We write $\vec{0}$ for the clock valuation assigning 0 to all clocks. Given $d \in \mathbb{R}_{\geq 0}$, $\mu + d$ is s.t. $(\mu + d)(x) = \mu(x) + d$, for all $x \in \mathbb{X}$. Given $R \subseteq \mathbb{X}$, we define the *reset* of a valuation μ, denoted by $[\mu]_R$, as follows: $[\mu]_R(x) = 0$ if $x \in R$, and $[\mu]_R(x) = \mu(x)$ otherwise.

We assume a set $\mathbb{P} = \{p_1, \ldots, p_M\}$ of *parameters*. A parameter *valuation* v is $v : \mathbb{P} \to \mathbb{Q}_+$. We assume $\bowtie \in \{<, \leq, =, \geq, >\}$. A guard g is a constraint over $\mathbb{X} \cup \mathbb{P}$ defined by a conjunction of inequalities of the form $x \bowtie \sum_{1 \leq i \leq M} \alpha_i p_i + d$, with $p_i \in \mathbb{P}$, and $\alpha_i, d \in \mathbb{Z}$. Given g, we write $\mu \models v(g)$ if the expression obtained by replacing each x with $\mu(x)$ and each p with $v(p)$ in g evaluates to true.

A. Parametric timed automata

Definition 1 (PTA)**.** A PTA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, E)$, where: *i*) Σ is a finite set of actions, *ii*) L is a finite set of locations, *iii*) $\ell_0 \in L$ is the initial location, *iv*) $\mathbb{X}$ is a finite set of clocks, *v*) $\mathbb{P}$ is a finite set of parameters, *vi*) I is the invariant, assigning to every $\ell \in L$ a guard $I(\ell)$, *vii*) E is a finite set of edges $e = (\ell, g, a, R, \ell')$ where $\ell, \ell' \in L$ are the source and target locations, $a \in \Sigma$, $R \subseteq \mathbb{X}$ is a set of clocks to be reset, and g is a guard.

Example 1. Consider the PTA in Fig. 1, containing two clocks x and y, and one parameter p. ℓ_0 is the initial location. Observe that the transition to ℓ_2 can only be taken if the difference between y and x is larger than 2. This can only happen for selected valuations of the parameter p.

Given v, we denote by $v(\mathcal{A})$ the non-parametric structure where all occurrences of a parameter p_i have been replaced by $v(p_i)$. We denote as a *timed automaton* any structure $v(\mathcal{A})$.

The *synchronous product* (using strong broadcast, i. e., synchronization on a given set of actions) of several PTAs gives a PTA.

Definition 2 (synchronized product of PTAs)**.** Let $N \in \mathbb{N}$. Given a set of PTAs $\mathcal{A}_i = (\Sigma_i, L_i, (\ell_0)_i, \mathbb{X}_i, \mathbb{P}_i, I_i, E_i)$, $1 \leq i \leq N$, and a set of actions Σ_s, the *synchronized product* of $\mathcal{A}_i$, $1 \leq i \leq N$, denoted by $\mathcal{A}_1 \parallel_{\Sigma_s} \mathcal{A}_2 \parallel_{\Sigma_s} \cdots \parallel_{\Sigma_s} \mathcal{A}_N$, is the tuple $(\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, E)$, where:

1) $\Sigma = \bigcup_{i=1}^{N} \Sigma_i$,
2) $L = \prod_{i=1}^{N} L_i$, $\ell_0 = ((\ell_0)_1, \ldots, (\ell_0)_N)$,
3) $\mathbb{X} = \bigcup_{1 \leq i \leq N} \mathbb{X}_i$, $\mathbb{P} = \bigcup_{1 \leq i \leq N} \mathbb{P}_i$,
4) $I((\ell_1, \ldots, \ell_N)) = \bigwedge_{i=1}^{N} I_i(\ell_i)$ for all $(\ell_1, \ldots, \ell_N) \in L$,

and E is defined as follows. For all $a \in \Sigma$, let ζ_a be the subset of indices $i \in 1, \ldots, N$ such that $a \in \Sigma_i$. For all $a \in \Sigma$, for all $(\ell_1, \ldots, \ell_N) \in L$, for all $(\ell'_1, \ldots, \ell'_N) \in L$, $\big((\ell_1, \ldots, \ell_N), g, a, R, (\ell'_1, \ldots, \ell'_N)\big) \in E$ if:

- if $a \in \Sigma_s$, then 1) for all $i \in \zeta_a$, there exist g_i, R_i such that $(\ell_i, g_i, a, R_i, \ell'_i) \in E_i$, $g = \bigwedge_{i \in \zeta_a} g_i$, $R = \bigcup_{i \in \zeta_a} R_i$, and, 2) for all $i \notin \zeta_a$, $\ell'_i = \ell_i$.
- otherwise (if $a \notin \Sigma_s$), then there exists $i \in \zeta_a$ such that 1) there exist g_i, R_i such that $(\ell_i, g_i, a, R_i, \ell'_i) \in E_i$, $g = g_i$, $R = R_i$, and, 2) for all $j \neq i$, $\ell'_j = \ell_j$.

That is, synchronization is only performed on Σ_s, and other actions are interleaved. When Σ_s is not specified, it is assumed to be equal to the intersection of the sets of actions. That is, given $\mathcal{A}_1$ over Σ_1 and $\mathcal{A}_2$ over Σ_2, $\mathcal{A}_1 \parallel \mathcal{A}_2$ denotes $\mathcal{A}_1 \parallel_{\Sigma_s} \mathcal{A}_2$ where $\Sigma_s = \Sigma_1 \cap \Sigma_2$.

Definition 3 (Semantics of a TA)**.** Given a PTA $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, E)$, and a parameter valuation v, the semantics of $v(\mathcal{A})$ is given by the timed transition system (TTS) $(S, s_0, \to)$, with

- $S = \{(\ell, \mu) \in L \times \mathbb{R}^H_{\geq 0} \mid \mu \models v(I(\ell))\}$, $s_0 = (\ell_0, \vec{0})$,
- $\to$ consists of the discrete and (continuous) delay transition relations: *i*) discrete transitions: $(\ell, \mu) \overset{e}{\mapsto} (\ell', \mu')$, if $(\ell, \mu), (\ell', \mu') \in S$, and there exists $e = (\ell, g, a, R, \ell') \in E$, such that $\mu' = [\mu]_R$, and $\mu \models v(g)$. *ii*) delay transitions: $(\ell, \mu) \overset{d}{\mapsto} (\ell, \mu + d)$, with $d \in \mathbb{R}_{\geq 0}$, if $\forall d' \in [0, d], (\ell, \mu + d') \in S$.

We write $(\ell, \mu) \overset{(e,d)}{\longrightarrow} (\ell', \mu')$ for a combination of a delay and discrete transition if $\exists \mu'' : (\ell, \mu) \overset{d}{\mapsto} (\ell, \mu'') \overset{e}{\mapsto} (\ell', \mu')$.

Given a TA $v(\mathcal{A})$ with concrete semantics $(S, s_0, \to)$, we refer to the states of S as the *concrete states* of $v(\mathcal{A})$. A *run* of $v(\mathcal{A})$ is an alternating sequence of concrete states of $v(\mathcal{A})$ and pairs of edges and delays starting from the initial state s_0 of the form $s_0, (e_0, d_0), s_1, \cdots$ with $i = 0, 1, \ldots$, $e_i \in E$, $d_i \in \mathbb{R}_{\geq 0}$ and $s_i \overset{(e_i,d_i)}{\longrightarrow} s_{i+1}$. Given $s = (\ell, \mu)$, we say that s is reachable in $v(\mathcal{A})$ if s appears in a run of $v(\mathcal{A})$. By extension, we say that ℓ is reachable; and by extension again, given a set T of locations, we say that T is reachable if there exists $\ell \in T$ such that ℓ is reachable in $v(\mathcal{A})$. We denote by $\mathsf{Loc}(v(\mathcal{A}))$ the set of all locations reachable in $v(\mathcal{A})$.

Example 2. Consider again the PTA $\mathcal{A}$ in Fig. 1. Let v_1 be such that $v_1(p) = 1$. Then, ℓ_2 is unreachable in $v_1(\mathcal{A})$: at $x = 1$, one can take a first time the self-loop over ℓ_0, yielding $y = 1$ and $x = 0$. The guard $y > 2$ to ℓ_2 is not yet satisfied.

Then at $x = 1$, one can take a second time the self-loop over ℓ_0, yielding $y = 2$ and $x = 0$. The guard $y > 2$ to ℓ_2 is still not satisfied. At $x = 1$, the guard $y < 3$ is not satisfied anymore, and the self-loop over ℓ_0 cannot be taken anymore. Therefore, $\mathsf{Loc}(v_1(\mathcal{A})) = \{\ell_0, \ell_1\}$.

Let v_2 be such that $v_2(p) = 0.9$. This time, ℓ_2 is reachable, by taking three times the self-loop over ℓ_0 when $y = 0.9$, $y = 1.8$ and $y = 2.7$ respectively. Therefore, $\mathsf{Loc}(v_2(\mathcal{A})) = \{\ell_0, \ell_1, \ell_2\}$.

B. Reachability synthesis

We will use reachability synthesis to solve the problem in Section III. This procedure, called EFsynth, takes as input a PTA $\mathcal{A}$ and a set of target locations T, and attempts to synthesize all parameter valuations v for which T is reachable in $v(\mathcal{A})$. $\mathsf{EFsynth}(\mathcal{A}, T)$ was formalized in e. g., [JLR15] and is a procedure that may not terminate, but that computes an exact result (sound and complete) if it terminates. EFsynth traverses the *parametric zone graph* of $\mathcal{A}$, which is a potentially infinite extension of the well-known zone graph of TAs (see, e. g., [And+09; JLR15]).

Example 3. Consider again the PTA $\mathcal{A}$ in Fig. 1. Let us compute the set of parameter valuations for which ℓ_2 is reachable. $\mathsf{EFsynth}(\mathcal{A}, \{\ell_2\}) = 0 < p < 1 \vee 1 < p < 1.5 \vee 2 < p < 3$. Intuitively, whenever $p \in (0, 1)$, one can take multiple times the self-loop over ℓ_0 so that eventually the guard $y > 2 \wedge x = 0$ is satisfied; whenever $p \in (1, 1.5)$, one can take exactly twice the self-loop over ℓ_0 so that the guard to ℓ_2 is satisfied; whenever $p \in (2, 3)$, one takes a single time the self-loop over ℓ_0, and then the guard to ℓ_2 becomes satisfied. For other valuations, there is no way to reach ℓ_2.

Remark 1. EFsynth can also be used to compute *unreachability* (or *safety*) synthesis, by taking the *negation* (i. e., the complement of the valuations set) of the result.

Example 4. Consider again the PTA $\mathcal{A}$ in Fig. 1. Let us compute the set of parameter valuations for which ℓ_2 is unreachable. $\neg\mathsf{EFsynth}(\mathcal{A}, \{\ell_2\})$ is $p = 0 \vee p = 1 \vee p \in [1.5, 2] \vee p \geq 3$.

C. Non-interference

Often, non-interference is defined using a set of low-level actions and a set of high-level ones. The idea is that an intruder is allowed to perform some high-level actions. The non-interference property is satisfied whenever the system behavior in absence of high level actions is equivalent to its behavior, observed on low level actions, when high level actions occur [BT03].

In the following, we assume a set of low-level actions L and a set of high-level actions H.

Definition 4 (restriction)**.** Let $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, E)$ be a PTA with $\Sigma = L \uplus H$ ($\uplus$ denotes disjoint union), and v be a parameter valuation. The *restriction of* $v(\mathcal{A})$ *to low-level actions*, denoted by $v(\mathcal{A})_{|L}$, is defined as the automaton identical to $v(\mathcal{A})$ except that any edge of the form (ℓ, g, a, R, ℓ') with $a \in H$ is discarded.

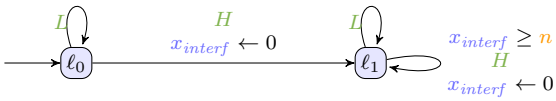


Figure 2: PTA $\mathcal{I}nterf_H^n$ [BT03]

Definition 5 (hiding)**.** Let $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, E)$ be a PTA with $\Sigma = L \uplus H$, and v be a parameter valuation. The *hiding of high-level actions in* $v(\mathcal{A})$, denoted by $v(\mathcal{A})_{\backslash H}$, is defined as the automaton identical to $v(\mathcal{A})$ except that any edge of the form (ℓ, g, a, R, ℓ') with $a \in H$ is replaced with an edge $(\ell, g, \epsilon, R, \ell')$.

ϵ is the special silent action.

In [BT03], n-non-interference is defined as a concept ensuring that the low level behavior is unaffected by attacks which are separated by more than n time units. An attack is a high-level action decided by the attacker. This concept helps to quantify the necessary *attacking speed* of the attacker.

Let us now recall from [BT03] the (P)TA $\mathcal{I}nterf_H^n$ in Fig. 2, where x_{interf} is a local clock only used in $\mathcal{I}nterf_H^n$, and where L (resp. H) denotes any transition labeled with an action $a \in L$ (resp. $a \in H$). The idea is that this PTA allows the execution of high-level actions only when they are separated by at least n time units. Note that, in our setting, n can be a timing parameter.

We now recall the concept of *location-non-interference* (called *state-non-interference* in [BT03]) that checks whether the set of locations (discrete states) reachable in the original automaton is identical to the set of locations reachable in the hiding[1] of H in the product of the original automaton with $\mathcal{I}nterf_H^n$.

Definition 6 (n-location-non-interference)**.** Let $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, E)$ be a PTA with $\Sigma = L \uplus H$, and v be a parameter valuation. Let $n \in \mathbb{Q}_+$. $v(\mathcal{A})$ is *n-location-non-interfering* if $\mathsf{Loc}\big(v(\mathcal{A})_{|L}\big)$ is equal to $\mathsf{Loc}\big((v(\mathcal{A}) \parallel \mathcal{I}nterf_H^n)_{\backslash H}\big)$ projected onto the locations of $v(\mathcal{A})$.

By "projected on $v(\mathcal{A})$", we mean the set $\{\ell \mid \exists \ell' : (\ell, \ell') \in \mathsf{Loc}\big((v(\mathcal{A}) \parallel \mathcal{I}nterf_H^n)\big)\}$. In Definition 6, the system is n-location-non-interfering if an intruder with the ability to disturb the system at most every n time units is not able to modify the set of reachable locations. Since the set of reachable locations is computable for TAs [AD94], n-location-non-interference (for a given n) is decidable for TAs [BT03].

Example 5. Consider again the PTA $\mathcal{A}$ in Fig. 1. Assume $L = \{l\}$ and $H = \{h\}$. That is, the intruder can take the self-loop over ℓ_0. Let v_3 be such that $v_3(p) = 1.1$. First, note that $\mathsf{Loc}\big(v_3(\mathcal{A})_{|L}\big) = \{\ell_0, \ell_1\}$ since the transition to ℓ_2 is syntactically removed, preventing x to be reset.

Fix $n = 1$. The product of $v_3(\mathcal{A})$ with $\mathcal{I}nterf_H^n$ prevents the system to synchronize faster than every 1 time unit on h:

[1]The hiding of H is not strictly speaking necessary in our setting since we are interested in the reachability of *locations* but we keep it for sake of consistency with [BT03].

therefore, taking the self-loop labeled with h when $y = 1.1$ and $y = 2.2$ respectively is possible, enabling the transition to ℓ_2 at $y = 2.2$. This gives that $\mathsf{Loc}\big((v_3(\mathcal{A}) \parallel \mathcal{I}nterf^n_H)_{\setminus H}\big)$ projected onto the locations of $v_3(\mathcal{A})$ is $\{\ell_0, \ell_1, \ell_2\}$. Therefore, $v_3(\mathcal{A})$ is not 1-location-non-interfering.

Now fix $n = 2$. In that case, the self-loop can be taken when $y = 1.1$, but not when $y = 2.2$ because the condition $x_{interf} \geq 2$ is not satisfied (recall that x_{interf} is reset in $\mathcal{I}nterf^n_H$ on the first transition labeled with h, and therefore we have $x_{interf} = 1.1$ when $y = 2.2$). So ℓ_2 is unreachable. Therefore, $v_3(\mathcal{A})$ is 2-location-non-interfering.

III. Parametric location-non-interference

In this work, we aim at considering a broader problem: instead of asking whether the intruder with a predefined power can disturb the system, we ask what is the power the intruder needs to perform a successful attack? More precisely, we aim at computing the speed of the intruder needed to successfuly disturb the system: that is, for what valuations of n is the system (not) n-location-non-interfering?

In addition, our PTA model can contain free parameters too; so the parameter is not only n but also the PTA parameters.

n-location-non-interference synthesis problem:
INPUT: A PTA $\mathcal{A}$ with parameters $\mathbb{P}$, a parameter n
PROBLEM: Synthesize valuations v of $\mathbb{P}$ and of n such that $v(\mathcal{A})$ is n-location-non-interfering.

Since our problem is *location*-based, we can solve it using reachability synthesis techniques for PTAs, more precisely using EFsynth. The core idea is to synthesize valuations of $\mathbb{P} \cup \{n\}$ such that the set of reachable locations remains identical in both $v(\mathcal{A})_{|L}$ and $(v(\mathcal{A}) \parallel \mathcal{I}nterf^n_H)_{\setminus H}$. This therefore reduces to a reachability synthesis problem.

Note that, since reachability-emptiness (i. e., the emptiness of the valuations set for which a given (set of) location(s) is reachable) is undecidable for PTAs [AHV93; And19], reachability synthesis algorithms are not guaranteed to terminate. (We discuss approximations later on.)

Example 6. Consider again the PTA $\mathcal{A}$ in Fig. 1. Assume $L = \{l\}$ and $H = \{h\}$. First observe that ℓ_1 (and of course ℓ_0) can be reached in $v(\mathcal{A})$ regardless of the value of p. Second, for all v, ℓ_2 is unreachable in $v(\mathcal{A})_{|L}$ since the self-loop on ℓ_0 is syntactically removed. Therefore, for all v, $\mathsf{Loc}\big(v(\mathcal{A})_{|L}\big) = \{\ell_0, \ell_1\}$.

As a consequence, n-location-non-interference synthesis for $\mathcal{A}$ reduces to unreachability synthesis of valuations of n and p for which ℓ_2 is unreachable.

The result of $\neg\mathsf{EFsynth}\big((\mathcal{A} \parallel \mathcal{I}nterf^n_H)_{\setminus H}, \{\ell_2\}\big)$ is:

$$
\begin{array}{ll}
 & (0 < p < 1 \wedge n > p) \\
\vee & (p = 1 \wedge n \geq 0) \\
\vee & (1 < p < 1.5 \wedge n > p) \\
\vee & (1.5 \leq p \leq 2 \wedge n \geq 0) \\
\vee & (p \geq 3 \wedge n \geq 0)
\end{array}
$$

That is, for any valuation of p and n within this constraint, the system is n-location-non-interfering, i. e., the intruder cannot impact the set of reachable locations.

This result can be intuitively explained as follows: whenever $p < 1$ (first disjunct), if the intruder can act strictly slower than every p time unit ($n > p$), only one self-loop on ℓ_0 can be taken, and ℓ_2 is unreachable, and therefore the system is n-location-non-interfering. Whenever $p = 1$ (second disjunct) or $1.5 \leq p \leq 2$ (4th disjunct) or $p \geq 3$ (last disjunct), we saw in Example 3 that ℓ_2 is unreachable, regardless of the value of n. Finally, whenever $1 < p < 1.5$ (3rd disjunct), ℓ_2 is reachable iff the intruder can act strictly slower than every p time units.

IV. Application to the Fischer protocol

A. The Fischer mutual exclusion protocol

The protocol proposed by Michael Fischer ensures that two processes never use a critical resource (often denoted by critical section) at the same time. The protocol is based on the speed of the processes.

We consider here the protocol as studied in [BT03]: suppose that two processes P_1 and P_2 running in parallel compete for the critical section. Assume that atomic reads and writes are permitted to a shared variable `v` (called `x` in [BT03]). Assume also that every access to the shared memory containing `v` takes acc units of time. Each process i executes the following code:

```
repeat
    await v = 0
    v := i
    delay b
until v = i
v := 0
(Critical section)
```

The idea is that process P_i is allowed into the critical section only when `v` $= i$; "`await v=0`" waits until `v` becomes 0; "`delay b`" waits exactly b time units, which is ensured by the process local clock. An assignment takes (at most) a time units. The modified model of [BT03] also considers that the maximum time needed to execute the critical section after `v` is checked is ucs.

In [BT03], the protocol is modeled using a network $\mathcal{A}$ of TAs made of *i*) process P_i, for $i = 1, 2$, *ii*) an intruder, that can take an att transition anytime, nondeterministically changing `v` to 0, 1 or 2, and *iii*) a "serializer" responsible to model the value of `v` according to the access and modification requests of P_1 and P_2; recall that these operations take acc time units.

The idea is as follows: if the intruder is fast enough, (s)he can successfully disturb the system, i. e., send both processes into the critical section at the same time, thus violating the mutual exclusion. On the contrary, if the intruder is slow, its nondeterministic modifications of `v` will have no effect on the security.

The crux of [BT03] is that the model is n-location-non-interfering iff the previously defined TA $\mathcal{A}$ cannot reach a location where both processes are in the critical section at the same time. A sufficient condition over n and b to ensure n-location-non-interference is manually inferred and proved.

Our goal is to automatically infer conditions over n, a, b, acc and ucs guaranteeing n-location-non-interference.

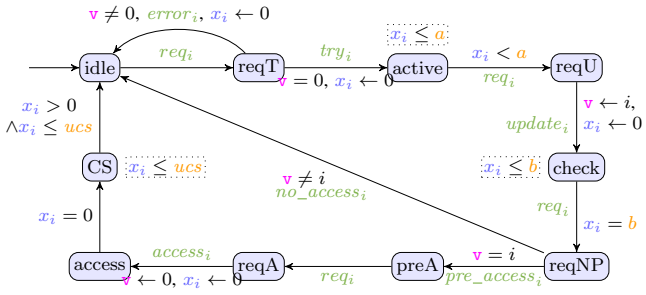


Figure 3: Modified model of process P_i for $i = 1, 2$

B. A modified Fischer model

1) An issue in the existing modeling: While modeling the TA model of [BT03] using a model checker, we spotted a modeling issue, that makes the model wrong. The issue lies in the serializer: the idea of the serializer is to ensure that two consecutive access or modification requests to v are separated by at least acc time units; but, due to the absence of invariants and of "as soon as possible" concept in the TAs of [BT03], there exist runs of the TA such that the access to v can last forever, even if no other process is competing. Although this cannot happen in reality, the model checker always reports that both processes can end up in the critical sections, whatever the values of b and n are.

Note that this mistake does not impact the definitions nor the overall reasoning of the benchmark application of [BT03], as the authors use a *manual* reasoning to infer the condition over n and b.

2) Our new model: The purpose of the serializer is both to encode the value of v, and to maintain a "queue" of requests to read or write accesses to the memory, ensuring that any two consecutive access is separated by acc time units. We therefore entirely rewrote the serializer, also impacting the model of the processes.

a) Processes: We first give the modified model of process P_i in Fig. 3. Process P_i features a local clock x_i and can read or write v. The main modification w.r.t. to [BT03, Fig. 6] is the duplication of all locations: indeed, instead of performing a single action (e. g., try_i) reading or writing v, the process first performs a *request* req_i, then followed by the action try_i; the serializer is responsible for answering the request as soon as possible, but not earlier than acc since the latest read or write action. Then, the PTA follows the program given in Section IV-A: P_i first waits until $v = 0$, then updates it to i; then, it waits exactly b time units, and checks whether v is still i; if not, it moves back to the original location. If $v = i$, the process sets v to 0, enters the critical section and, after at most ucs time units, leaves it to go back to the idle location.

b) Intruder: The intruder is almost identical to the one in [BT03]: it is a one-location PTA with three self-loops synchronizing on att and setting v to 0, 1 or 2 respectively (given in Fig. 4). Setting $H = \{att\}$, the synchronization

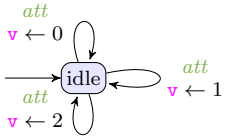


Figure 4: Model of the intruder

of the intruder with $\mathcal{I}nterf_H^n$ ensures that the intruder may modify v with at least n time units since its last modification.

c) Serializer: The seralizer automaton is both very simple and quite complex. Its overall goal is very simple, and is to perform the following behavior: "whenever a process (P_1, P_2 or the intruder) requests a write or read access to the memory, the serializer shall grant it as soon as possible, and acc time units later than the previous access—unless another process is also requesting access, in which case the request should be queued". Implementing this in a purely automata-based formalism is however cumbersome, much more than the simplistic serializer of [BT03], that does not encompass for the "as soon as possible" concept. This results in a PTA made of 12 locations and 91 transitions.[2]

V. Experiments

We model the aforementioned automata using the IMITATOR [And+12] parametric timed model checker (version 2.12 "Butter Lobster"). We then run safety synthesis (i. e., the negation of EFsynth, implemented in IMITATOR) so as to synthesize parameter valuations for which mutual exclusion is guaranteed, i. e., both processes cannot be in the CS location at the same time.

A. An approximated result

Running IMITATOR on the model, the analysis does not terminate; this is not surprising as EFsynth is not guaranteed to terminate due to the undecidability of the reachability-emptiness for PTAs [AHV93]. A closer look at the analysis let us realize that, after passing the depth of 24 (IMITATOR explores the zone graph in a breadth-first search manner), no new constraint is synthesized, until at least a depth of 1000 (after which we interrupted the analysis, after about 20 hours of processing). The resulting constraint (that does not change after depth 24, reached in about 650 s) is made of 22 disjunctions of convex constraints over the system parameters.

A property of EFsynth is that it returns an under-approximation of the constraint when interrupted; when its negation (safety synthesis) is run, an *over-approximation* is returned. That is to say, the obtained result contains all possible valuations for which non-interference is satisfied, but the result may also potentially contain valuations for which the system is *not* non-interfering. We therefore *tested* valuations from our constraint using the non-parametric timed model checker Uppaal [LPY97]. We randomly picked up several dozens of parameter valuations, and checked using Uppaal

[2]All models and results are available at www.imitator.fr/static/ICECCS20.

that non-interference is satisfied iff the valuation belongs to the resulting constraint. The UPPAAL model is identical to the IMITATOR model, but is non-parametric, and therefore the analysis is guaranteed to terminate (depending on the parameter valuations, termination is obtained within a few seconds). This does not formally prove that our result is exact (sound and complete), but increases the degree of confidence. Proving the exactness of our constraint (or developing a new synthesis algorithm able to detect that the constraint is exact and to terminate the analysis) is among our future work.

B. Interpretation

The 22 disjunctions of convex constraints give a set of conditions for which the system is non-interfering, that is the mutual exclusion is guaranteed *and* an attacker able to disturb the system at most every n time units cannot succeed in violating mutual exclusion by its actions.

For sake of exemplification, let us consider the first disjunct (the full constraint is available online):

$$\begin{array}{ll} & n \geq 0 \\ \wedge & b \geq acc + n \\ \wedge & b \geq 3 \times acc \\ \wedge & a > 0 \\ \wedge & acc > ucs > 0 \end{array}$$

Recall that b is the waiting time before testing again the value of $\mathtt{v}$, n is the minimum time between any two consecutive high-level actions (of the intruder), acc is the memory access time, and ucs is an upper bound on the time during which a process remains in the critical section. This constraint ensures that mutual exclusion is guaranteed even when an attacker can change the value of $\mathtt{v}$ no faster than every n time units if the following conditions are satisfied:

- the delay (b) is longer than the access time (acc) and the minimum disturbance time (n); that is to say, even when the intruder modifies the system, the process can still detect it as its delay is long enough; this helps guaranteeing non-interference;
- the delay is longer than three access times ($3 \times acc$); that is to say, the delay is long enough to detect whether the other process performs try_i, $update_i$ and pre_access_i during the delay; this helps guaranteeing validity of the mutual exclusion;
- and the memory access time is longer than the time during which a process remains in the critical section.

Further sufficient conditions ensuring non-interference are guaranteed by the other disjuncts (see Web page).

VI. CONCLUSION

a) Conclusion: We introduced a definition of n-location-non-interference, that aims at quantifying the necessary attacker frequency to be able to modify the set of reachable locations in a timed automaton. Using the IMITATOR parametric timed model checker, we obtained preliminary results on an improved version of the Fischer mutual exclusion protocol.

b) Future works: As we only obtained an over-approximation of the result, our first future work is to prove the exactness (soundness and completeness) of the obtained constraint, either by proving it using an *ad-hoc* reasoning for our case study, or by developing new automated techniques allowing IMITATOR to terminate as soon as the constraint is indeed complete. Alternatively, designing approximated techniques is another interesting direction.

While the general emptiness problem (the emptiness of the set of both timing parameter valuations *and* admissible values of n for which the system is n-location-non-interfering) is very likely to be undecidable (due to the undecidability of reachability emptiness in [AHV93]), the more specific problem of deciding whether there exists a valuation of n for which the (non-parametric) system is n-location-non-interfering remains open. Also, we aim at tackling efficient *synthesis* of these sets, independently of the decidability issues. More generally, proposing new state space reduction techniques dedicated to the problem of n-location-non-interference is among our future works.

REFERENCES

[AD94] Rajeev Alur and David L. Dill. "A theory of timed automata". In: *Theoretical Computer Science* 126.2 (Apr. 1994), pp. 183–235. ISSN: 0304-3975. DOI: 10.1016/0304-3975(94)90010-8 (cit. on pp. 1, 3).

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. "Parametric real-time reasoning". In: *STOC*. (May 16–18, 1993). Ed. by S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal. San Diego, California, United States: ACM, 1993, pp. 592–601. ISBN: 0-89791-591-7. DOI: 10.1145/167088.167242 (cit. on pp. 1, 4–6).

[And+09] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. "An Inverse Method for Parametric Timed Automata". In: *International Journal of Foundations of Computer Science* 20.5 (Oct. 2009), pp. 819–836. DOI: 10.1142/S0129054109006905 (cit. on p. 3).

[And+12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. "IMITATOR 2.5: A Tool for Analyzing Robustness in Scheduling Problems". In: *FM*. (Aug. 27–31, 2012). Ed. by Dimitra Giannakopoulou and Dominique Méry. Vol. 7436. Lecture Notes in Computer Science. Paris, France: Springer, Aug. 2012, pp. 33–36. DOI: 10.1007/978-3-642-32759-9_6 (cit. on pp. 1, 5).

[And19] Étienne André. "What's decidable about parametric timed automata?" In: *International Journal on Software Tools for Technology Transfer* 21.2 (Apr. 2019), pp. 203–219. DOI: 10.1007/s10009-017-0467-0 (cit. on p. 4).

[AS19] Étienne André and Jun Sun. "Parametric Timed Model Checking for Guaranteeing Timed Opacity". In: *ATVA*. (Oct. 28–31, 2019). Ed. by Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza. Vol. 11781. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, 2019, pp. 115–130. DOI: 10.1007/978-3-030-31784-3_7 (cit. on p. 1).

[Bar+02] Roberto Barbuti, Nicoletta De Francesco, Antonella Santone, and Luca Tesei. "A Notion of Non-Interference for Timed Automata". In: *Fundamenta Informaticae* 51.1-2 (2002), pp. 1–11 (cit. on p. 1).

[Ben+15] Gilles Benattar, Franck Cassez, Didier Lime, and Olivier H. Roux. “Control and synthesis of non-interferent timed systems”. In: *International Journal of Control* 88.2 (2015), pp. 217–236. DOI: 10.1080/00207179.2014.944356 (cit. on p. 1).

[BT03] Roberto Barbuti and Luca Tesei. “A Decidable Notion of Timed Non-Interference”. In: *Fundamenta Informaticae* 54.2-3 (2003), pp. 137–150 (cit. on pp. 1, 3–5).

[Cas09] Franck Cassez. “The Dark Side of Timed Opacity”. In: *ISA*. (June 25–27, 2009). Ed. by Jong Hyuk Park, Hsiao-Hwa Chen, Mohammed Atiquzzaman, Changhoon Lee, Tai-Hoon Kim, and Sang-Soo Yeo. Vol. 5576. Lecture Notes in Computer Science. Seoul, Korea: Springer, 2009, pp. 21–30. DOI: 10.1007/978-3-642-02617-1_3 (cit. on p. 1).

[GMR07] Guillaume Gardey, John Mullins, and Olivier H. Roux. “Non-Interference Control Synthesis for Security Timed Automata”. In: *Electronic Notes in Theoretical Computer Science* 180.1 (2007), pp. 35–53. DOI: 10.1016/j.entcs.2005.05.046 (cit. on p. 1).

[JLR15] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. “Integer Parameter Synthesis for Real-Time Systems”. In: *IEEE Transactions on Software Engineering* 41.5 (2015), pp. 445–461. DOI: 10.1109/TSE.2014.2357445 (cit. on p. 3).

[Koc96] Paul C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *CRYPTO*. (Aug. 18–22, 1996). Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Santa Barbara, California, USA: Springer, 1996, pp. 104–113. DOI: 10.1007/3-540-68697-5_9 (cit. on p. 1).

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. “UPPAAL in a Nutshell”. In: *International Journal on Software Tools for Technology Transfer* 1.1-2 (1997), pp. 134–152. DOI: 10.1007/s100090050010 (cit. on p. 5).

[NNV17] Flemming Nielson, Hanne Riis Nielson, and Panagiotis Vasilikos. “Information Flow for Timed Automata”. In: *Models, Algorithms, Logics and Tools*. Ed. by Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna Ingólfsdóttir, Axel Legay, and Radu Mardare. Vol. 10460. Lecture Notes in Computer Science. Springer, 2017, pp. 3–21. DOI: 10.1007/978-3-319-63121-9_1 (cit. on p. 1).

[VNN18] Panagiotis Vasilikos, Flemming Nielson, and Hanne Riis Nielson. “Secure Information Release in Timed Automata”. In: *POST*. (Apr. 14–20, 2018). Ed. by Lujo Bauer and Ralf Küsters. Vol. 10804. Lecture Notes in Computer Science. Thessaloniki, Greece: Springer, 2018, pp. 28–52. DOI: 10.1007/978-3-319-89722-6_2 (cit. on p. 1).

Guaranteeing Timed Opacity using Parametric Timed Model Checking

ÉTIENNE ANDRÉ, Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
DIDIER LIME, École Centrale de Nantes, LS2N, UMR CNRS 6004, Nantes, France
DYLAN MARINHO, Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
JUN SUN, School of Information Systems, Singapore Management University, Singapore

Information leakage can have dramatic consequences on systems security. Among harmful information leaks, the timing information leakage occurs whenever an attacker successfully deduces confidential internal information. In this work, we consider that the attacker has access (only) to the system execution time. We address the following timed opacity problem: given a timed system, a private location and a final location, synthesize the execution times from the initial location to the final location for which one cannot deduce whether the system went through the private location. We also consider the full timed opacity problem, asking whether the system is opaque for all execution times. We show that these problems are decidable for timed automata (TAs) but become undecidable when one adds parameters, yielding parametric timed automata (PTAs). We identify a subclass with some decidability results. We then devise an algorithm for synthesizing PTAs parameter valuations guaranteeing that the resulting TA is opaque. We finally show that our method can also apply to program analysis.

CCS Concepts: • **Security and privacy → Logic and verification**; • **Theory of computation → Quantitative automata**; **Verification by model checking**; **Logic and verification**.

Additional Key Words and Phrases: opacity, timed automata, IMITATOR, parameter synthesis.

ACM Reference Format:
Étienne André, Didier Lime, Dylan Marinho, and Jun Sun. 2022. Guaranteeing Timed Opacity using Parametric Timed Model Checking. *ACM Trans. Softw. Eng. Methodol.* 37, 4, Article 111 (August 2022), 37 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Timed systems often combine hard real-time constraints with other complications such as concurrency. Information leakage can have dramatic consequences on the security of such systems. Among harmful information leaks, the *timing information leakage* is the ability for an attacker to deduce internal information depending on timing information. In this work, we focus on timing leakage through the total execution time, i. e., when a system works as an almost black-box and the ability of the attacker is limited to know the model and observe the total execution time.

Authors' addresses: Étienne André, Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France; Didier Lime, École Centrale de Nantes, LS2N, UMR CNRS 6004, Nantes, France; Dylan Marinho, Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France; Jun Sun, School of Information Systems, Singapore Management University, Singapore.

1049-331X/2022/8-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

Opacity. In its most general form on partially observed labeled transitions systems, given a set of runs that reveal a secret (e. g., they perform a secret action or visit a secret state), *opacity* states that if there exists a run of the system that reveals the secret (i. e., belongs to the given secret set), there exists another run, with the same observation, that does not reveal that secret [1]. This secret is completely generic and, depending on its actual definition, properties and their decidability can differ.

In our setting, we define a form of opacity in which the observation is only *the time to reach a designated location.*

Contributions for TAs. In this work, we consider the setting of timed automata (TAs), which is a popular extension of finite-state automata with clocks [2]. We consider the following version of timed opacity: given a TA, a private location denoting the execution of some secret behavior and a final location denoting the completion of the execution, the TA is opaque for a given execution time d (i. e., the time of a run from the initial location to the final location) if there exist two runs of duration d from the initial location to the final location, one going through the private location, and another run *not* going through the private location. That is, for this particular execution time, the system is opaque if one cannot deduce whether the system went through the private location. Such a notion of timed opacity can be used to capture many interesting security problems: for instance, it is possible to deduce whether a secret satisfies a certain condition based on whether a certain branch is visited or not.

To be explicit, the attacker knows a TA model of the system, and can observe the execution time from the system start until it reaches some particular final location. No other actions can be observed. Then, the system is timed opaque if the attacker cannot deduce whether the system has visited some particular private location. From a higher-level point of view, this means that the attacker cannot deduce some private information, such as whether some location has been visited, or whether some branch of a given program was visited, by only observing the execution time. In practice, this corresponds to a setting where the attacker may interact with some computational process on a remote machine (e. g., a server) and receives the responses only at the end of the process (e. g., a server message is received).

We consider two problems based on this notion of timed opacity:

(1) a computation problem: the computation of the set of possible execution times for which the system is timed opaque; and
(2) a decision problem: whether the TA is timed opaque for all execution times (referred to as full timed opacity).

We first prove that these problems can be effectively solved for TAs. We implement our procedure and apply it to a set of benchmarks containing notably a set of Java programs known for their (absence of) timing information leakage.

Contributions for parametric TAs. As a second setting, we consider a higher-level version of these problems by allowing (internal) timing parameters in the system, which can model uncertainty or unknown constants at early design stage. The setting becomes parametric timed automata (PTAs) [3].

On the theoretical side, we answer an existential parametric version of the two aforementioned problems, that is, the existence of (at least) one parameter valuation for which the TA is (fully) timed opaque. Although we show that these problems are in general undecidable, we exhibit a subclass with some decidability results.

Then, we address a practical problem: given a timed system with timing parameters, a private location and a final location, synthesize the timing parameters and the execution times for which

one cannot deduce whether the system went through the private location. We devise a general procedure not guaranteed to terminate, but that behaves well on examples from the literature.

Summary of the contributions. To sum up, this manuscript proposes the following contributions:

(1) a notion of timed opacity, and a notion of full timed opacity for TAs;
(2) a procedure to solve the timed opacity computation problem for TAs, and a procedure to answer the full timed opacity decision problem for TAs;
(3) a study of two theoretical decision problems extending the two aforementioned problems to the parametric setting, and exhibition of a decidable subclass;
(4) a practical algorithm to synthesize parameter valuations and execution times for which the TA is guaranteed to be opaque;
(5) a set of experiments on a set of benchmarks, including PTAs translations from Java programs.

This manuscript is an extension of [4] with the following improvements.

- We provide all proofs of the results published in [4].
- We extend the theoretical part, by considering not one problem (as in [4]) but two versions (timed opacity w.r.t. a set of execution times, and full timed opacity), both for TAs and PTAs (including the subclass of L/U-PTAs).
- We propose a more elegant proof of Proposition 5.2 (formerly [4, Proposition 1]), based on RA arithmetic [5].
- On the practical side, we give hints to extend our construction to a richer framework (Section 9.3).

Outline. After reviewing related works in Section 2, Section 3 recalls necessary concepts and Section 4 introduces the problem. Section 5 addresses timed opacity for timed automata. We then address the parametric version of timed opacity, with theory studied in Sections 6 and 7, algorithmic in Section 8 and experiments in Section 9. Section 10 concludes the paper.

2 RELATED WORKS

Opacity and timed automata. This work is closely related to the line of work on defining and analyzing information flow in timed automata. It is well-known (see e. g., [6, 7, 8, 9, 10]) that time is a potential attack vector against secure systems. That is, it is possible that a non-interferent (secure) system can become interferent (insecure) when timing constraints are added [11].

In *non-interference*, actions are partitioned into two levels of privilege, *high* and *low*, and we require that the system in which high-level actions are removed is equivalent to the system in which they are hidden (i. e., replaced by an unobservable action). Different equivalences lead to different flavors of non-interference. In [12, 13], a first notion of *timed* non-interference is proposed for TAs. This notion is extended to PTAs in [14], with a semi-algorithm.

In [11], Gardey *et al.* define timed strong non-deterministic non-interference (SNNI) based on timed language equivalence between the automaton with hidden low-level actions and the automaton with removed low-level actions. Furthermore, they show that the problem of determining whether a timed automaton satisfies SNNI is undecidable. In contrast, timed cosimulation-based SNNI, timed bisimulation-based SNNI and timed state SNNI are decidable. Classical SNNI is the one corresponding to the equality of the languages of the two systems. As such it is clearly a special case of opacity in which the secret runs are those containing a high-level action [1]. Other equivalence relations (namely (timed) cosimulation, (timed) bisimulation, sets of states) are not as easily relatable to opacity. No implementation is provided in [11].

In [15], it is proved that it is undecidable whether a TA is opaque, for the following definition of opacity: the system is opaque if an attacker cannot deduce whether some set of actions was

performed, by only observing a given set of observable actions together with their timestamp. This problem is proved undecidable even for the restricted class of event-recording automata [16], which is a subclass of TAs. No implementation nor procedure is provided. In contrast, our definition of opacity is decidable for TAs, notably because in our setting the attacker power is more restricted (they can only observe the "execution time"); in addition, our definition of opacity has some practical relevance nonetheless, when an attacker is able to interact remotely with the system under attack, and is therefore able to measure the response time.

In [17], the authors consider a *time-bound* opacity, where the attacker has to disclose the secret before an upper bound, using a partial observability. The authors prove that this problem is decidable for TAs. A construction and an algorithm are also provided to solve it; a case study is verified using SpaceEx [18]. In contrast, our definition of opacity only assumes observation of the execution time, does not assume any time-bounded setting, and our most general problem is parametric.

In [19], the authors propose a type system dealing with non-determinism and (continuous) real-time, the adequacy of which is ensured using non-interference. We share the common formalism of TAs; however, we mainly focus on leakage as execution time, and we *synthesize* internal parts of the system (clock guards), in contrast to [19] where the system is fixed.

In [20], Vasilikos *et al.* define the security of timed automata in term of information flow using a bisimulation relation over a set of observable nodes and develop an algorithm for deriving a sound constraint for satisfying the information flow property locally based on relevant transitions.

In [21], Gerking *et al.* study non-interference properties with input, high and low actions and provide a resolution method reducing a secure behavior to an unreachability construction. The proof-of-concept consists in the exhibition of a test automaton with a dedicated location that indicates violations of noninterference whenever it is reachable during execution. Then, Uppaal [22] is used to obtain the answer.

In [10], Benattar *et al.* study the control synthesis problem of timed automata for SNNI. That is, given a timed automaton, they propose a method to automatically generate a (largest) sub-system such that it is non-interferent, if possible. Different from the above-mentioned work, our work considers *parametric* timed automata, i. e., timed systems with unknown design parameters, and focuses on synthesizing parameter valuations which guarantee information flow property. Compared to [10], our approach is more realistic as it does not require change of program structure. Rather, our result provides guidelines on how to choose the timing parameters (e. g., how long to wait after certain program statements) for avoiding information leakage.

In [23, 24], Wang *et al.* investigate interesting opacity problems for real-time automata. These works come with a dedicated Python implementation. Although their definition shares similarities with ours, real-time automata are a severely restricted formalism compared to TAs. Indeed, timed aspects are only considered by interval restrictions over the total elapsed time along transitions. Real-time automata can be seen as a subclass of TAs with a single clock, reset at each transition. Also, parameters are not considered in their work.

To the best of our knowledge, our approach is the first work on parametric model checking for timed automata for information flow property. In addition, and in contrast to most of the aforementioned works, our approach comes with an implementation.

Execution times and timed automata. In this paper, we need to compute execution times in timed automata, i. e., the durations of all runs reaching the final state. Despite its natural aspect, this problem seems sparsely investigated in the literature. Both [25, 26] deal with the computation of duration sets; we do reuse some of the reasoning from [25] in our proof of Proposition 5.2. Conversely, results from [27] cannot be used in our work: while the equality must be forbidden in PTCTL formulae to make the problems decidable, equality constraints in a PTCTL formula would

be required for such an approach to answer our problems. Furthermore, the results presented in [28] cannot be applied to our study, since they concern one-clock timed automata.

Mitigating information leakage. Complex systems may exhibit security problems through information leakage due to the presence of unintended communication media, called side channels. An example is time side channels in which measuring, e. g., execution times, gives information on some sensitive information. In 2018, the Spectre vulnerability [29] exploited speculative execution to bring secret information into the cache; subsequently, cache-timing attacks were launched to exfiltrate these secrets. Therefore, mitigation of timing attacks is of utmost importance.

Our work is related to work on mitigating information leakage through those time side channels [30, 31, 32, 33, 34]. In [30], Agat *et al.* proposed to eliminate time side channels through type-driven cross-copying. In [31], Molnar *et al.* proposed, along the program counter model, a method for mitigating side channel through merging branches. A similar idea was proposed in [35]. Coppens *et al.* [32] developed a compiler backend for removing such leaks on x86 processors. In [33], Wang *et al.* proposed to automatically generate masking code for eliminating side channels through program synthesis. In [34], Wu *et al.* proposed to eliminate time side channels through program repair. Different from the above-mentioned works, we reduce the problem of mitigating time side channels as a parametric model checking problem and solve it using parametric reachability analysis techniques.

This work is related to work on identifying information leakage through timing analysis [36, 37, 38, 39, 40, 41, 42]. In [37], Chattopadhyay and Roychoudhury applied model checking to perform cache timing analysis. In [43], Chu *et al.* performed similar analysis through symbolic execution. In [38], Abbasi *et al.* apply the NuSMV model checker to verify integrated circuits against information leakage through side channels. In [41], a tool is developed to identify time side channel through static analysis. In [39], Sung *et al.* developed a framework based on LLVM for cache timing analysis.

3 PRELIMINARIES

In this work, we assume a system is modeled in the form of a parametric timed automaton (PTA). In Section 9.2, we discuss how we can model programs with unknown design parameters (e. g., a Java program with a statement `Thread.sleep(n)` where `n` is unknown) as PTA.

3.1 Clocks, parameters and guards

We assume a set $\mathbb{X} = \{x_1, \ldots, x_H\}$ of *clocks*, i. e., real-valued variables that all evolve over time at the same rate. A clock valuation is a function $\mu : \mathbb{X} \to \mathbb{R}_{\geq 0}$. We write $\vec{0}$ for the clock valuation assigning 0 to all clocks. Given $d \in \mathbb{R}_{\geq 0}$, $\mu + d$ denotes the valuation s.t. $(\mu + d)(x) = \mu(x) + d$, for all $x \in \mathbb{X}$. Given $R \subseteq \mathbb{X}$, we define the *reset* of a valuation μ, denoted by $[\mu]_R$, as follows: $[\mu]_R(x) = 0$ if $x \in R$, and $[\mu]_R(x) = \mu(x)$ otherwise.

We assume a set $\mathbb{P} = \{p_1, \ldots, p_M\}$ of *parameters*, i. e., unknown constants. A parameter *valuation* v is a function $v : \mathbb{P} \to \mathbb{Q}_+$. We assume $\bowtie \in \{<, \leq, =, \geq, >\}$. A guard g is a constraint over $\mathbb{X} \cup \mathbb{P}$ defined by a conjunction of inequalities of the form $x \bowtie \sum_{1 \leq i \leq M} \alpha_i p_i + d$, with $p_i \in \mathbb{P}$, and $\alpha_i, d \in \mathbb{Z}$. Given g, we write $\mu \models v(g)$ if the expression obtained by replacing each x with $\mu(x)$ and each p with $v(p)$ in g evaluates to true.

3.2 Parametric timed automata

Parametric timed automata (PTAs) extend timed automata with parameters within guards and invariants in place of integer constants [3].

3.2.1 *Syntax.*

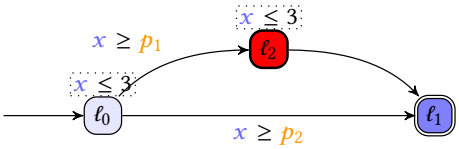


Fig. 1. A PTA example

Definition 3.1 (PTA). A PTA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, \ell_f, \mathbb{X}, \mathbb{P}, I, E)$, where:

(1) Σ is a finite set of actions,
(2) L is a finite set of locations,
(3) $\ell_0 \in L$ is the initial location,
(4) $\ell_f \in L$ is the (unique) final location,
(5) $\mathbb{X}$ is a finite set of clocks,
(6) $\mathbb{P}$ is a finite set of parameters,
(7) I is the invariant, assigning to every $\ell \in L$ a guard $I(\ell)$,
(8) E is a finite set of edges $e = (\ell, g, a, R, \ell')$ where $\ell, \ell' \in L$ are the source and target locations, $a \in \Sigma$, $R \subseteq \mathbb{X}$ is a set of clocks to be reset, and g is a guard.

Example 3.2. Consider the PTA in Fig. 1 (inspired by [11, Fig. 1b]), using one clock x and two parameters p_1 and p_2. ℓ_0 is the initial location, while we assume that ℓ_1 is the (only) *final* location, i. e., a location in which an attacker can measure the execution time from the initial location.

L/U-PTAs. For some theoretical problems solved in Sections 6 and 7, we will consider the subclass of PTAs called "lower-bound/upper-bound parametric timed automata" (L/U-PTAs), introduced in [44].

Definition 3.3 (L/U-PTA [44]). An *L/U-PTA* is a PTA where the set of parameters is partitioned into lower-bound parameters and upper-bound parameters, where each upper-bound (resp. lower-bound) parameter p_i must be such that, for every guard or invariant constraint $x \bowtie \sum_{1 \leq i \leq M} \alpha_i p_i + d$, we have: $\bowtie \in \{\leq, <\}$ implies $\alpha_i \geq 0$ (resp. $\alpha_i \leq 0$) and $\bowtie \in \{\geq, >\}$ implies $\alpha_i \leq 0$ (resp. $\alpha_i \geq 0$).

Example 3.4. The PTA in Fig. 1 is an L/U-PTA with $\{p_1, p_2\}$ as lower-bound parameters, and $\emptyset$ as upper-bound parameters.

The PTA in Fig. 5 is not an L/U-PTA, because p is compared to cl both as a lower-bound (in "$p \times 32^2 \leq cl$") and as an upper-bound ("$cl \leq p \times 32^2 + \epsilon$").

Given a parameter valuation v, we denote by $v(\mathcal{A})$ the non-parametric structure where all occurrences of any parameter p_i have been replaced by $v(p_i)$. We denote as a *timed automaton* any structure $v(\mathcal{A})$, by assuming a rescaling of the constants: by multiplying all constants in $v(\mathcal{A})$ by the least common multiple of their denominators, we obtain an equivalent (integer-valued) TA, as defined in [2].

Synchronized product of PTAs. The *synchronous product* (using strong broadcast, i. e., synchronization on a given set of actions), or *parallel composition*, of several PTAs gives a PTA.

Definition 3.5 (synchronized product of PTAs). Let $N \in \mathbb{N}$. Given a set of PTAs $\mathcal{A}_i = (\Sigma_i, L_i, (\ell_0)_i, (\ell_f)_i, \mathbb{X}_i, \mathbb{P}_i, I_i, E_i)$, $1 \leq i \leq N$, and a set of actions Σ_s, the *synchronized product* of $\mathcal{A}_i$, $1 \leq i \leq N$, denoted by $\mathcal{A}_1 \parallel_{\Sigma_s} \mathcal{A}_2 \parallel_{\Sigma_s} \cdots \parallel_{\Sigma_s} \mathcal{A}_N$, is the tuple $(\Sigma, L, \ell_0, \ell_f, \mathbb{X}, \mathbb{P}, I, E)$, where:

(1) $\Sigma = \bigcup_{i=1}^{N} \Sigma_i$,
(2) $L = \prod_{i=1}^{N} L_i$,

(3) $\ell_0 = ((\ell_0)_1, \ldots, (\ell_0)_N)$,
(4) $\ell_f = ((\ell_f)_1, \ldots, (\ell_f)_N)$,
(5) $\mathbb{X} = \bigcup_{1 \leq i \leq N} \mathbb{X}_i$,
(6) $\mathbb{P} = \bigcup_{1 \leq i \leq N} \mathbb{P}_i$,
(7) $I((\ell_1, \ldots, \ell_N)) = \bigwedge_{i=1}^{N} I_i(\ell_i)$ for all $(\ell_1, \ldots, \ell_N) \in L$,

and E is defined as follows. For all $a \in \Sigma$, let ζ_a be the subset of indices $i \in 1, \ldots, N$ such that $a \in \Sigma_i$. For all $a \in \Sigma$, for all $(\ell_1, \ldots, \ell_N) \in L$, for all $(\ell'_1, \ldots, \ell'_N) \in L$, $\big((\ell_1, \ldots, \ell_N), g, a, R, (\ell'_1, \ldots, \ell'_N)\big) \in E$ if:

- if $a \in \Sigma_s$, then
  (1) for all $i \in \zeta_a$, there exist g_i, R_i such that $(\ell_i, g_i, a, R_i, \ell'_i) \in E_i$, $g = \bigwedge_{i \in \zeta_a} g_i$, $R = \bigcup_{i \in \zeta_a} R_i$, and,
  (2) for all $i \notin \zeta_a$, $\ell'_i = \ell_i$.
- otherwise (if $a \notin \Sigma_s$), then there exists $i \in \zeta_a$ such that
  (1) there exist g_i, R_i such that $(\ell_i, g_i, a, R_i, \ell'_i) \in E_i$, $g = g_i$, $R = R_i$, and,
  (2) for all $j \neq i$, $\ell'_j = \ell_j$.

That is, synchronization is only performed on Σ_s, and other actions are interleaved.

3.2.2 Concrete semantics of TAs. Let us now recall the concrete semantics of TA.

Definition 3.6 (Semantics of a TA). Given a PTA $\mathcal{A} = (\Sigma, L, \ell_0, \ell_f, \mathbb{X}, \mathbb{P}, I, E)$, and a parameter valuation v, the semantics of $v(\mathcal{A})$ is given by the timed transition system (TTS) [45] $T_{v(\mathcal{A})} = (S, s_0, \rightarrow)$, with

- $S = \{(\ell, \mu) \in L \times \mathbb{R}^H_{\geq 0} \mid \mu \models v(I(\ell))\}$,
- $s_0 = (\ell_0, \vec{0})$,
- $\rightarrow$ consists of the discrete and (continuous) delay transition relations:
  (1) discrete transitions: $(\ell, \mu) \overset{e}{\mapsto} (\ell', \mu')$, if $(\ell, \mu), (\ell', \mu') \in S$, and there exists $e = (\ell, g, a, R, \ell') \in E$, such that $\mu' = [\mu]_R$, and $\mu \models v(g)$.
  (2) delay transitions: $(\ell, \mu) \overset{d}{\mapsto} (\ell, \mu + d)$, with $d \in \mathbb{R}_{\geq 0}$, if $\forall d' \in [0, d], (\ell, \mu + d') \in S$.

Moreover we write $(\ell, \mu) \xrightarrow{(d,e)} (\ell', \mu')$ for a combination of a delay and discrete transition if $\exists \mu'' : (\ell, \mu) \overset{d}{\mapsto} (\ell, \mu'') \overset{e}{\mapsto} (\ell', \mu')$.

Given a TA $v(\mathcal{A})$ with concrete semantics $(S, s_0, \rightarrow)$, we refer to the states of $T_{v(\mathcal{A})}$ as the *concrete states* of $v(\mathcal{A})$. A *run* of $v(\mathcal{A})$ is a (finite or infinite) alternating sequence of concrete states of $v(\mathcal{A})$ and pairs of delays and edges starting from the initial state s_0 of the form $s_0, (d_0, e_0), s_1, \cdots$ with $i = 0, 1, \ldots, e_i \in E$, $d_i \in \mathbb{R}_{\geq 0}$ and $s_i \xrightarrow{(d_i, e_i)} s_{i+1}$.

Given a state $s = (\ell, \mu)$, we say that s is reachable in $v(\mathcal{A})$ if s appears in a run of $v(\mathcal{A})$. By extension, we say that ℓ is reachable; and by extension again, given a set L_T of locations, we say that L_T is reachable if there exists $\ell \in L_T$ such that ℓ is reachable in $v(\mathcal{A})$.[1] Given $\ell, \ell' \in L$ and a run ρ, we say that ℓ is reached on the way to ℓ' in ρ if ρ is of the form $(\ell_0, \mu_0), (d_0, e_0), (\ell_1, \mu_1), \cdots, (\ell_m, \mu_m), (d_m, e_m), \cdots (\ell_n, \mu_n)$ for some $m, n \in \mathbb{N}$ such that $\ell_m = \ell$, $\ell_n = \ell'$ and $\forall 0 \leq i \leq m-1, \ell_i \neq \ell'$. Conversely, ℓ is avoided on the way to ℓ' in ρ if ρ is of the form $(\ell_0, \mu_0), (d_0, e_0), (\ell_1, \mu_1), \cdots, (\ell_n, \mu_n)$ with $\ell_n = \ell'$ and $\forall 0 \leq i \leq n, \ell_i \neq \ell$. Given $\ell, \ell' \in L$, we say that ℓ is *reachable on the way to* ℓ' in $v(\mathcal{A})$ if there exists a run ρ of $v(\mathcal{A})$ for which ℓ is reached on the way to ℓ' in ρ. Otherwise, ℓ is *unreachable on the way to* ℓ'.

[1] We use an existential quantification over the set L_T of locations, i. e., the set of locations is reachable if at least one target location is reachable. This is a standard definition for reachability synthesis (see, e. g., [46]), and our Algorithm 1 uses a singleton set for L_T anyway.

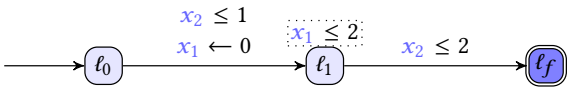


Fig. 2. A TA example

The *duration* of a finite run $\rho : s_0, (d_0, e_0), s_1, \cdots, (d_{i-1}, e_{i-1}), (\ell_i, \mu_i)$ is $dur(\rho) = \sum_{0 \leq j \leq i-1} d_j$. We also say that ℓ_i is reachable in time $dur(\rho)$.

Example 3.7. Consider again the PTA $\mathcal{A}$ in Fig. 1, and let v be such that $v(p_1) = 1$ and $v(p_2) = 2$. Consider the following run ρ of $v(\mathcal{A})$: $(\ell_0, x = 0), (1.4, e_2), (\ell_2, x = 1.4), (1.3, e_3), (\ell_1, x = 2.7)$, where e_2 is the edge from ℓ_0 to ℓ_2 in Fig. 1, and e_3 is the edge from ℓ_2 to ℓ_1. We write "$x = 1.4$" instead of "μ such that $\mu(x) = 1.4$". We have $dur(\rho) = 1.4 + 1.3 = 2.7$. In addition, ℓ_2 is reached on the way to ℓ_1 in ρ.

3.2.3 Timed automata regions. Let us next recall the concept of regions and the region graph [2].

Given a TA $v(\mathcal{A})$, for a clock x_i, we denote by c_i the largest constant to which x_i is compared within the guards and invariants of $v(\mathcal{A})$ (that is, $c_i = \max_i(\{ d_i \mid x \bowtie d_i$ appears in a guard or invariant of $v(\mathcal{A})\})$. Given a clock valuation μ and a clock x_i, let $\lfloor \mu(x_i) \rfloor$ and $\text{fract}(\mu(x_i))$ denote respectively the integral part and the fractional part of $\mu(x_i)$.

Example 3.8. Consider again the PTA in Fig. 1, and let v be such that $v(p_1) = 2$ and $v(p_2) = 4$. In the TA $v(\mathcal{A})$, the clock x is compared to the constants in $\{2, 3, 4\}$. In that case, $c = 4$ is the largest constant to which the clock x is compared.

Definition 3.9 (Region equivalence). We say that two clock valuations μ and μ' are equivalent, denoted $\mu \approx \mu'$, if the following three conditions hold for any clocks x_i, x_j:

(1) either
  (a) $\lfloor \mu(x_i) \rfloor = \lfloor \mu'(x_i) \rfloor$ or
  (b) $\mu(x_i) > c_i$ and $\mu'(x_i) > c_i$
(2) $\text{fract}(\mu(x_i)) \leq \text{fract}(\mu(x_j))$ iff $\text{fract}(\mu'(x_i)) \leq \text{fract}(\mu'(x_j))$
(3) $\text{fract}(\mu(x_i)) = 0$ iff $\text{fract}(\mu'(x_i)) = 0$

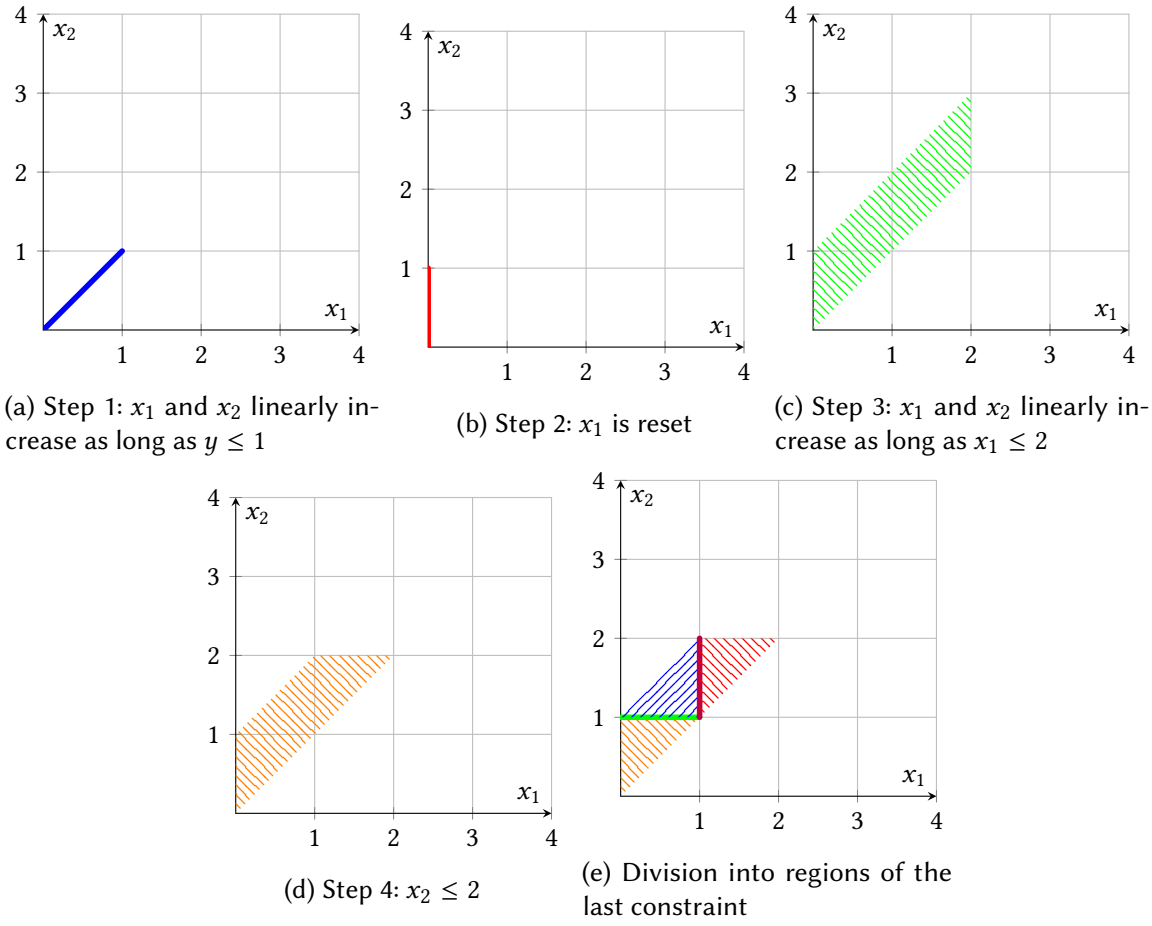
The equivalence relation $\approx$ is extended to the states of $T_{v(\mathcal{A})}$: if $s = (\ell, \mu), s' = (\ell', \mu')$ are two states of $T_{v(\mathcal{A})}$, we write $s \approx s'$ iff $\ell = \ell'$ and $\mu \approx \mu'$.

We denote by $[s]$ the equivalence class of s for $\approx$. A *region* is an equivalence class $[s]$ of $\approx$. The set of all regions is denoted $\mathcal{R}_{v(\mathcal{A})}$. Given a state $s = (\ell, \mu)$ and $d \geq 0$, we write $s + d$ to denote $(\ell, \mu + d)$.

Definition 3.10 (Region graph [25]). The *region graph* $\mathcal{RG}_{v(\mathcal{A})} = (\mathcal{R}_{v(\mathcal{A})}, \mathcal{F}_{v(\mathcal{A})})$ is a finite graph with:

- $\mathcal{R}_{v(\mathcal{A})}$ as the set of vertices
- given two regions $r = [s], r' = [s'] \in \mathcal{R}_{v(\mathcal{A})}$, we have $(r, r') \in \mathcal{F}_{v(\mathcal{A})}$ if one of the following holds:
  - if $s \overset{e}{\mapsto} s' \in T_{v(\mathcal{A})}$ for some $e \in E$ (*discrete instantaneous transition*);
  - if r' is a time successor of r: $r \neq r'$ and there exists d such that $s + d \in r'$ and $\forall d' < d, s + d' \in r \cup r'$ (*delay transition*);
  - $r = r'$ is unbounded: $s = (\ell, \mu)$ with $\mu(x_i) > c_i$ for all x_i (*equivalent unbounded regions*).

Example 3.11 (Region abstraction). Consider the TA in Fig. 2. We present in Fig. 3 some of its regions. For example, the region depicted in Fig. 3b is the successor (by taking the discrete transition from ℓ_0 to ℓ_1, resetting x_1) of the region depicted in Fig. 3a.



(a) Step 1: x_1 and x_2 linearly increase as long as $y \leq 1$

(b) Step 2: x_1 is reset

(c) Step 3: x_1 and x_2 linearly increase as long as $x_1 \leq 2$

(d) Step 4: $x_2 \leq 2$

(e) Division into regions of the last constraint

Fig. 3. Region abstraction of the TA in Fig. 2

Consider the valuations μ_1 and μ_2 such that $\mu_1(x_1) = 0$, $\mu_1(x_2) = 0.35$, $\mu_2(x_1) = 0$, and $\mu_2(x_2) = 0.75$. These two valuations satisfy all three conditions of Definition 3.9, and therefore $\mu_1 \approx \mu_2$. Note that they both belong to the region depicted in Fig. 3b, and any pair of valuations in this region is (by definition) equivalent.

3.3 Symbolic semantics

Let us now recall the symbolic semantics of PTAs (see e. g., [44, 47]).

Constraints. We first need to define operations on constraints. A linear term over $\mathbb{X} \cup \mathbb{P}$ is of the form $\sum_{1 \leq i \leq H} \alpha_i x_i + \sum_{1 \leq j \leq M} \beta_j p_j + d$, with $x_i \in \mathbb{X}$, $p_j \in \mathbb{P}$, and $\alpha_i, \beta_j, d \in \mathbb{Z}$. A *constraint* C (i. e., a convex polyhedron[2]) over $\mathbb{X} \cup \mathbb{P}$ is a conjunction of inequalities of the form $lt \bowtie 0$, where lt is a linear term.

Given a parameter valuation v, $v(C)$ denotes the constraint over $\mathbb{X}$ obtained by replacing each parameter p in C with $v(p)$. Likewise, given a clock valuation μ, $\mu(v(C))$ denotes the expression obtained by replacing each clock x in $v(C)$ with $\mu(x)$. We write $\mu \models v(C)$ whenever $\mu(v(C))$ evaluates to true. We say that v *satisfies* C, denoted by $v \models C$, if the set of clock valuations satisfying $v(C)$ is nonempty. We say that C is *satisfiable* if $\exists \mu, v$ s.t. $\mu \models v(C)$.

[2]Strictly speaking, we manipulate *polytopes*, while polyhedra refer to 3-dimensional polytopes. However, for sake of consistency with the parametric timed model checking literature, and with the Parma polyhedra library (among others), we refer to these geometric objects as *polyhedra*.

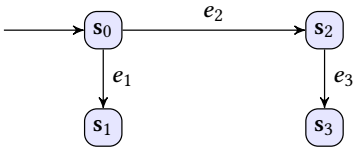


Fig. 4. Parametric zone graph of Fig. 1

We define the *time elapsing* of C, denoted by $C^{\nearrow}$, as the constraint over $\mathbb{X}$ and $\mathbb{P}$ obtained from C by delaying all clocks by an arbitrary amount of time. That is,

$$\mu' \models v(C^{\nearrow}) \text{ if } \exists \mu : \mathbb{X} \to \mathbb{R}_+, \exists d \in \mathbb{R}_+ \text{ s.t. } \mu \models v(C) \wedge \mu' = \mu + d.$$

Given $R \subseteq \mathbb{X}$, we define the *reset* of C, denoted by $[C]_R$, as the constraint obtained from C by resetting the clocks in R to 0, and keeping the other clocks unchanged. That is,

$$\mu' \models v([C]_R) \text{ if } \exists \mu : \mathbb{X} \to \mathbb{R}_+ \text{ s.t. } \mu \models v(C) \wedge \forall x \in \mathbb{X} \begin{cases} \mu'(x) = 0 & \text{if } x \in R \\ \mu'(x) = \mu(x) & \text{otherwise.} \end{cases}$$

We denote by $C{\downarrow_{\mathbb{P}}}$ the projection of C onto $\mathbb{P}$, i. e., obtained by eliminating the variables not in $\mathbb{P}$ (e. g., using Fourier-Motzkin [48]).

Definition 3.12 (Symbolic state). A symbolic state is a pair (ℓ, C) where $\ell \in L$ is a location, and C its associated parametric zone.

Definition 3.13 (Symbolic semantics). Given a PTA $\mathcal{A} = (\Sigma, L, \ell_0, \ell_f, \mathbb{X}, \mathbb{P}, I, E)$, the symbolic semantics of $\mathcal{A}$ is the labeled transition system called *parametric zone graph* $\mathcal{PZG} = (E, \mathbf{S}, \mathbf{s}_0, \Rightarrow)$, with

- $\mathbf{S} = \{(\ell, C) \mid C \subseteq I(\ell)\}$,
- $\mathbf{s}_0 = \big(\ell_0, (\bigwedge_{1 \leq i \leq H} x_i = 0)^{\nearrow} \wedge I(\ell_0)\big)$, and
- $\big((\ell, C), e, (\ell', C')\big) \in \Rightarrow$ if $e = (\ell, g, a, R, \ell') \in E$ and

$$C' = \big([(C \wedge g)]_R \wedge I(\ell')\big)^{\nearrow} \wedge I(\ell')$$

with C' satisfiable.

That is, in the parametric zone graph, nodes are symbolic states, and arcs are labeled by *edges* of the original PTA.

If $(\mathbf{s}, e, \mathbf{s}') \in \Rightarrow$, we write $\mathrm{Succ}(\mathbf{s}, e) = \mathbf{s}'$. By extension, we write $\mathrm{Succ}(\mathbf{s})$ for $\cup_{e \in E} \mathrm{Succ}(\mathbf{s}, e)$.

In the non-parametric timed automata setting, a zone can be seen as a *convex union* of regions. In the parametric setting, a parametric zone constrains both clocks and parameters in such a way that, for each admissible parameter valuation, the resulting projection on clocks is a zone.

Example 3.14. Consider again the PTA $\mathcal{A}$ in Fig. 1. The parametric zone graph of $\mathcal{A}$ is given in Fig. 4, where e_1 is the edge from ℓ_0 to ℓ_1 in Fig. 1, e_2 is the edge from ℓ_0 to ℓ_2, and e_3 is the edge from ℓ_2 to ℓ_1. In addition, the symbolic states are:

$\mathbf{s}_0 = (\ \ell_0\ ,\ 0 \leq x \leq 3 \wedge p_1 \geq 0 \wedge p_2 \geq 0\ \ )$
$\mathbf{s}_1 = (\ \ell_1\ ,\ x \geq p_2 \wedge 0 \leq p_2 \leq 3 \wedge p_1 \geq 0\ \ )$
$\mathbf{s}_2 = (\ \ell_2\ ,\ 3 \geq x \geq p_1 \wedge 0 \leq p_1 \leq 3 \wedge p_2 \geq 0\ )$
$\mathbf{s}_3 = (\ \ell_1\ ,\ x \geq p_1 \wedge 0 \leq p_1 \leq 3 \wedge p_2 \geq 0\ \ )$.

3.4 Reachability synthesis

We use reachability synthesis to solve the problems defined in Section 4. This procedure, called EFsynth, takes as input a PTA $\mathcal{A}$ and a set of target locations L_T, and attempts to synthesize all parameter valuations v for which L_T is reachable in $v(\mathcal{A})$. $\mathsf{EFsynth}(\mathcal{A}, L_T)$ was formalized in e. g., [46] and is a procedure that may not terminate, but that computes an exact result (sound and complete) if it terminates. EFsynth traverses the *parametric zone graph* of $\mathcal{A}$.

Example 3.15. Consider again the PTA $\mathcal{A}$ in Fig. 1. $\mathsf{EFsynth}(\mathcal{A}, \{\ell_1\}) = p_1 \leq 3 \vee p_2 \leq 3$. Intuitively, it corresponds to all parameter constraints in the parametric zone graph in Fig. 4 associated to symbolic states with location ℓ_1.

We finally recall the correctness of EFsynth.

Lemma 3.16 ([46]). *Let $\mathcal{A}$ be a PTA, and let L_T be a subset of the locations of $\mathcal{A}$. Assume EFsynth$(\mathcal{A}, L_T)$ terminates with result K. Then $v \models K$ iff L_T is reachable in $v(\mathcal{A})$.*

4 TIMED OPACITY PROBLEMS

4.1 Definitions

Let us first introduce two key concepts to define our notion of opacity. $DReach_{\ell}^{v(\mathcal{A})}(\ell')$ (resp. $DReach_{\neg\ell}^{v(\mathcal{A})}(\ell')$) is the set of all the durations of the runs for which ℓ is reachable (resp. unreachable) on the way to ℓ'. Formally:

$$DReach_{\ell}^{v(\mathcal{A})}(\ell') = \{d \mid \exists \rho \text{ in } v(\mathcal{A}) \text{ such that } d = dur(\rho) \wedge \ell \text{ is reached on the way to } \ell' \text{ in } \rho\}$$

and

$$DReach_{\neg\ell}^{v(\mathcal{A})}(\ell') = \{d \mid \exists \rho \text{ in } v(\mathcal{A}) \text{ such that } d = dur(\rho) \wedge \ell \text{ is avoided on the way to } \ell' \text{ in } \rho\}.$$

These concepts can be seen as the set of execution times from the initial location ℓ_0 to a target location ℓ' while passing (resp. not passing) by a private location ℓ. Observe that, from the definition of $dur(\rho)$, this "execution time" does not include the time spent in ℓ'.

Example 4.1. Consider again the PTA in Fig. 1, and let v be such that $v(p_1) = 1$ and $v(p_2) = 2$. We have $DReach_{\ell_2}^{v(\mathcal{A})}(\ell_1) = [1, 3]$ and $DReach_{\neg\ell_2}^{v(\mathcal{A})}(\ell_1) = [2, 3]$.

We now introduce the concept of "timed opacity w.r.t. a set of durations (or execution times) D": a system is opaque w.r.t. a given location ℓ_{priv} on the way to ℓ_f for execution times D whenever, for any duration in D, it is not possible to deduce whether the system went through ℓ_{priv} or not. In other words, if an attacker measures an execution time within D from the initial location to the target location ℓ_f, then this attacker is not able to deduce whether the system visited ℓ_{priv}.

Definition 4.2 (timed opacity w.r.t. D). Given a TA $v(\mathcal{A})$, a private location ℓ_{priv}, a target location ℓ_f and a set of execution times D, we say that $v(\mathcal{A})$ is *opaque w.r.t. ℓ_{priv} on the way to ℓ_f for execution times D* if $D \subseteq DReach_{\ell_{priv}}^{v(\mathcal{A})}(\ell_f) \cap DReach_{\neg\ell_{priv}}^{v(\mathcal{A})}(\ell_f)$.

If one does not have the ability to tune the system (i. e., change internal delays, or add some `sleep()` or `Wait()` statements in the program), one may be first interested in knowing whether the system is opaque for all execution times (i. e., the durations of the runs from the initial location to the target location ℓ_f). In other words, if a system is *fully opaque*, for any possible measured execution time, an attacker is not able to deduce anything on the system, in terms of visit of ℓ_{priv}.

Definition 4.3 (full timed opacity). Given a TA $v(\mathcal{A})$, a private location ℓ_{priv} and a target location ℓ_f, we say that $v(\mathcal{A})$ is *fully opaque w.r.t. ℓ_{priv} on the way to ℓ_f* if $DReach_{\ell_{priv}}^{v(\mathcal{A})}(\ell_f) = DReach_{\neg\ell_{priv}}^{v(\mathcal{A})}(\ell_f)$.

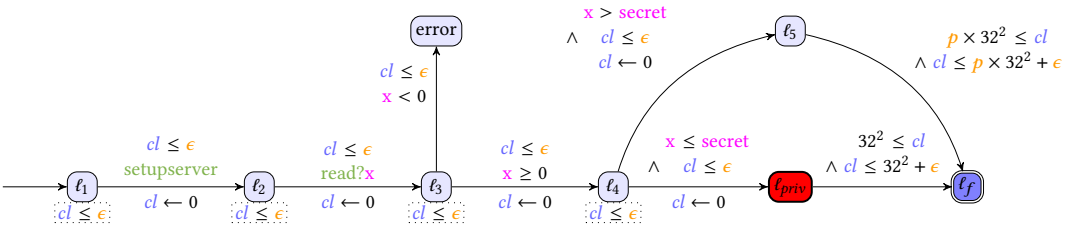


Fig. 5. A Java program encoded in a PTA

That is, a system is fully opaque if, for any execution time d, a run of duration d reaches ℓ_f after going through ℓ_{priv} iff another run of duration d reaches ℓ_f without going through ℓ_{priv}.

Remark 1. This definition is symmetric: a system is not opaque iff an attacker can deduce ℓ_{priv} or $\neg\ell_{priv}$. For instance, if there is no path through ℓ_{priv} to ℓ_f, but a path to ℓ_f, a system is not opaque w.r.t. Definition 4.3.

Example 4.4. Consider the PTA $\mathcal{A}$ in Fig. 5 where cl is a clock, while ϵ, p are parameters. We use a sightly extended PTA syntax: $read?x$ reads the value input on a given channel $read$, and assigns it to a (discrete, global) variable x. secret is a constant variable of arbitrary value. If both x and secret are finite-domain variables (e. g., bounded integers) then they can be seen as syntactic sugar for locations. Such variables are supported by most model checkers, including Uppaal [22] and IMITATOR [49].

This PTA encodes a server process and is a (manual) translation of a Java program from the DARPA Space/Time Analysis for Cybersecurity (STAC) library[3], that compares a user-input variable with a given secret and performs different actions taking different times depending on this secret. The original Java program is *vulnerable*, and tagged as such in the DARPA library, because some sensitive information can be deduced from the timing information. The original Java code is given in Appendix A.

In our encoding, a single instruction takes a time in $[0, \epsilon]$, while p is a (parametric) factor to one of the `sleep` instructions of the program. Note that in the original Java code in Appendix A, at line 25, there is no parameter p but an integer 2; that is, the code is fixed to have $v(p) = 2$. For sake of simplicity, we abstract away instructions not related to time, and merge subfunctions calls. For this work, we simplify the problem and abstract in this way. Precisely modeling the timing behavior of a program is itself a complicated problem (due to caching, speculative execution, etc.) and we leave to future work.

Let v_1 such that $v_1(\epsilon) = 1$ and $v_1(p) = 2$. For this example, $DReach_{\ell_{priv}}^{v_1(\mathcal{A})}(\ell_f) = [1024, 1029]$ while $DReach_{\neg\ell_{priv}}^{v_1(\mathcal{A})}(\ell_f) = [2048, 2053]$. Therefore, $v_1(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ_f for execution times $D = [1024, 1029] \cap [2048, 2053] = \emptyset$.

Let v_2 such that $v_2(\epsilon) = 2$ and $v_2(p) = 1.002$. $DReach_{\ell_{priv}}^{v_2(\mathcal{A})}(\ell_f) = [1024, 1034]$ while $DReach_{\neg\ell_{priv}}^{v_2(\mathcal{A})}(\ell_f) = [1026.048, 1036.048]$. Therefore, $v_2(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ_f for execution times $D = [1026.048, 1034]$.

Obviously, neither $v_1(\mathcal{A})$ nor $v_2(\mathcal{A})$ are fully opaque w.r.t. ℓ_{priv} on the way to ℓ_f.

4.2 Decision and computation problems

We can now define the timed opacity computation problem, which consists in computing the possible execution times ensuring opacity w.r.t. a private location. In other words, the attacker

[3]https://github.com/Apogee-Research/STAC/blob/master/Canonical_Examples/Source/Category1_vulnerable.java

model is as follows: the attacker knows the system model in the form of a TA, and can only observe the computation time between the start of the program and the time it reaches a given (final) location.

Timed opacity computation Problem:
INPUT: A TA $v(\mathcal{A})$, a private location ℓ_{priv}, a target location ℓ_f
PROBLEM: Compute the execution times D such that $v(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ_f for these execution times D.

Let us illustrate that this computation problem is certainly not easy. For the TA $\mathcal{A}$ in Fig. 6, the execution times D for which $\mathcal{A}$ is opaque w.r.t. ℓ_{priv} on the way to ℓ_f is exactly $\mathbb{N}$; that is, only integer times are opaque (as the system can only leave ℓ_{priv} and hence enter ℓ_f at an integer time).

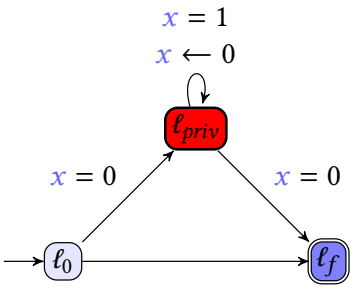


Fig. 6. TA for which the set of opaque execution times is $\mathbb{N}$

The synthesis counterpart allows for a higher-level problem by also synthesizing the internal timings guaranteeing opacity.

Timed opacity synthesis Problem:
INPUT: A PTA $\mathcal{A}$, a private location ℓ_{priv}, a target location ℓ_f
PROBLEM: Synthesize the parameter valuations v and the execution times D_v such that $v(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ_f for these execution times D_v.

Note that the execution times can depend on the parameter valuations.

We can also define the full timed opacity decision problem, which consists in answering whether a timed automaton is fully opaque w.r.t. a private location.

Full timed opacity decision Problem:
INPUT: A TA $v(\mathcal{A})$, a private location ℓ_{priv}, a target location ℓ_f
PROBLEM: Is $v(\mathcal{A})$ fully opaque w.r.t. ℓ_{priv} on the way to ℓ_f?

Note that a last problem of interest would be *full timed opacity synthesis*, aiming at synthesizing (ideally the entire set of) parameter valuations v for which $v(\mathcal{A})$ is *fully opaque* w.r.t. ℓ_{priv} on the way to ℓ_f. This is left as future work.

5 TIMED OPACITY PROBLEMS FOR TIMED AUTOMATA

In this section, we address the non-parametric problems defined in Section 4.2, i. e., the timed opacity computation problem (Section 5.2) and full timed opacity decision problem (Section 5.3). We show that both problems can be solved using a construction of the *DReach* sets based on the RA arithmetic [5] (Section 5.1).

In this section, let $\mathcal{A}$ denote a (non-parametric) timed automaton.

5.1 Computing $DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$ and $DReach^{\mathcal{A}}_{\neg\ell_{priv}}(\ell_f)$

We must be able to express the set of execution times, i. e., the durations of all runs from the initial location to a given target location. While the problem of expressing the set of execution times seems very natural for timed automata, it was barely addressed in the literature, with the exception of [25, 26].

5.1.1 The RA arithmetic. We use the RA arithmetic, which is the set of first-order formulae, interpreted over the real numbers, of $\langle \mathbb{R}, +, <, \mathbb{N}, 0, 1\rangle$ where $\mathbb{N}$ is a unary predicate such that $\mathbb{N}(z)$ is true iff z is a natural number. This arithmetic has a decidable theory with a complexity of 3EXPTIME [5].

5.1.2 Computing execution times of timed automata. With $r, r' \in \mathcal{R}_{\mathcal{A}}$, we denote by $\lambda_{r,r'}$ the set of durations d such that there exists a finite path $\rho = (s_i)_i$ in $T_{\mathcal{A}}$ such that $dur(\rho) = d$ and the associated path $\pi(\rho) = (r_k)_{0\leq k\leq K}$ in the region graph satisfies $r_0 = r, r_K = r'$. It is shown in [25, Proposition 5.3] that these sets $\lambda_{r,r'}$ can be defined in RA arithmetic. Moreover, they are definable by a disjunction of terms of the form $d = m$, $\exists z, \mathbb{N}(z) \wedge d = m + cz$ and $\exists z, \mathbb{N}(z) \wedge m + cz < d < m + cz + 1$, where $c, m \in \mathbb{N}$.

Let us give the main idea of the proof presented in [25] (even though this explanation is not necessary to follow our reasoning). The idea of the proof of [25] is to consider a TA $\mathcal{A}^0$ obtained from $\mathcal{A}$ by adding a new clock x_0 which is reset to 0 each time it reaches the value 1 and to count all of the resets of x_0. The construction of $\mathcal{A}^0$ ensures that each (finite) run ρ of $T_{\mathcal{A}}$ corresponds to a run ρ^0 of $T_{\mathcal{A}^0}$ (at each state, the value of x_0 is the fractional part of the total time elapsed), and conversely (erasing x_0). The authors propose next a classical automaton C as a particular subgraph of the region graph $\mathcal{R}_{\mathcal{A}^0}$, where the only action a denotes the reset of x_0 (all other transitions are labeled with the silent action). The conclusion follows because $t \in \lambda_{r,r'}$ if $\lfloor t \rfloor$ is the length of a path in C', the deterministic automaton obtained from C by subset construction.

Lemma 5.1 (Reachability-duration computation). *The sets $DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$ and $DReach^{\mathcal{A}}_{\neg\ell_{priv}}(\ell_f)$ are computable and definable in RA arithmetic.*

Proof. Let $\mathcal{A}$ be a TA. We aim at reducing the computation of the sets $DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$ and $DReach^{\mathcal{A}}_{\neg\ell_{priv}}(\ell_f)$ to the computation of sets $\lambda_{r,r'}$.

First, let us compute $DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$. From $\mathcal{A}$, we define a TA $\mathcal{A}'$ by adding a Boolean discrete variable b, initially false. Recall that discrete variables over a finite domain are syntactic sugar for *locations*: therefore, ℓ_f with b = false and ℓ_f with b = true can be seen as two different locations. Then, we set $b \leftarrow$ true on any transition whose target location is ℓ_{priv}; therefore, b = true denotes that ℓ_{priv} has been visited. We denote by $\ell'_{f_{\text{true}}}$ the final state of $\mathcal{A}'$ where b = true. $DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$ is exactly the set of executions times in $\mathcal{A}'$ between ℓ_0 and $\ell'_{f_{\text{true}}}$. For all the regions r'_i associated to $\ell'_{f_{\text{true}}}$, we can compute (using [25]) λ_{r,r'_i}, where r is the region associated to ℓ_0 in $\mathcal{A}'$. Therefore, $DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$ can be computed as the union of all the λ_{r,r'_i} (of which there is a finite number), which is definable in RA arithmetic.

Second, let us compute $DReach^{\mathcal{A}}_{\neg\ell_{priv}}(\ell_f)$. We define another TA $\mathcal{A}''$ obtained from $\mathcal{A}$ by deleting all the transitions leading to ℓ_{priv}. Therefore, the set of durations reaching ℓ_f in $\mathcal{A}''$ is exactly the set of durations reaching ℓ_f in $\mathcal{A}$ associated to runs not visiting ℓ_{priv}. $DReach^{\mathcal{A}}_{\neg\ell_{priv}}(\ell_f)$ is exactly the set of executions times in $\mathcal{A}''$ between ℓ_0 and ℓ_f. For all the regions r'_i associated to ℓ_f, we can compute λ_{r,r'_i}, where r is the region associated to ℓ_0 in $\mathcal{A}''$. Therefore, $DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$ can be computed as the union of all the λ_{r,r'_i}. □

5.2 Answering the timed opacity computation problem

Proposition 5.2 (timed opacity computation). *The timed opacity computation problem is solvable for TAs.*

Proof. Let $\mathcal{A}$ be a TA. From Lemma 5.1, we can compute and define in RA arithmetic the sets $DReach^{\mathcal{A}}_{\neg\ell_{priv}}(\ell_f)$ and $DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$.

By the decidability of RA arithmetic, the intersection of these sets is computable. Then, the set $D = DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f) \cap DReach^{\mathcal{A}}_{\neg\ell_{priv}}(\ell_f)$ is effectively computable (with a 3EXPTIME upper bound). □

This positive result can be put in perspective with the negative result of [15] that proves that it is undecidable whether a TA (and even the more restricted subclass of event-recording automata [16]) is opaque, in a sense that the attacker can deduce some actions, by looking at observable actions together with their timing. The difference in our setting is that only the global time is observable, which can be seen as a single action, occurring once only at the end of the computation. In other words, our attacker is less powerful than the attacker in [15].

5.3 Checking for full timed opacity

Proposition 5.3 (full timed opacity decision). *The full timed opacity decision problem is decidable for TAs.*

Proof. Let $\mathcal{A}$ be a TA. From Lemma 5.1, we can compute and define in RA arithmetic the sets $DReach^{\mathcal{A}}_{\neg\ell_{priv}}(\ell_f)$ and $DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$.

From the decidability of RA arithmetic, the equality between these sets is decidable. Therefore, $DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f) \stackrel{?}{=} DReach^{\mathcal{A}}_{\neg\ell_{priv}}(\ell_f)$ is decidable. □

From [5] and [25, Theorem 7.5], the computation of a set $\lambda_{r,r'}$ is 2EXPTIME and the RA arithmetic has a decidable theory with complexity 3EXPTIME. Therefore, our construction is 3EXPTIME, which is an upper-bound for the problem complexity. Note that, as in [25], we did not compute a lower bound for the complexity of Propositions 5.2 and 5.3. This remains to be done as future work.

Example 5.4. Consider again the PTA $\mathcal{A}$ in Fig. 1, and let v be such that $v(p_1) = 1$ and $v(p_2) = 2$. Recall from Example 4.1 that $DReach^{v(\mathcal{A})}_{\ell_2}(\ell_1) = [1,3]$ and $DReach^{v(\mathcal{A})}_{\neg\ell_2}(\ell_1) = [2,3]$. Thus, $DReach^{v(\mathcal{A})}_{\ell_2}(\ell_1) \neq DReach^{v(\mathcal{A})}_{\neg\ell_2}(\ell_1)$ and therefore $v(\mathcal{A})$ is *not (fully)* opaque w.r.t. ℓ_2 on the way to ℓ_1.

Now, consider v' such that $v'(p_1) = v'(p_2) = 1.5$. This time, $DReach^{v'(\mathcal{A})}_{\ell_2}(\ell_1) = DReach^{v'(\mathcal{A})}_{\neg\ell_2}(\ell_1) = [1.5,3]$ and therefore $v'(\mathcal{A})$ is *(fully)* opaque w.r.t. ℓ_2 on the way to ℓ_1.

6 THE THEORY OF PARAMETRIC TIMED OPACITY W.R.T. EXECUTION TIMES

We address in this section the parametric problems of timed opacity w.r.t. execution times.

Let us consider the following decision problem, i. e., the problem of checking the *emptiness* of the parameter valuations and execution times set guaranteeing timed opacity w.r.t. execution times. The decision problem associated to *full* timed opacity will be considered in Section 7.

Timed opacity emptiness Problem:
Input: A PTA $\mathcal{A}$, a private location ℓ_{priv}, a target location ℓ_f
Problem: Is the set of valuations v such that $v(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ_f for a non-empty set of execution times empty?

The negation of the timed opacity emptiness consists in deciding whether there exists at least one parameter valuation for which $v(\mathcal{A})$ is opaque for at least some execution time.

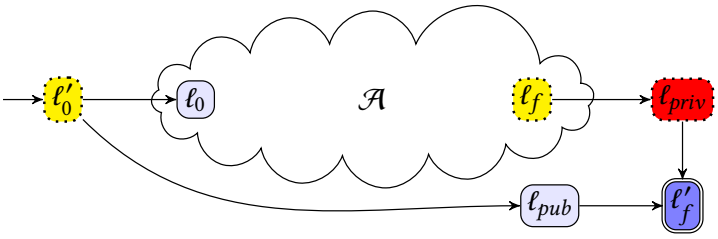


Fig. 7. Reduction from reachability-emptiness

6.1 Undecidability in general

We prove undecidability results for a "sufficient" number of clocks and parameters. Put it differently, our proofs of undecidability require a minimum number of clocks and parameters to work; the problems are obviously undecidable for larger numbers, but the decidability is open for smaller numbers. This will be briefly discussed in Section 10.

With the rule of thumb that all non-trivial decision problems are undecidable for general PTAs [50], the following result is not surprising, and follows from the undecidability of reachability-emptiness for PTAs.

Theorem 6.1 (undecidability). *The timed opacity emptiness problem is undecidable for general PTAs.*

Proof. We reduce from the reachability-emptiness problem, i. e., the existence of a parameter valuation for which there exists a run reaching a given location in a PTA, which is undecidable (e. g., [3, 51, 52, 46, 53]). Consider an arbitrary PTA $\mathcal{A}$ with initial location ℓ_0 and a given location ℓ_f. It is undecidable whether there exists a parameter valuation for which there exists a run reaching ℓ_f(proofs of undecidability in the literature generally reduce from the halting problem of a 2-counter machine which is undecidable [54], so one can see $\mathcal{A}$ as an encoding of a 2-counter machine).

Now, add the following locations and transitions (all unguarded) as in Fig. 7: a new urgent[4] initial location ℓ'_0 with outgoing transitions to ℓ_0 and to a new location ℓ_{pub}; a new urgent location ℓ_{priv} with an incoming transition from ℓ_f; a new final location ℓ'_f with incoming transitions from ℓ_{priv} and ℓ_{pub}. Also, ℓ_f is made urgent. Let $\mathcal{A}'$ denote this new PTA.

First note that, due to the unguarded transitions, ℓ'_f is reachable for any parameter valuation and for any execution time by runs going through ℓ_{pub} and not going through ℓ_{priv}. That is, for all v, $DReach^{v(\mathcal{A}')}_{\neg\ell_{priv}}(\ell'_f) = [0, \infty)$.

Assume there exists some parameter valuation v such that ℓ_f is reachable from ℓ_0 in $v(\mathcal{A})$ for some execution times D: then, due to our construction with additional urgent locations, ℓ_{priv} is reachable on the way to ℓ'_f in $v(\mathcal{A}')$ for the exact same execution times D. Therefore, $v(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ'_f for execution times D.

Conversely, if ℓ_f is not reachable from ℓ_0 in $\mathcal{A}$ for any valuation, then ℓ_{priv} is not reachable on the way to ℓ'_f for any valuation in $\mathcal{A}'$. Therefore, there is no valuation v such that $v(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ'_f for any execution time. Therefore, there exists a valuation v such that $v(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ'_f iff ℓ_f is reachable in $\mathcal{A}$—which is undecidable. □

[4]Where time cannot elapse (depicted in dotted yellow in our figures).

Remark 2. Our proof reduces from the reachability-emptiness problem, for which several undecidability proofs were proposed (notably [3, 51, 52, 46, 53]), with various flavors (numbers of parameters, integer- or dense-time, integer- or rational-valued parameters, etc.). See [50] for a survey. Notably, this means (from [53]) that Theorem 6.1 holds for PTAs with at least 3 clocks and a single parameter.

6.2 A decidable subclass

We now show that the timed opacity emptiness problem is decidable for the subclass of PTAs called L/U-PTAs [44]. Despite early positive results for L/U-PTAs [44, 55], more recent results (notably [46, 56, 57, 58]) mostly proved undecidable properties of L/U-PTAs, and therefore this positive result is welcome.

Theorem 6.2 (decidability). *The timed opacity emptiness problem is decidable for L/U-PTAs.*

Proof. We reduce to the timed opacity computation problem of a given TA, which is decidable (Proposition 5.2).

Let $\mathcal{A}$ be an L/U-PTA. Let $\mathcal{A}_{0,\infty}$ denote the structure obtained as follows: any occurrence of a lower-bound parameter is replaced with 0, and any occurrence of a conjunct $x \triangleleft p$ (where p is necessarily an upper-bound parameter) is deleted, i. e., replaced with **true**.

Let us show that the set of valuations v such that $v(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ_f for a non-empty set of execution times is non empty iff the solution to the timed opacity computation problem for $\mathcal{A}_{0,\infty}$ is non-empty.

- $\Rightarrow$ Assume there exists a valuation v such that $v(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ_f for a non-empty set of execution. Therefore, the solution to the timed opacity computation problem for $\mathcal{A}_{0,\infty}$ is non-empty. That is, there exists a duration d such that there exists a run of duration d such that ℓ_{priv} is reachable on the way to ℓ_f, and there exists a run of duration d such that ℓ_{priv} is unreachable on the way to ℓ_f.
  We now need the following monotonicity property of L/U-PTAs:

  Lemma 6.3 ([44]). *Let $\mathcal{A}$ be an L/U-PTA and v be a parameter valuation. Let v' be a valuation such that for each upper-bound parameter p^u, $v'(p^u) \geq v(p^u)$ and for each lower-bound parameter p^l, $v'(p^l) \leq v(p^l)$. Then any run of $v(\mathcal{A})$ is a run of $v'(\mathcal{A})$.*

  Therefore, from Lemma 6.3, the runs of $v(\mathcal{A})$ of duration d such that ℓ_{priv} is reachable (resp. unreachable) on the way to ℓ_f are also runs of $\mathcal{A}_{0,\infty}$. Therefore, there exists a non-empty set of durations such that $\mathcal{A}_{0,\infty}$ is opaque, i. e., solution to the timed opacity computation problem for $\mathcal{A}_{0,\infty}$ is non-empty.
- $\Leftarrow$ Assume the solution to the timed opacity computation problem for $\mathcal{A}_{0,\infty}$ is non-empty. That is, there exists a duration d such that there exists a run of duration d such that ℓ_{priv} is reachable on the way to ℓ_f in $\mathcal{A}_{0,\infty}$, and there exists a run of duration d such that ℓ_{priv} is unreachable on the way to ℓ_f in $\mathcal{A}_{0,\infty}$.
  The result could follow immediately—if only assigning 0 and ∞ to parameters was a proper parameter valuation. From [44, 55], if a location is reachable in the TA obtained by valuating lower-bound parameters with 0 and upper-bound parameters with ∞, then there exists a sufficiently large constant C such that this run exists in $v(\mathcal{A})$ such that v assigns 0 to lower-bound and C to upper-bound parameters. Here, we can trivially pick $d+1$, as any clock constraint $x \leq d+1$ or $x < d+1$ will be satisfied for a run of duration d. Let v assign 0 to lower-bound and d to upper-bound parameters. Then, there exists a run of duration d such that ℓ_{priv} is reachable on the way to ℓ_f in $v(\mathcal{A})$, and there exists a run of duration d such that ℓ_{priv} is unreachable on the way to ℓ_f in $v(\mathcal{A})$. Therefore, the set of valuations v such that

$v(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ_f for a non-empty set of execution times is non empty—which concludes the proof. □

Remark 3. The class of L/U-PTAs is known to be relatively meaningful, and many case studies from the literature fit into this class, including case studies proposed even before this class was defined in [44], e. g., a toy railroad crossing model and a model of Fischer mutual exclusion protocol given in [3] (see [50] for a survey). Even though the PTA in Fig. 5 does not fit in this class, it can easily be transformed into an L/U-PTA, by duplicating p into p^l (used in lower-bound comparisons with clocks) and p^u (used in upper-bound comparisons with clocks).

6.3 Intractability of synthesis for L/U-PTAs

Even though the *emptiness* problem for the timed opacity w.r.t. a set of execution times D is decidable for L/U-PTAs (Theorem 6.2), the *synthesis* of the parameter valuations remains intractable in general, as shown in the following Proposition 6.4. By intractable, we mean more precisely that the solution, if it can be computed, cannot (in general) be represented using any formalism for which the emptiness of the intersection with equality constraints is decidable. That is, a formalism in which it is decidable to answer the question "is the computed solution intersected with an equality test between variables empty?" cannot be used to represent the solution. By empty, we mean emptiness of the valuations set. For example, let us question whether we could represent the solution of the timed opacity synthesis problem for L/U-PTAs using the formalism of a *finite union of polyhedra*: testing whether a finite union of polyhedra intersected with "equality constraints" (typically $p_1 = p_2$) is empty or not *is* decidable. The Parma polyhedra library [59] can typically compute the answer to this question. Therefore, from the following Proposition 6.4, finite unions of polyhedra cannot be used to represent the solution of the timed opacity synthesis problem for L/U-PTAs. As finite unions of polyhedra are a very common formalism (not to say the *de facto* standard) to represent the solutions of various timing parameters synthesis problems, the synthesis is then considered to be infeasible in practice, or *intractable* (following the vocabulary used in [46, Theorem 2]).

Proposition 6.4 (intractability). *If it can be computed, the solution to the timed opacity synthesis problem for L/U-PTAs cannot in general be represented using any formalism for which the emptiness of the intersection with equality constraints is decidable.*

Proof. We reuse a reasoning inspired by [46, Theorem 2], and we reduce from the undecidability of the timed opacity emptiness problem for general PTAs (Theorem 6.1). Assume the solution of the timed opacity synthesis problem for L/U-PTAs can be represented in a formalism for which the emptiness of the intersection with equality constraints is decidable.

Assume an arbitrary PTA $\mathcal{A}$ with notably two locations ℓ_{priv} and ℓ_f. From $\mathcal{A}$, we define an L/U-PTA $\mathcal{A}'$ as follow: for each parameter p_i that is used both as an upper bound and a lower bound, replace its occurrences as upper bounds by a fresh parameter p_i^u and its occurrences as lower bounds by a fresh parameter p_i^l.

By assumption, the solution of the synthesis problem $\Gamma = \{(v, D_v) \mid v(\mathcal{A}')$ is opaque w.r.t. ℓ_{priv} on the way to ℓ_f for times $D_v\}$ for $\mathcal{A}'$ can be computed and represented in a formalism for which the emptiness of the intersection with equality constraints is decidable.

However, solving the emptiness of $\{(v, D_v) \in \Gamma \mid \bigwedge_i v(p_i^u) = v(p_i^l)\}$ (which is decidable by assumption), we can decide the timed opacity emptiness for the PTA $\mathcal{A}$ (which is undecidable from Theorem 6.1). This leads to a contradiction, and therefore the solution of the timed opacity synthesis

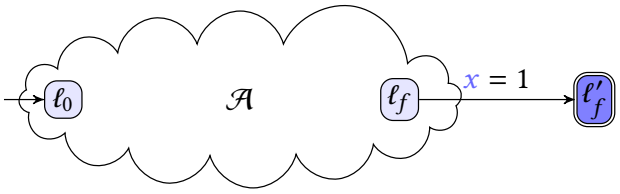


Fig. 8. Undecidability of EF-emptiness over constant time

problem for L/U-PTAs cannot in general be represented in a formalism for which the emptiness of the intersection with equality constraints is decidable. □

7 THE THEORY OF PARAMETRIC FULL TIMED OPACITY

We address here the following decision problem, which asks about the emptiness of the parameter valuations set guaranteeing full timed opacity.

Full timed opacity Emptiness Problem:
INPUT: A PTA $\mathcal{A}$, a private location ℓ_{priv}, a target location ℓ_f
PROBLEM: Is the set of valuations v such that $v(\mathcal{A})$ is fully opaque w.r.t. ℓ_{priv} on the way to ℓ_f empty?

Equivalently, we are interested in deciding whether there exists at least one parameter valuation for which $v(\mathcal{A})$ is fully opaque.

7.1 Undecidability in general

Considering that Theorem 6.1 shows that the undecidability of the timed opacity emptiness problem, the undecidability of the emptiness problem for the full timed opacity is not surprising, but does not follow immediately.

To prove this result (that will be stated formally in Theorem 7.2), we first need the following lemma stating that the reachability-emptiness (hereafter sometimes referred to as "EF-emptiness") problem is undecidable in constant time, i. e., for a fixed time bound T. That is, the following lemma shows that, given a PTA $\mathcal{A}$, a target location ℓ_T and a time bound T, it is undecidable whether the set of parameter valuations for which there exists a run reaching ℓ_T in exactly T time units is empty or not.

LEMMA 7.1 (REACHABILITY IN CONSTANT TIME). *The reachability-emptiness problem in constant time is undecidable for PTAs with 4 clocks and 2 parameters.*

PROOF. In [58, Theorem 3.12], we showed that the EF-emptiness problem is undecidable over bounded time for PTAs with (at least) 3 clocks and 2 parameters. That is, given a fixed bound T and a location ℓ_f, it is undecidable whether the set of parameter valuations for which at least one run reaches ℓ_f within T time units is empty or not.

We reduce the reachability in bounded time (i. e., in at most T time units) to the reachability in constant time (i. e., in exactly T time units). In this proof, we fix $T = 1$.

Assume a PTA $\mathcal{A}$ with a location ℓ_f. We define a PTA $\mathcal{A}'$ as in Fig. 8 by adding a new location ℓ'_f, and a transition from ℓ_f to ℓ'_f guarded by $x = 1$, where x is a new clock (initially 0), not used in $\mathcal{A}$ and therefore never reset in the automaton.

Let us show that there is no valuation such that ℓ_f is reachable in at most 1 time unit iff there is no valuation such that ℓ'_f is reachable exactly in 1 time unit.

⇐ Assume EF-emptiness holds in constant time for $\mathcal{A}'$, i. e., there exists no parameter valuation for which ℓ'_f is reachable in $T = 1$ time units. Then, from the construction of $\mathcal{A}'$, no parameter valuation exists for which ℓ_f is reachable in $T \leq 1$ time units.

⇒ Conversely, assume EF-emptiness holds over bounded time for $\mathcal{A}$, i. e., there exists no parameter valuation for which ℓ_f is reachable in $T \leq 1$ time units. Then, from the construction of $\mathcal{A}'$, no parameter valuation exists for which ℓ'_f is reachable in $T = 1$ time units.

This concludes the proof of the lemma. □

We can now state and prove Theorem 7.2.

Theorem 7.2 (undecidability). *The full timed opacity emptiness problem is undecidable for general PTAs with (at least) 4 clocks and 2 parameters.*

Proof. We reduce from the reachability-emptiness problem in constant time, which is undecidable (Lemma 7.1).

Consider an arbitrary PTA $\mathcal{A}$ with (at least) 4 clocks and 2 parameters, with initial location ℓ_0 and a given location ℓ_f. We add the following locations and transitions in $\mathcal{A}$ to obtain a PTA $\mathcal{A}'$, as in Fig. 9: a new urgent initial location ℓ'_0, with outgoing transition to ℓ_0 and to a new location ℓ_{pub}, a new urgent location ℓ_{priv} with an incoming transition from ℓ_f, a new urgent and final location ℓ'_f with incoming transitions from ℓ_{priv} and ℓ_{pub}, and a guard $x = 1$ (with a new clock x, never reset) on the transition from ℓ_{pub} to ℓ'_f.

First, note that, due to the guarded transition, ℓ'_f is reachable for any parameter valuation and (only) for an execution time equal to 1 by runs going through ℓ_{pub} and not going through ℓ_{priv}. That is, for all v, $DReach^{v(\mathcal{A})}_{\neg \ell_{priv}}(\ell_f) = \{1\}$.

We show that there exists a valuation v such that $v(\mathcal{A}')$ is fully opaque w.r.t. ℓ_{priv} on the way to ℓ'_f iff there exists a valuation v such that ℓ_f is reachable in $v(\mathcal{A})$ for execution time equal to 1.

⇐ Assume there exists some valuation v such that ℓ_f is reachable from ℓ_0 in $\mathcal{A}$ (only) for execution time equal to 1. Then, due to our construction, ℓ_{priv} is reachable on the way to ℓ'_f in $v(\mathcal{A}')$ for the exact same execution time 1. Therefore, $v(\mathcal{A}')$ is fully opaque w.r.t. ℓ_{priv} on the way to ℓ'_f.

⇒ Conversely, if ℓ_f is not reachable from ℓ_0 in $\mathcal{A}$ for any valuation for execution time 1, then ℓ_{priv} is not reachable on the way to ℓ'_f for any valuation of $\mathcal{A}'$. Therefore, there is no valuation v such that $v(\mathcal{A}')$ is fully opaque w.r.t. ℓ_{priv} on the way to ℓ'_f for execution time 1.

Therefore, there exists a valuation v such that $v(\mathcal{A}')$ is fully opaque w.r.t. ℓ_{priv} on the way to ℓ'_f iff there exists a valuation v such that ℓ_f is reachable in $v(\mathcal{A})$ for execution time equal to 1—which is undecidable. This concludes the proof.

Let us briefly discuss the minimum number of clocks necessary to obtain undecidability using our proof (the case of smaller numbers of clocks remains open). Recall that Lemma 7.1 needs 4 clocks; in the current proof of Theorem 7.2, we add a new clock x which is never reset; however, since the proof of Lemma 7.1 also uses a clock which is never reset, therefore we can reuse it, and our proof does not need an additional clock. So the result holds for 4 clocks and 2 parameters. □

7.2 Undecidability for L/U-PTAs

Note that reasoning like in Section 6.2, i. e., reducing the emptiness problem to a decision problem of the non-parametric $\mathcal{A}_{0,\infty}$, is not relevant. Fig. 10 shows an L/U-PTA $\mathcal{A}$ (and more precisely, a U-PTA, i. e., an L/U-PTA with an empty set of lower-bound parameters [55]) which is not fully opaque for any parameter valuation, but whose associated TA $\mathcal{A}_{0,\infty}$ is.

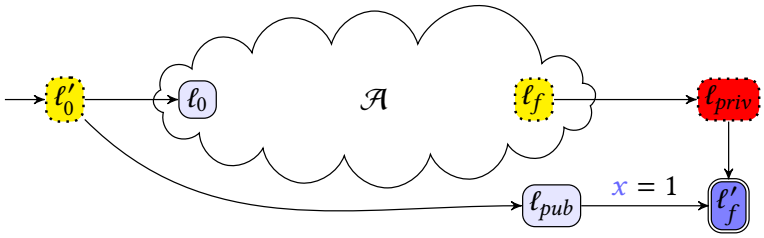


Fig. 9. Reduction from reachability-emptiness

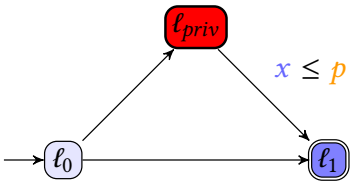


Fig. 10. $\mathcal{A}_{0,\infty}$ is not sufficient for the full timed opacity emptiness problem

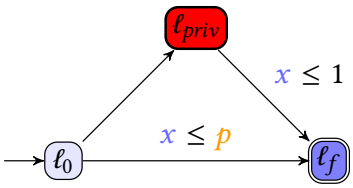


Fig. 11. No monotonicity for full timed opacity in L/U-PTAs

In addition, while it is well-known that L/U-PTAs enjoy a monotonicity for reachability properties ("enlarging an upper-bound parameter or decreasing a lower-bound parameter preserves reachability") as recalled in Lemma 6.3, we can show in the following example that this is not the case for full timed opacity.

Example 7.3. Consider the PTA in Fig. 11. First assume v such that $v(p) = 0.5$. Then, $v(\mathcal{A})$ is not full timed opaque: indeed, ℓ_f can be reached in 1 time unit via ℓ_{priv}, but not without going through ℓ_{priv}.

Second, assume v' such that $v'(p) = 1$. Then, $v'(\mathcal{A})$ is full timed opaque: indeed, ℓ_f can be reached for any duration in $[0, 1]$ by runs both going through and not going through ℓ_{priv}.

Finally, let us enlarge p further, and assume v'' such that $v''(p) = 2$. Then, $v''(\mathcal{A})$ becomes again not full timed opaque: indeed, ℓ_f can be reached in 2 time units not going through ℓ_{priv}, but cannot be reached in 2 time units by going through ℓ_{priv}.

As a side note, remark that this PTA is actually a U-PTA, that is, monotonicity for this problem does not even hold for U-PTAs.

In fact, we show that, while the timed opacity emptiness problem is decidable for L/U-PTAs (Theorem 6.2), the full timed opacity emptiness problem becomes undecidable for this same class (from 4 parameters). This confirms (after previous works in [46, 56, 57]) that L/U-PTAs stand at the frontier between decidability and undecidability.

Theorem 7.4. *The full timed opacity emptiness problem is undecidable for L/U-PTAs with (at least) 4 clocks and 4 parameters.*

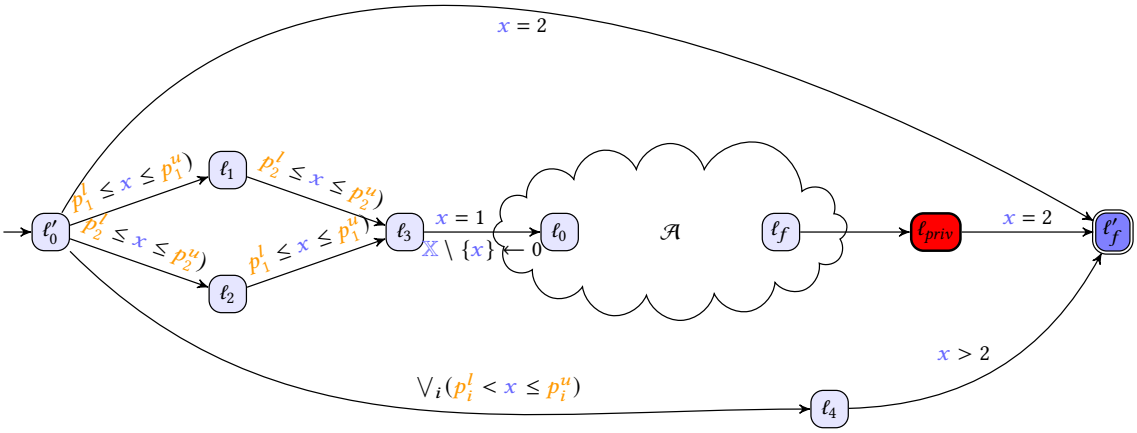


Fig. 12. Undecidability of full timed opacity emptiness for L/U-PTAs

Proof. Let us recall from [58, Theorem 3.12] that the EF-emptiness problem is undecidable over bounded time for PTAs with (at least) 3 clocks and 2 parameters. Assume a PTA $\mathcal{A}$ with 3 clocks and 2 parameters, say p_1 and p_2, and a target location ℓ_f. Take 1 as a time bound. From [58, Theorem 3.12], it is undecidable whether there exists a parameter valuation for which ℓ_f is reachable in time ≤ 1.

The idea of our proof is that, as in [46, 60], we "split" each of the two parameters used in $\mathcal{A}$ into a lower-bound parameter (p_1^l and p_2^l) and an upper-bound parameter (p_1^u and p_2^u). Each construction of the form $x < p_i$ (resp. $x \leq p_i$) is replaced with $x < p_i^u$ (resp. $x \leq p_i^u$) while each construction of the form $x > p_i$ (resp. $x \geq p_i$) is replaced with $x > p_i^l$ (resp. $x \geq p_i^l$); $x = p_i$ is replaced with $p_i^l \leq x \leq p_i^u$.

The idea is that the PTA $\mathcal{A}$ is exactly equivalent to our construction with duplicated parameters only when $p_1^l = p_1^u$ and $p_2^l = p_2^u$. The crux of the rest of this proof is that we will "rule out" any parameter valuation not satisfying these equalities, so as to use directly the undecidability result of [58, Theorem 3.12].

Now, consider the extension of $\mathcal{A}$ given in Fig. 12, and let $\mathcal{A}'$ be this extension. We assume that x is an extra clock not used in $\mathcal{A}$. The syntax "$\mathbb{X} \setminus \{x\} \leftarrow 0$" denotes that all clocks of the original PTA $\mathcal{A}$ are reset—but not the new clock x. The guard on the lower transition from ℓ'_0 to ℓ_4 stands for 2 different transitions guarded with $p_1^l < x \leq p_1^u$, and $p_2^l < x \leq p_2^u$, respectively. Let us first make the following observations:

(1) for any parameter valuation, one can take the upper transition from ℓ'_0 to ℓ'_f at time 2, i. e., ℓ'_f is always reachable in time 2 without going through location ℓ_{priv};
(2) the original automaton $\mathcal{A}$ can only be entered whenever $p_1^l \leq p_1^u$ and $p_2^l \leq p_2^u$; going from ℓ'_0 to ℓ_0 takes exactly 1 time unit (due to the $x = 1$ guard);
(3) if a run reaches ℓ_{priv} on the way to ℓ'_f, then its duration is necessarily 2;
(4) from [58, Theorem 3.12], it is undecidable whether there exists a parameter valuation for which there exists a run reaching ℓ_f from ℓ_0 in time ≤ 1, i. e., reaching ℓ_f from ℓ'_0 in time ≤ 2.

Let us consider the following cases:

(1) if $p_1^l > p_1^u$ or $p_2^l > p_2^u$, then thanks to the transitions from ℓ_0' to ℓ_0, there is no way to enter the original PTA $\mathcal{A}$ (and therefore to reach ℓ_{priv} on the way to ℓ_f'); since these valuations can still reach ℓ_f' for some execution times (notably $x = 2$ through the upper transition from ℓ_0' to ℓ_f'), then $\mathcal{A}'$ is not fully opaque for any of these valuations.
(2) if $p_1^l < p_1^u$ or $p_2^l < p_2^u$, then one of the lower transitions from ℓ_0' to ℓ_4 can be taken, and therefore ℓ_f' is reachable in a time > 2 without going through ℓ_{priv}. Since no run can reach ℓ_f' while going through ℓ_{priv} for a duration $\neq 2$, then again $\mathcal{A}'$ is not fully opaque for any of these valuations.
(3) if $p_1^l = p_1^u$ and $p_2^l = p_2^u$, then the behavior of the modified $\mathcal{A}$ (with duplicate parameters) is exactly the one of the original $\mathcal{A}$. Also, note that the lower transitions from ℓ_0' to ℓ_f' (via ℓ_4) cannot be taken. In contrast, the upper transition from ℓ_0' to ℓ_f' can still be taken, and therefore there exists a run of duration 2 reaching ℓ_f' without visiting ℓ_{priv}.
Now, assume there exists a parameter valuation for which there exists a run of $\mathcal{A}$ of duration ≤ 1 reaching ℓ_f. And, as a consequence, ℓ_{priv} is reachable, and therefore there exists some run of duration 2 (including the 1 time unit to go from ℓ_0 to ℓ_0') reaching ℓ_f' after visiting ℓ_{priv}. From the above reasoning, all runs reaching ℓ_f' have duration 2; in addition, we exhibited a run visiting ℓ_{priv} and a run not visiting ℓ_{priv}; therefore the modified automaton $\mathcal{A}'$ is fully opaque for such a parameter valuation.
Conversely, assume there exists no parameter valuation for which there exists a run of $\mathcal{A}$ of duration ≤ 1 reaching ℓ_f. In that case, $\mathcal{A}'$ is not fully opaque for any parameter valuation.

As a consequence, there exists a parameter valuation v' for which $v'(\mathcal{A}')$ is fully opaque iff there exists a parameter valuation v for which there exists a run in $v(\mathcal{A})$ of duration ≤ 1 reaching ℓ_f—which is undecidable from [58, Theorem 3.12]. □

8 PARAMETER SYNTHESIS FOR OPACITY

Despite the negative theoretical result of Theorem 6.1, we now address the timed opacity synthesis problem for the full class of PTAs. Our method may not terminate (due to the undecidability) but, if it does, its result is correct. Our workflow can be summarized as follows.

(1) We enrich the original PTA by adding a Boolean flag b and a final synchronization action;
(2) We perform *self-composition* (i. e., parallel composition with a copy of itself) of this modified PTA;
(3) We perform reachability-synthesis using EFsynth on ℓ_f with contradictory values of b.

We detail each operation in the following.
In this section, we assume a PTA $\mathcal{A}$, a given private location ℓ_{priv} and a given final location ℓ_f.

8.1 Enriching the PTA

We first add a Boolean flag b initially set to false, and then set to true on any transition whose target location is ℓ_{priv} (in the line of the proof of Proposition 5.2). Therefore, b = true denotes that ℓ_{priv} has been visited. Second, we add a synchronization action finish on any transition whose target location is ℓ_f. Third, we add a new clock x_{abs} (never reset) together with a new parameter p_{abs}, and we guard all transitions to ℓ_f with $x_{abs} = p_{abs}$. This will allow to measure the (parametric) execution time. Let $\mathsf{Enrich}(\mathcal{A}, \ell_{priv}, \ell_f)$ denote this procedure.

Example 8.1. Fig. 13 shows the transformed version of the PTA in Fig. 5.

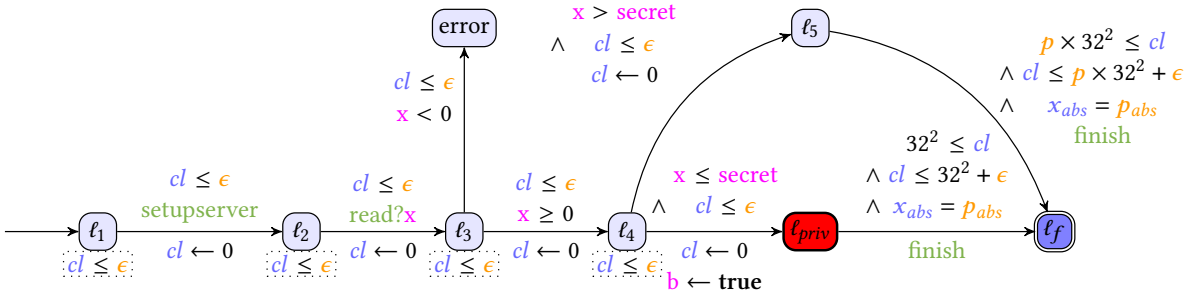


Fig. 13. Transformed version of Fig. 5

Algorithm 1: SynthOp($\mathcal{A}, \ell_{priv}, \ell_f$)

input : A PTA $\mathcal{A}$ with parameters set $\mathbb{P}$, locations ℓ_{priv}, ℓ_f
output: Parameter constraint K over $\mathbb{P} \cup \{p_{abs}\}$

1 $\mathcal{A}' \leftarrow$ Enrich($\mathcal{A}, \ell_{priv}, \ell_f$)
2 $\mathcal{A}'' \leftarrow \mathcal{A}' \parallel_{\{\text{finish}\}}$ Copy($\mathcal{A}'$)
3 **return** EFsynth$\Big(\mathcal{A}'', \{(\ell_f \wedge b = \text{true}, \ell'_f \wedge b' = \text{false})\}\Big)$

8.2 Self-composition

We use here the principle of *self-composition* [61], i. e., composing the PTA with a copy of itself. More precisely, given a PTA $\mathcal{A}' = \text{Enrich}(\mathcal{A}, \ell_{priv}, \ell_f)$, we first perform an identical copy of $\mathcal{A}'$ *with distinct variables*: that is, a clock x of $\mathcal{A}'$ is distinct from a clock x in the copy of $\mathcal{A}'$—which can be trivially performed using variable renaming.[5] Let Copy($\mathcal{A}'$) denote this copy of $\mathcal{A}'$. We then compute $\mathcal{A}' \parallel_{\{\text{finish}\}} \text{Copy}(\mathcal{A}')$. That is, $\mathcal{A}'$ and Copy($\mathcal{A}'$) evolve completely independently due to the interleaving—except that they are forced to enter ℓ_f at the same time, thanks to the synchronization action finish.

8.3 Synthesis

Then, we apply reachability synthesis EFsynth (over all parameters, i. e., the "internal" timing parameters, but also the p_{abs} parameter) to the following goal location: the original $\mathcal{A}'$ is in ℓ_f with $b = \text{true}$ while its copy Copy($\mathcal{A}'$) is in ℓ'_f with $b' = \text{false}$ (primed variables denote variables from the copy). Intuitively, we synthesize timing parameters and execution times such that there exists a run reaching ℓ_f with $b = \text{true}$ (i. e., that has visited ℓ_{priv}) and there exists another run of same duration reaching ℓ_f with $b = \text{false}$ (i. e., that has not visited ℓ_{priv}).

Let SynthOp($\mathcal{A}, \ell_{priv}, \ell_f$) denote the entire procedure. We formalize SynthOp in Algorithm 1, where "$\ell_f \wedge b = \text{true}$" denotes the location ℓ_f with $b = \text{true}$. Recall that p_{abs} is added by the enrichment step described in Section 8.1. The set of execution times D is therefore given by the possible valuations of p_{abs}; these valuations may depend on the model timing parameters (in the form of a constraint). Finally note that EFsynth is called on a set made of a single location of $\mathcal{A}' \parallel_{\{\text{finish}\}} \text{Copy}(\mathcal{A}')$; by definition of the synchronous product, this location is a *pair* of locations, one from $\mathcal{A}'$ (i. e., "$\ell_f \wedge b = \text{true}$") and one from Copy($\mathcal{A}'$) (i. e., "$\ell'_f \wedge b' = \text{false}$").

[5] In fact, the fresh clock x_{abs} and parameter p_{abs} can be shared to save two variables, as x_{abs} is never reset, and both PTAs enter ℓ_f at the same time, therefore both "copies" of x_{abs} and p_{abs} always share the same values.

Example 8.2. Consider again the PTA $\mathcal{A}$ in Fig. 5: its enriched version $\mathcal{A}'$ is given in Fig. 13. Fix $v(\epsilon) = 1$, $v(p) = 2$. We then perform the synthesis applied to the self-composition of $\mathcal{A}'$ according to Algorithm 1. The result obtained with IMITATOR is: $p_{abs} = \emptyset$ (as expected from Example 4.4).

Now fix $v(\epsilon) = 2$, $v(p) = 1.002$. We obtain: $p_{abs} \in [1026.048, 1034]$ (again, as expected from Example 4.4).

Now let us keep all parameters unconstrained. The result of Algorithm 1 is the following 3-dimensional constraint:

$$\begin{array}{ll} & 5 \times \epsilon + 1024 \geq p_{abs} \geq 1024 \\ \wedge & 1024 \times p + 5 \times \epsilon \geq p_{abs} \geq 1024 \times p \geq 0 \end{array}$$

8.4 Correctness

We will state below that, whenever $\mathsf{SynthOp}(\mathcal{A}, \ell_{priv}, \ell_f)$ terminates, then its result is an exact (sound and complete) answer to the timed opacity synthesis problem.

Let us first prove a technical lemma used later to prove the soundness of SynthOp.

Lemma 8.3. *Assume $\mathsf{SynthOp}(\mathcal{A}, \ell_{priv}, \ell_f)$ terminates with result K. For all $v \models K$, there exists a run ending in ℓ_f at time $v(p_{abs})$ in $v(\mathcal{A})$.*

Proof. From the construction of the procedure Enrich, we added a new clock x_{abs} (never reset) together with a new parameter p_{abs}, and we guarded all transitions to ℓ_f with $x_{abs} = p_{abs}$. Therefore, valuations of p_{abs} correspond exactly to the times at which ℓ_f can be reached in $v(\mathcal{A})$. □

We can now prove soundness and completeness.

Proposition 8.4 (soundness). *Assume $\mathsf{SynthOp}(\mathcal{A}, \ell_{priv}, \ell_f)$ terminates with result K. For all $v \models K$, there exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is reachable on the way to ℓ_f in $v(\mathcal{A})$ and there exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is avoided on the way to ℓ_f in $v(\mathcal{A})$.*

Proof. $\mathsf{SynthOp}(\mathcal{A}, \ell_{priv}, \ell_f)$ is the result of EFsynth called on the self-composition of $\mathsf{Enrich}(\mathcal{A}, \ell_{priv}, \ell_f)$. Recall that Enrich has enriched $\mathcal{A}$ with the addition of a guard $x_{abs} = p_{abs}$ on the incoming transitions of ℓ_f, as well as a Boolean flag b that is true iff ℓ_{priv} was visited along a run. Assume $v \models K$. From Lemma 3.16, there exists a run of $\mathcal{A}''$ reaching $\ell_f \wedge b = \text{true}, \ell'_f \wedge b' = \text{false}$. From Lemma 8.3, this run takes $v(p_{abs})$ time units. From the self-composition that is made of interleaving only (except for the final synchronization), there exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is reachable on the way to ℓ_f in $v(\mathcal{A})$ and there exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is avoided on the way to ℓ_f in $v(\mathcal{A})$. □

Proposition 8.5 (completeness). *Assume $\mathsf{SynthOp}(\mathcal{A}, \ell_{priv}, \ell_f)$ terminates with result K. Assume v. Assume there exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is reachable on the way to ℓ_f in $v(\mathcal{A})$ and there exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is avoided on the way to ℓ_f in $v(\mathcal{A})$. Then $v \models K$.*

Proof. Assume $\mathsf{SynthOp}(\mathcal{A}, \ell_{priv}, \ell_f)$ terminates with result K. Assume v. Assume there exists a run ρ of duration $v(p_{abs})$ such that ℓ_{priv} is reachable on the way to ℓ_f in $v(\mathcal{A})$ and there exists a run ρ' of duration $v(p_{abs})$ such that ℓ_{priv} is avoided on the way to ℓ_f in $v(\mathcal{A})$.

First, from Enrich, there exists a run ρ of duration $v(p_{abs})$ such that ℓ_{priv} is reachable (resp. avoided) on the way to ℓ_f in $v(\mathcal{A})$ implies that there exists a run ρ of duration $v(p_{abs})$ such that $\ell_f \wedge b = \text{true}$ (resp. $b = \text{false}$) is reachable in $v(\mathsf{Enrich}(\mathcal{A}))$.

Since our self-composition allows any interleaving, runs ρ of $v(\mathcal{A}')$ and ρ' in $v(\mathsf{Copy}(\mathcal{A}'))$ are independent—except for reaching ℓ_f. Since ρ and ρ' have the same duration $v(p_{abs})$, then they both reach ℓ_f at the same time and, from our definition of self-composition, they can simultaneously fire

action finish and enter ℓ_f at time $v(p_{abs})$. Hence, there exists a run reaching $\ell_f \wedge b = \text{true}$, $\ell'_f \wedge b' = \text{false}$ in $v(\mathcal{A}'')$.

Finally, from Lemma 3.16, $v \models K$. □

Theorem 8.6 (correctness). *Assume SynthOp$(\mathcal{A}, \ell_{priv}, \ell_f)$ terminates with result K. Assume v. The following two statements are equivalent:*

(1) There exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is reachable on the way to ℓ_f in $v(\mathcal{A})$ and there exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is avoided on the way to ℓ_f in $v(\mathcal{A})$.

(2) $v \models K$.

Proof. From Propositions 8.4 and 8.5 □

9 EXPERIMENTS

9.1 Experimental environment

We use IMITATOR [49], a parametric timed model checking tool taking as input networks of PTAs extended with several handful features such as shared global discrete variables, PTA synchronization through strong broadcast, non-timing rational-valued parameters, etc. IMITATOR supports various parameter synthesis algorithms, including reachability synthesis. IMITATOR represents symbolic states as polyhedra, relying on PPL [59]. IMITATOR is a leading tool for parameter synthesis for extensions of parametric timed automata. Related tools are Romeo [62] (which cannot be used here, as it does not support parametric timed automata, but extensions of Petri nets), SpaceEx [18] (which does not perform parameter synthesis), or Uppaal [22] (which cannot be used here, as our algorithm requires timing parameters, not supported by Uppaal).

We ran experiments using IMITATOR 2.10.4 "Butter Jellyfish" (build 2477 `HEAD/5b53333`) on a Dell XPS 13 9360 equipped with an Intel® Core™ i7-7500U CPU @ 2.70GHz with 8 GiB memory running Linux Mint 18.3 64 bits.[6]

9.2 Translating programs into PTAs

We will consider case studies from the PTA community and from previous works focusing on privacy using (parametric) timed automata. In addition, we will be interested in analyzing programs too. In order to apply our method to the analysis of programs, we need a systematic way of translating a program (e. g., a Java program) into a PTA. In general, precisely modeling the execution time of a program using models like timed automata is highly non-trivial due to complication of hardware pipelining, caching, OS scheduling, etc. The readers are referred to the rich literature in, for instance, [63]. In this work, we instead make the following simplistic assumption on execution time of a program statement and focus on solving the parameter synthesis problem. How to precisely model the execution time of programs is orthogonal and complementary to our work.

We assume that the execution time of a program statement other than `Thread.sleep(n)` is within a range $[0, \epsilon]$ where ϵ is a small integer constant (in milliseconds), whereas the execution time of statement `Thread.sleep(n)` is within a range $[n, n + \epsilon]$. In fact, we choose to keep ϵ *parametric* to be as general as possible, and to not depend on particular architectures.

Our test subject is a set of benchmark programs from the DARPA Space/Time Analysis for Cybersecurity (STAC) program.[7] These programs are being released publicly to facilitate researchers to develop methods and tools for identifying STAC vulnerabilities in the programs. These programs are simple yet non-trivial, and were built on purpose to highlight vulnerabilities that can be easily missed by existing security analysis tools.

[6]Sources, models and results are available at doi.org/10.5281/zenodo.3251141 and imitator.fr/static/ATVA19/.
[7]https://github.com/Apogee-Research/STAC/

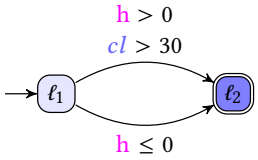


Fig. 14. [20, Fig. 5]

9.3 A richer framework

The symbolic representation of variables and parameters in IMITATOR allows us to reason *symbolically* concerning variables. That is, instead of enumerating all possible (bounded) values of x and secret in Fig. 5, we turn them to parameters (i. e., unknown constants), and IMITATOR performs a symbolic reasoning. Even better, the analysis terminates for this example even when no bound is provided on these variables. This is often not possible in (non-parametric) timed automata based model checkers, which usually have to enumerate these values. Therefore, in our PTA representation of Java programs, we turn all user-input variable and secret constant variables to non-timing rational-valued parameters, also supported by IMITATOR. Other local variables are implemented using IMITATOR discrete (shared, global) variables.

We also discuss how to enlarge the scope of our framework.

Multiple private locations. This can be easily achieved by setting b to true along any incoming transition of one of these private locations.

Multiple final locations. The technique used depends on whether these multiple final locations can be distinguished or not. If they are indistinguishable (i. e., the observer knows when the program has terminated, but not in which state), then it suffices to merge all these final locations in a single one, and our framework trivially applies. If they are distinguishable, then one analysis needs to be conducted on each of these locations (with a different parameter p_{abs} for each of these), and the obtained constraints must be intersected.

Access to high-level variables. In the literature, a distinction is sometimes made between low-level ("public") and high-level ("private") variables. Opacity or non-interference can be defined in terms of the ability for an observer to deduce some information on the high-level variables.

Example 9.1. For example, in Fig. 14 (where cl is a clock and h a variable), if ℓ_2 is reachable in 20 time units, then it is clear that the value of the high-level variable h is negative.

Our framework can also be used to address this problem, e. g., by setting b to true, not on locations but on selected tests / valuations of such variables.

Example 9.2. For example, setting b to true on the upper transition from ℓ_1 to ℓ_2 in Fig. 14, the answer to the timed opacity computation problem is $D = (30, \infty)$, and the system is therefore not opaque since ℓ_2 can be reached for any execution time in $[0, \infty)$.

9.4 Experiments

9.4.1 Benchmarks. As a proof of concept, we applied our method to a set of examples from the literature. The first five models come from previous works from the literature [11, 10, 20], also addressing non-interference or opacity in timed automata.[8] In addition, we used two common

[8] As most previous works on opacity and timed automata do not come with an implementation nor with benchmarks, it is not easy to find larger models coming in the form of TAs.

Table 1. Experiments: timed opacity

Model			Transf. PTA			Result	
Name	$\|\mathcal{A}\|$	$\|\mathbb{X}\|$	$\|\mathcal{A}\|$	$\|\mathbb{X}\|$	$\|\mathbb{P}\|$	Time (s)	Opaque?
[20, Fig. 5]	1	1	2	3	3	0.02	(×)
[11, Fig. 1b]	1	1	2	3	1	0.04	(×)
[11, Fig. 2a]	1	1	2	3	1	0.05	(×)
[11, Fig. 2b]	1	1	2	3	1	0.02	(×)
Web privacy problem [10]	1	2	2	4	1	0.07	(×)
Coffee	1	2	2	5	1	0.05	√
Fischer-HSRV02	3	2	6	5	1	5.83	(×)
STAC:1:n			2	3	6	0.12	(×)
STAC:1:v			2	3	6	0.11	×
STAC:3:n			2	3	8	0.72	√
STAC:3:v			2	3	8	0.74	(×)
STAC:4:n			2	3	8	6.40	×
STAC:4:v			2	3	8	265.52	×
STAC:5:n			2	3	6	0.24	√
STAC:11A:v			2	3	8	47.77	(×)
STAC:11B:v			2	3	8	59.35	(×)
STAC:12c:v			2	3	8	18.44	×
STAC:12e:n			2	3	8	0.58	×
STAC:12e:v			2	3	8	1.10	(×)
STAC:14:n			2	3	8	22.34	(×)

models from the (P)TA literature, not necessarily linked to security: a toy coffee machine (Coffee) used as benchmark in a number of papers, and a model Fischer's mutual exclusion protocol (Fischer-HRSV02) [44]. In both cases, we added manually a definition of private location (the number of sugars ordered, and the identity of the process entering the critical section, respectively), and we verified whether they are opaque w.r.t. these internal behaviors.

We also applied our approach to a set of Java programs from the aforementioned STAC library. We use identifiers of the form STAC:1:n where 1 denotes the identifier in the library, while n (resp. v) denotes non-vulnerable (resp. vulnerable). We manually translated these programs to PTAs, following the method described in Section 9.2. We used a representative set of programs from the library; however, some of them were too complex to fit in our framework, notably when the timing leaks come from calls to external libraries (STAC:15:v), when dealing with complex computations such as operations on matrices (STAC:16:v) or when handling probabilities (STAC:18:v). Proposing efficient and accurate ways to represent arbitrary programs into (parametric) timed automata is orthogonal to our work, and is the object of future works.

9.4.2 Timed opacity computation. First, we *verified* whether a given TA model is opaque, i. e., if for all execution times reaching a given final location, both an execution goes through a given private location and an execution does not go through this private location. To this end, we also answer the timed opacity computation problem, i. e., to synthesize all execution times for which the system is opaque. While this problem can be verified on the region graph (Proposition 5.2), we use the same framework as in Section 8, but without parameters in the original TA. That is, we use the Boolean flag b and the parameter p_{abs} to compute all possible execution times. In other words, we use a parametric analysis to solve a non-parametric problem.

We tabulate the experiments results in Table 1. We give from left to right the model name, the numbers of automata and of clocks in the original timed automaton (this information is not relevant

for Java programs as the original model is not a TA), the numbers of automata[9], of clocks and of parameters in the transformed PTA, the computation time in seconds (for the timed opacity computation problem), and the result. In the result column, "√" (resp. "×") denotes that the model is opaque (resp. is not opaque, i. e., vulnerable), while "(×)" denotes that the model is not opaque, but could be fixed. That is, although $DReach^{v(\mathcal{A})}_{\ell_{priv}}(\ell_f) \neq DReach^{v(\mathcal{A})}_{\neg\ell_{priv}}(\ell_f)$, their intersection is non-empty and therefore, by tuning the computation time, it may be possible to make the system opaque. This will be discussed in Section 9.5.

Even though we are interested here in timed opacity computation (and not in synthesis), note that all models derived from Java programs feature the parameter ϵ. The result is obtained by variable elimination, i. e., by existential quantification over the parameters different from p_{abs}. In addition, the number of parameters is increased by the parameters encoding the symbolic variables (such as x and secret in Fig. 5).

Concerning the Java programs, we decided to keep the most abstract representation, by imposing that each instruction lasts for a time in $[0, \epsilon]$, with ϵ a parameter. However, fixing an identical (parametric) time ϵ for all instructions, or fixing an arbitrary time in a constant interval $[0, \epsilon]$ (for some constant ϵ, e. g., 1), or even fixing an identical (constant) time ϵ (e. g., 1) for all instructions, significantly speeds up the analysis. These choices can be made for larger models.

Discussion. Overall, our method is able to answer the timed opacity computation problem for practical case studies, exhibiting which execution times are opaque (timed opacity computation problem), and whether *all* execution times indeed guarantee opacity (timed opacity problem).

In many cases, while the system is not opaque, we are able to *infer* the execution times guaranteeing opacity (cells marked "(×)"). This is an advantage of our method w.r.t. methods outputting only binary answers.

We observed some mismatches in the Java programs, i. e., some of the programs marked n (non-vulnerable) in the library are actually vulnerable according to our method. This mainly comes from the fact that the STAC library expect tools to use imprecise analyses on the execution times, while we use an exact method. Therefore, a very small mismatch between $DReach^{v(\mathcal{A})}_{\ell_{priv}}(\ell_f)$ and $DReach^{v(\mathcal{A})}_{\neg\ell_{priv}}(\ell_f)$ will lead our algorithm to answer "not opaque", while some methods may not be able to differentiate this mismatch from imprecision (noise). This is notably the case of STAC:14:n where some action lasts either 5,010,000 or 5,000,000 time units depending on some secret, which our method detects to be different, while the library does not. For STAC:1:n, using our data, the difference in the execution time upper bound between an execution performing some secret action and an execution not performing it is larger than 1 %, which we believe is a value which is not negligible, and therefore this case study might be considered as vulnerable.

STAC:4:n requires a more detailed discussion. This particular program is targeting vulnerabilities that can be detected easily *when they accumulate*, typically in loops. This program checks a number of times (10) a user-input password, and each password check is made in the most insecure way, i. e., by returning "incorrect" as soon as one character differs between the input password and the expected password. This way is very insecure because the execution time is proportional to the number of consecutive correct characters in the input password and, by observing the execution time, an attacker can guess how many characters are correct, and therefore using a limited number of tests, (s)he will eventually guess the correct password. The difference between the vulnerable (STAC:4:v) and the non-vulnerable (STAC:4:n) versions is that the non-vulnerable version immediately stops if the password is incorrect, and performs the 10 checks only if the

[9]As usual, it may be simpler to write PTA models as a network of PTAs. Recall from Definition 3.5 that a network of PTAs gives a PTA. In this case, $|\mathcal{A}|$ denotes the number of input PTA components.

password is correct. Therefore, while the computation time is very different between the correct input password and any incorrect input password, it is however very similar between an incorrect input password that would only be incorrect because, say, of the last character (e. g., "kouignamaz" while the expected password is "kouignaman"), and a completely incorrect input password differing as early as the first character (e. g., "andouille"). This makes the attacker's task very difficult. The main reason for the STAC library to label STAC:4:n as a non-vulnerable program is because of the "very similar" nature of the computation times between an incorrect input password that would only be incorrect because of the last character, and a completely incorrect input password. (In contrast, the vulnerable version STAC:4:v is completely vulnerable because this time difference is amplified by the loop, here 10 times.) While "very similar" might be acceptable for most tools, in our setting based on formal verification, we *do* detect that testing "kouignamaz" or testing "kouignamzz" will yield a slightly faster computation time for the second input, because the first incorrect letter occurs earlier—and the program is therefore vulnerable.

9.4.3 Timed opacity synthesis. Then, we address the timed opacity synthesis problem. In this case, we *synthesize* both the execution time and the internal values of the parameters for which one cannot deduce private information from the execution time.

We consider the same case studies as for timed opacity computation; however, the Java programs feature no internal "parameter" and cannot be used here. Still, as a proof of concept, we artificially enriched one of them (STAC:3:v) as follows: in addition to the parametric value of ϵ and the execution time, we parameterized one of the sleep timers. The resulting constraint can help designers to refine this latter value to ensure opacity. Note that it may not be that easy to tune a Java program to make it non-opaque: while this is reasonably easy on the PTA level (restraining the execution times using an additional clock), this may not be clear on the original model: Making a program terminate slower than originally is easy with a Sleep statement; but making it terminate "earlier" is less obvious, as it may mean an abrupt termination, possibly leading to wrong results.

We tabulate the results in Table 2, where the columns are similar to Table 1. A difference is that the first $|\mathbb{P}|$ column denotes the number of parameters in the original model (without counting these added by our transformation). In addition, Table 2 does not contain a "vulnerable?" column as we *synthesize* the condition for which the model is non-vulnerable, and therefore the answer is non-binary. However, in the last column ("Constraint"), we make explicit whether no valuations ensure opacity ("$\bot$"), all of them ("$\top$"), or some of them ("K").

Discussion. An interesting outcome is that the computation time is comparable to the (non-parametric) timed opacity computation, with an increase of up to 20 % only. In addition, for all case studies, we exhibit at least some valuations for which the system can be made opaque. Also note that our method always terminates for these models, and therefore the result exhibited is complete. Interestingly, Coffee is opaque for any valuation of the 3 internal parameters.

9.5 "Repairing" a non-opaque PTA

Our method gives a result in time of a union of polyhedra over the internal timing parameters and the execution time. On the one hand, we believe tuning the internal timing parameters should be easy: for a program, an internal timing parameter can be the duration of a sleep, for example. On the other hand, tuning the execution time of a program may be more subtle. A solution is to enforce a minimal execution time by adding a second thread in parallel with a Wait() primitive to ensure a minimal execution time. Ensuring a *maximal* execution time can be achieved with an exception stopping the program after a given time; however there is a priori no guarantee that the result of the computation is correct.

Table 2. Experiments: timed opacity synthesis

Model				Transf. PTA			Result	
Name	$\lvert\mathcal{A}\rvert$	$\lvert\mathbb{X}\rvert$	$\lvert\mathbb{P}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathbb{X}\rvert$	$\lvert\mathbb{P}\rvert$	Time (s)	Constraint
[20, Fig. 5]	1	1	0	2	3	4	0.02	K
[11, Fig. 1b]	1	1	0	2	3	3	0.03	K
[11, Fig. 2]	1	1	0	2	3	3	0.05	K
Web privacy problem [10]	1	2	2	2	4	3	0.07	K
Coffee	1	2	3	2	5	4	0.10	$\top$
Fischer-HSRV02	3	2	2	6	5	3	7.53	K
STAC:3:v			2	2	3	9	0.93	K

10 CONCLUSION

In this work, we proposed an approach based on parametric timed model checking to not only decide whether the model of a timed system can be subject to timing information leakage, but also to *synthesize* internal timing parameters and execution times that render the system opaque. We implemented our approach in a framework based on IMITATOR, and performed experiments on case studies from the literature and from a library of Java programs.

We now discuss future works in the following.

Theory. We proved decidability of the timed opacity computation problem (Proposition 5.2) and of the full timed opacity decision problem (Proposition 5.3) for TAs, but we only provided an upper bound (3EXPTIME) on the complexity. It can be easily shown that these problems are at least PSPACE, but the exact complexity remains to be exhibited.

In addition, the decidability of several "low-dimensional" problems (i. e., with "small" number of clocks or parameters) remains open. Among these, the one-clock case for parametric timed opacity emptiness (Theorem 6.1) remains open: that is, is the timed opacity emptiness problem decidable for PTAs using a single clock? Our method in Section 8 consists in duplicating the automaton and adding a clock that is never reset, thus resulting in a PTA with 3 clocks, for which reachability-emptiness is undecidable [3]. However, since one of the clocks is never reset, and since the automaton is structurally constrained (it is the result of the composition of two copies of the same automaton), decidability might be envisioned. Recall that the 2-clock reachability-emptiness problem is a famous open problem [50], despite recent advances, notably over discrete time [64, 65]. The 1-clock question also remains open for full timed opacity emptiness (Theorem 7.2). The minimum number of parameters required for our proof of the undecidability of the full timed opacity emptiness problem for PTAs (resp. L/U-PTAs) to work is 2 (resp. 4), as seen in Theorem 7.2 (resp. Theorem 7.4); it is open whether using less parameters can render these problems decidable.

Finally, concerning L/U-PTAs, we proved two negative results, despite the decidability of the timed opacity emptiness problem (Theorem 6.2): the undecidability of the full timed opacity emptiness (Theorem 7.4) and the intractability of timed opacity synthesis (Proposition 6.4). It remains open whether these results still apply to the more restrictive class of U-PTAs [55].

Full timed opacity synthesis. We leave full timed opacity synthesis as future work; while we could certainly reuse partially our algorithm, this is not entirely trivial, as we need to select only the parameter valuations for which the whole set of execution times is exactly the set of opaque times.

Applications. The translation of the STAC library required some non-trivial creativity: while the translation from programs to quantitative extensions of automata is orthogonal to our work, proposing automated translations of (possibly annotated) programs to timed automata dedicated to timing analysis is on our agenda.

Adding probabilities to our framework will be interesting, helping to quantify the execution times of "untimed" instructions in program with a finer grain than an interval; also note that some benchmarks make use of probabilities (notably STAC:18:v).

Finally, IMITATOR is a general model checker, not specifically aimed at solving the problem we address here. Notably, constraints managed by PPL contain all variables (clocks, timing parameters, and parameters encoding symbolic variables of programs), yielding an exponential complexity. Separating certain types of independent variables (typically parameters encoding symbolic variables of programs, and other variables) should increase efficiency.

ACKNOWLEDGMENTS

We would like to thank Sudipta Chattopadhyay for helpful suggestions, Nicolas Markey for an interesting discussion on the proof of Proposition 5.2, Jiaying Li for his help with preliminary model conversion, and an anonymous reviewer of ATVA 2019 for suggesting Remark 1.

This work is partially supported by the ANR national research program PACS (ANR-14-CE28-0002), by the ANR-NRF research program ProMiS (ANR-19-CE25-0015), and by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST.

REFERENCES

[1] Jeremy W. Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. 2008. Opacity generalised to transition systems. *International Journal of Information Security*, 7, 6, 421–435. DOI: 10.1007/s10207-008-0058-x.

[2] Rajeev Alur and David L. Dill. 1994. A theory of timed automata. *Theoretical Computer Science*, 126, 2, (April 1994), 183–235. DOI: 10.1016/0304-3975(94)90010-8.

[3] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. 1993. Parametric real-time reasoning. In *STOC* (May 16, 1993–May 18, 1993). S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors. ACM, San Diego, California, United States, 592–601. DOI: 10.1145/167088.167242.

[4] Étienne André and Jun Sun. 2019. Parametric timed model checking for guaranteeing timed opacity. In *ATVA* (Lecture Notes in Computer Science) (October 28, 2019–October 31, 2019). Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors. Volume 11781. Springer, Taipei, Taiwan, 115–130. DOI: 10.1007/978-3-030-31784-3_7.

[5] Volker Weispfenning. 1999. Mixed real-integer linear quantifier elimination. In *ISSAC* (July 29, 1999–July 31, 1999). Keith O. Geddes, Bruno Salvy, and Samuel S. Dooley, editors. Association for Computing Machinery, Vancouver, BC, Canada, 129–136. DOI: 10.1145/309831.309888.

[6] Paul C. Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO* (Lecture Notes in Computer Science) (August 18, 1996–August 22, 1996). Neal Koblitz, editor. Volume 1109. Springer, Santa Barbara, California, USA, 104–113. DOI: 10.1007/3-540-68697-5_9.

[7] Edward W. Felten and Michael A. Schneider. 2000. Timing attacks on Web privacy. In *CCS* (November 1, 2000–November 4, 2000). Dimitris Gritzalis, Sushil Jajodia, and Pierangela Samarati, editors. ACM, Athens, Greece, 25–32. DOI: 10.1145/352600.352606.

[8] Andrew Bortz and Dan Boneh. 2007. Exposing private information by timing Web applications. In *WWW* (May 8, 2007–May 12, 2007). Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors. ACM, Banff, Alberta, Canada, 621–628. DOI: 10.1145/1242572.1242656.

[9] Robert Kotcher, Yutong Pei, Pranjal Jumde, and Collin Jackson. 2013. Cross-origin pixel stealing: timing attacks using CSS filters. In *CCS* (November 4, 2013–November 8, 2013). Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors. ACM, Berlin, Germany, 1055–1062. DOI: 10.1145/2508859.2516712.

[10] Gilles Benattar, Franck Cassez, Didier Lime, and Olivier H. Roux. 2015. Control and synthesis of non-interferent timed systems. *International Journal of Control*, 88, 2, 217–236. DOI: 10.1080/00207179.2014.944356.

[11] Guillaume Gardey, John Mullins, and Olivier H. Roux. 2007. Non-interference control synthesis for security timed automata. *Electronic Notes in Theoretical Computer Science*, 180, 1, 35–53. DOI: 10.1016/j.entcs.2005.05.046.

[12] Roberto Barbuti, Nicoletta De Francesco, Antonella Santone, and Luca Tesei. 2002. A notion of non-interference for timed automata. *Fundamenta Informormaticae*, 51, 1-2, 1–11.

[13] Roberto Barbuti and Luca Tesei. 2003. A decidable notion of timed non-interference. *Fundamenta Informaticae*, 54, 2-3, 137–150.

[14] Étienne André and Aleksander Kryukov. 2020. Parametric non-interference in timed automata. In *ICECCS* (March 4, 2021–March 6, 2021). Yi Li and Alan Liew, editors. Singapore, 37–42. DOI: 10.1109/ICECCS51672.2020.00012.

[15] Franck Cassez. 2009. The dark side of timed opacity. In *ISA* (Lecture Notes in Computer Science) (June 25, 2009–June 27, 2009). Jong Hyuk Park, Hsiao-Hwa Chen, Mohammed Atiquzzaman, Changhoon Lee, Tai-Hoon Kim, and Sang-Soo Yeo, editors. Volume 5576. Springer, Seoul, Korea, 21–30. DOI: 10.1007/978-3-642-02617-1_3.

[16] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. 1999. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211, 1-2, 253–273. DOI: 10.1016/S0304-3975(97)00173-4.

[17] Ikhlass Ammar, Yamen El Touati, Moez Yeddes, and John Mullins. 2021. Bounded opacity for timed systems. *Journal of Information Security and Applications*, 61, (September 2021), 1–13. DOI: 10.1016/j.jisa.2021.102926.

[18] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: scalable verification of hybrid systems. In *CAV* (Lecture Notes in Computer Science) (July 14, 2011–July 20, 2011). Ganesh Gopalakrishnan and Shaz Qadeer, editors. Volume 6806. Springer, Snowbird, UT, USA, 379–395. DOI: 10.1007/978-3-642-22110-1_30.

[19] Flemming Nielson, Hanne Riis Nielson, and Panagiotis Vasilikos. 2017. Information flow for timed automata. In *Models, Algorithms, Logics and Tools* (Lecture Notes in Computer Science). Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna Ingólfsdóttir, Axel Legay, and Radu Mardare, editors. Volume 10460. Springer, 3–21. DOI: 10.1007/978-3-319-63121-9_1.

[20] Panagiotis Vasilikos, Flemming Nielson, and Hanne Riis Nielson. 2018. Secure information release in timed automata. In *POST* (Lecture Notes in Computer Science) (April 14, 2018–April 20, 2018). Lujo Bauer and Ralf Küsters, editors. Volume 10804. Springer, Thessaloniki, Greece, 28–52. DOI: 10.1007/978-3-319-89722-6_2.

[21] Christopher Gerking, David Schubert, and Eric Bodden. 2018. Model checking the information flow security of real-time systems. In *ESSoS* (Lecture Notes in Computer Science) (June 26, 2018–June 27, 2018). Mathias Payer, Awais Rashid, and Jose M. Such, editors. Volume 10953. Springer, Paris, France, 27–43. DOI: 10.1007/978-3-319-94496-8_3.

[22] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. 1997. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1, 1-2, 134–152. DOI: 10.1007/s100090050010.

[23] Lingtai Wang and Naijun Zhan. 2018. Decidability of the initial-state opacity of real-time automata. In *Symposium on Real-Time and Hybrid Systems - Essays Dedicated to Professor Chaochen Zhou on the Occasion of His 80th Birthday*. Lecture Notes in Computer Science. Volume 11180. Cliff B. Jones, Ji Wang, and Naijun Zhan, editors. Springer, 44–60. DOI: 10.1007/978-3-030-01461-2_3.

[24] Lingtai Wang, Naijun Zhan, and Jie An. 2018. The opacity of real-time automata. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37, 11, 2845–2856. DOI: 10.1109/TCAD.2018.2857363.

[25] Véronique Bruyère, Emmanuel Dall'Olio, and Jean-Francois Raskin. 2008. Durations and parametric model-checking in timed automata. *ACM Transactions on Computational Logic*, 9, 2, 12:1–12:23. DOI: 10.1145/1342991.1342996.

[26] Amnon Rosenmann. 2019. The timestamp of timed automata. In *FORMATS* (Lecture Notes in Computer Science) (August 27, 2019–August 29, 2019). Étienne André and Mariëlle Stoelinga, editors. Volume 11750. Springer, Amsterdam, The Netherlands, 181–198. DOI: 10.1007/978-3-030-29662-9_11.

[27] Véronique Bruyère and Jean-François Raskin. 2007. Real-time model-checking: parameters everywhere. *Logical Methods in Computer Science*, 3, 1:7, 1–30. DOI: 10.2168/LMCS-3(1:7)2007.

[28] Patricia Bouyer, Léo Henry, Samy Jaziri, Thierry Jéron, and Nicolas Markey. 2021. Diagnosing timed automata using timed markings. *International Journal on Software Tools for Technology Transfer*, 23, 2, 229–253. DOI: 10.1007/s10009-021-00606-2.

[29] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2020. Spectre attacks: exploiting speculative execution. *Communications of the ACM*, 63, 7, 93–101. DOI: 10.1145/3399742.

[30] Johan Agat. 2000. Transforming out timing leaks. In *POPL* (January 19, 2000–January 21, 2000). Boston, Massachusetts, USA, 40–53. DOI: 10.1145/325694.325702.

[31] David Molnar, Matt Piotrowski, David Schultz, and David A. Wagner. 2005. The program counter security model: automatic detection and removal of control-flow side channel attacks. In *ICISC* (December 1, 2005–December 2, 2005). Seoul, Korea, 156–168. DOI: 10.1007/11734727_14.

[32] Bart Coppens, Ingrid Verbauwhede, Koen De Bosschere, and Bjorn De Sutter. 2009. Practical mitigations for timing-based side-channel attacks on modern x86 processors. In *S&P* (May 17, 2009–May 20, 2009). Oakland, California, USA, 45–60. DOI: 10.1109/SP.2009.19.

[33] Chao Wang and Patrick Schaumont. 2017. Security by compilation: an automated approach to comprehensive side-channel resistance. *SIGLOG News*, 4, 2, 76–89. DOI: 10.1145/3090064.3090071.

[34] Meng Wu, Shengjian Guo, Patrick Schaumont, and Chao Wang. 2018. Eliminating timing side-channel leaks using program repair. In *ISSTA* (July 16, 2018–July 21, 2018). Amsterdam, The Netherlands, 15–26. DOI: 10.1145/3213846.3213851.

[35] Gilles Barthe, Tamara Rezk, and Martijn Warnier. 2006. Preventing timing leaks through transactional branching instructions. *Electronic Notes in Theoretical Computer Science*, 153, 2, 33–55. DOI: 10.1016/j.entcs.2005.10.031.

[36] Chungha Sung, Brandon Paulsen, and Chao Wang. 2018. CANAL: a cache timing analysis framework via LLVM transformation. In *ASE* (September 3, 2018–March 7, 2018). Montpellier, France, 904–907. DOI: 10.1145/3238147.3240485.

[37] Sudipta Chattopadhyay and Abhik Roychoudhury. 2011. Scalable and precise refinement of cache timing analysis via model checking. In *RTSS* (November 29, 2011–December 2, 2011). Vienna, Austria, 193–203. DOI: 10.1109/RTSS.2011.25.

[38] Imran Hafeez Abbasi, Faiq Khalid Lodhi, Awais Mehmood Kamboh, and Osman Hasan. 2016. Formal verification of gate-level multiple side channel parameters to detect hardware trojans. In *FTSCS* (Communications in Computer and Information Science) (November 14, 2016). Cyrille Artho and Peter Csaba Ölveczky, editors. Volume 694. Tokyo, Japan, 75–92. DOI: 10.1007/978-3-319-53946-1_5.

[39] Jun Zhang, Pengfei Gao, Fu Song, and Chao Wang. 2018. Scinfer: refinement-based verification of software countermeasures against side-channel attacks. In *CAV, Part II* (July 14, 2018–July 17, 2018). Oxford, UK, 157–177. DOI: 10.1007/978-3-319-96142-2_12.

[40] Louise A. Dennis, Marija Slavkovik, and Michael Fisher. 2016. "How did they know?" – Model-checking for analysis of information leakage in social networks. In *COIN@AAMAS* (Lecture Notes in Computer Science) (May 9, 2016). Stephen Cranefield, Samhar Mahmoud, Julian A. Padget, and Ana Paula Rocha, editors. Volume 10315. Springer, Singapore, 42–59. DOI: 10.1007/978-3-319-66595-5_3.

[41] Goran Doychev, Dominik Feld, Boris Köpf, Laurent Mauborgne, and Jan Reineke. 2013. CacheAudit: A tool for the static analysis of cache side channels. In *USENIX Security Symposium* (August 14, 2013–August 16, 2013). Washington, DC, USA, 431–446.

[42] Shengjian Guo, Meng Wu, and Chao Wang. 2018. Adversarial symbolic execution for detecting concurrency-related cache timing leaks. In *ESEC/SIGSOFT FSE* (November 4, 2018–November 9, 2018). Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu, editors. ACM, Lake Buena Vista, FL, USA, 377–388. DOI: 10.1145/3236024.3236028.

[43] Duc-Hiep Chu, Joxan Jaffar, and Rasool Maghareh. 2016. Precise cache timing analysis via symbolic execution. In *RTAS* (April 11, 2016–April 14, 2016). IEEE Computer Society, Vienna, Austria, 293–304. DOI: 10.1109/RTAS.2016.7461358.

[44] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. 2002. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53, 183–220. DOI: 10.1016/S1567-8326(02)00037-1.

[45] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. 1991. Temporal proof methodologies for real-time systems. In *POPL* (January 21, 1991–January 23, 1991). David S. Wise, editor. ACM Press, Orlando, Florida, USA, 353–366. DOI: 10.1145/99583.99629.

[46] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. 2015. Integer parameter synthesis for real-time systems. *IEEE Transactions on Software Engineering*, 41, 5, 445–461. DOI: 10.1109/TSE.2014.2357445.

[47] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. 2009. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20, 5, 819–836. DOI: 10.1142/S0129054109006905.

[48] Alexander Schrijver. 1999. *Theory of linear and integer programming. Wiley-Interscience series in discrete mathematics and optimization.* Wiley. ISBN: 978-0-471-98232-6.

[49] Étienne André. 2021. IMITATOR 3: synthesis of timing parameters beyond decidability. In *CAV* (Lecture Notes in Computer Science) (July 18, 2021–July 23, 2021). Rustan Leino and Alexandra Silva, editors. Volume 12759. Springer, virtual, 1–14. DOI: 10.1007/978-3-030-81685-8_26.

[50] Étienne André. 2019. What's decidable about parametric timed automata? *International Journal on Software Tools for Technology Transfer*, 21, 2, (April 2019), 203–219. DOI: 10.1007/s10009-017-0467-0.

[51] Joseph S. Miller. 2000. Decidability and complexity results for timed automata and semi-linear hybrid automata. In *HSCC* (Lecture Notes in Computer Science) (March 23, 2000–March 25, 2000). Nancy A. Lynch and Bruce H. Krogh, editors. Volume 1790. Springer, Pittsburgh, PA, USA, 296–309. DOI: 10.1007/3-540-46430-1_26.

[52] Laurent Doyen. 2007. Robust parametric reachability for timed automata. *Information Processing Letters*, 102, 5, 208–213. DOI: 10.1016/j.ipl.2006.11.018.

[53] Nikola Beneš, Peter Bezděk, Kim Gulstrand Larsen, and Jiří Srba. 2015. Language emptiness of continuous-time parametric timed automata. In *ICALP, Part II* (Lecture Notes in Computer Science) (July 6, 2015–July 10, 2015). Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi,

and Bettina Speckmann, editors. Volume 9135. Springer, Kyoto, Japan, (July 2015), 69–81. DOI: 10.1007/978-3-662-47666-6_6.

[54] Marvin L. Minsky. 1967. *Computation: Finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. ISBN: 0-13-165563-9.

[55] Laura Bozzelli and Salvatore La Torre. 2009. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design*, 35, 2, 121–151. DOI: 10.1007/s10703-009-0074-0.

[56] Étienne André and Didier Lime. 2017. Liveness in L/U-parametric timed automata. In *ACSD* (June 25, 2017–June 30, 2017). Alex Legay and Klaus Schneider, editors. IEEE, Zaragoza, Spain, 9–18. DOI: 10.1109/ACSD.2017.19.

[57] Étienne André, Didier Lime, and Mathias Ramparison. 2018. TCTL model checking lower/upper-bound parametric timed automata without invariants. In *FORMATS* (Lecture Notes in Computer Science) (September 4, 2018–September 6, 2018). David N. Jansen and Pavithra Prabhakar, editors. Volume 11022. Springer, Beijing, China, 1–17. DOI: 10.1007/978-3-030-00151-3_3.

[58] Étienne André, Didier Lime, and Nicolas Markey. 2020. Language preservation problems in parametric timed automata. *Logical Methods in Computer Science*, 16, (January 2020), 1, (January 2020). DOI: 10.23638/LMCS-16(1:5)2020.

[59] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. 2008. The Parma Polyhedra Library: toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72, 1–2, 3–21. DOI: 10.1016/j.scico.2007.08.001.

[60] Étienne André, Vincent Bloemen, Laure Petrucci, and Jaco van de Pol. 2019. Minimal-time synthesis for parametric timed automata. In *TACAS, Part II* (Lecture Notes in Computer Science) (April 6, 2019–April 11, 2019). Tomáš Vojnar and Lijun Zhang, editors. Volume 11428. Springer, Prague, Czech Repubic, 211–228. DOI: 10.1007/978-3-030-17465-1_12.

[61] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. 2011. Secure information flow by self-composition. *Mathematical Structures in Computer Science*, 21, 6, 1207–1252. DOI: 10.1017/S0960129511000193.

[62] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. 2009. Romeo: A parametric model-checker for Petri nets with stopwatches. In *TACAS* (Lecture Notes in Computer Science) (March 22, 2009–March 29, 2009). Stefan Kowalewski and Anna Philippou, editors. Volume 5505. Springer, York, United Kingdom, (March 2009), 54–57. DOI: 10.1007/978-3-642-00768-2_6.

[63] Mingsong Lv, Wang Yi, Nan Guan, and Ge Yu. 2010. Combining abstract interpretation with model checking for timing analysis of multicore software. In *RTSS* (November 30, 2010–December 3, 2010). IEEE Computer Society, San Diego, California, USA, 339–349. DOI: 10.1109/RTSS.2010.30.

[64] Daniel Bundala and Joël Ouaknine. 2014. Advances in parametric real-time reasoning. In *MFCS, Part I* (Lecture Notes in Computer Science) (August 25, 2014–August 29, 2014). Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors. Volume 8634. Springer, Budapest, Hungary, 123–134. DOI: 10.1007/978-3-662-44522-8.

[65] Stefan Göller and Mathieu Hilaire. 2021. Reachability in two-parametric timed automata with one parameter is EXPSPACE-complete. In *STACS* (LIPIcs) (March 16, 2021–March 19, 2021). Markus Bläser and Benjamin Monmege, editors. Volume 187. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Saarbrücken, Germany, 36:1–36:18. DOI: 10.4230/LIPIcs.STACS.2021.36.

A THE CODE OF THE JAVA EXAMPLE

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

//Coefficients are Disregarded
public class Category1_vulnerable {
    private static final int port = 8000;
    private static final int secret = 1234;
    private static final int n = 32;
    private static ServerSocket server;

    private static void checkSecret(int guess) throws InterruptedException {
        if (guess <= secret) {
            for (int i = 0; i < n; i++) {
                for (int t = 0; t < n; t++) {
                    Thread.sleep(1);
                }
            }
        } else {
            for (int i = 0; i < n; i++) {
                for (int t = 0; t < n; t++) {
                    Thread.sleep(2);
                }
            }
        }
    }

    private static void startServer() {
        try {
            server = new ServerSocket(port);
            System.out.println("Server Started Port: " + port);
            Socket client;
            PrintWriter out;
            BufferedReader in;
            String userInput;
            int guess;
            while (true) {
                client = server.accept();
                out = new PrintWriter(client.getOutputStream(), true);
                in = new BufferedReader(new InputStreamReader(client.getInputStream()));

                userInput = in.readLine();
                try {
                    guess = Integer.parseInt(userInput);
                    if(guess < 0) {
                        throw new IllegalArgumentException();
                    }
                    checkSecret(guess);
                    out.println("Process Complete");
                } catch (IllegalArgumentException | InterruptedException e) {
                    out.println("Unable to Process Input");
                }
                client.shutdownOutput();
                client.shutdownInput();
                client.close();
            }
        } catch (IOException e) {
            System.exit(-1);
        }
    }

    public static void main(String[] args) throws InterruptedException {
        startServer();
    }
}
```

Note that the two "`for`" loops featuring a `Thread.sleep(1)` (resp. 2) could be equivalently replaced with a simple `Thread.sleep(32*32)` (resp. `Thread.sleep(2*32*32)`) statement, but

(1) this is the way the program is presented in the DARPA library, and
(2) a (minor) difficulty may come from these loops instead of a simple `Thread.sleep(32*32)` statement.