ANR-NRF ProMiS

# Deliverable 1

## Preliminary model for information leakage

*Author: Sun Jun*

The Spectre vulnerability in modern processors has been reported in 2018. The key insight is that speculative execution in processors can be misused to access secrets speculatively. Subsequently even though the speculatively executed states are squashed, the secret may linger in micro-architectural data structures such as cache, and hence can be potentially accessed by an attacker via side channels. The Spectre vulnerability is merely one example of a family of vulnerabilities which could lead to the so-called side channel attacks. In general, side channel attacks utilize information which is leaked through certain side channels (e. g., time, energy, cache state and sound wave) in order to reveal system secrets. For instance, a timing side channel attack simply observes variations in how long it takes to perform certain operations, and determines the value of a certain secret (e. g., an encryption key) in the system. Such attacks involve analysis of timing measurements and have been demonstrated to be effective in attacking a range of systems. Timing side channel attacks are known to be challenging to model, detect and mitigate. This part of the ProMiS project thus aims to develop an approach which allows us to systematically model and analyze information leakage through the timing side channel. Timed Automaton (TA) [AD94] is a prevalent formalism for real-time system modeling and verification. With TA, real-time behaviors are captured by clocking constraints on the guards of system transitions and resetting clocks. Parametric Timed Automaton (PTA) [AHV93] extends TA with unknown constant parameters, e. g., by allowing parameters in the timing constraints. Existing research on PTA focuses on problems such as parametric model checking, which aims to synthesize valuations of the unknown parameters such that a desirable property is satisfied. In such settings, a system design is modeled in the form of a PTA. The parameters are constant values that are to be fixed (as design choices).

We propose to model systems that are potentially subject to timing side channel attacks in the form of PTA and formulate the problem of mitigating timing attacks as a language inclusion checking problem between PTA (a.k.a. a parameter synthesis problem). That is, we synthesize constraints on the parameters to make the model under such constraints satisfy the non-interference and non-deducibility property (which are formal definitions of the absence of timing side channel attacks). We show that the synthesis can be reduced to the parametric timed language inclusion checking between PTA.

To the best of our knowledge, this is the first work on modeling the problem of mitigating timing side channel attacks as a language inclusion checking problem between PTA. To demonstrate the feasibility of our modeling, we evaluated our approach using 14 programs from the DARPA Space/Time Analysis for Cybersecurity (STAC) benchmark. The results show that we are able to precise model the programs and demonstrate the corresponding timing attacks based on the models. Furthermore, for all of the programs in the experiments, we successfully derive meaningful constraints on the timing parameters such that timing attacks are mitigated. The details can be found in the attached report.

Document completed by Sun Jun and Étienne André on March 14, 2022.

## References

[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 Timed Language Inclusion Checking for Mitigating Timing Attacks

Yueling Zhang Singapore Management University, Singapore
Jun Sun Singapore Management University, Singapore
Étienne André Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
Sudipta Chattopadhyay Singapore University of Technology and Design, Singapore

**Abstract**—We propose a semi-algorithm for the language inclusion checking of Parameterized Timed Automaton (PTA). The algorithm is further optimized with zone abstraction and simulation reduction. Such an algorithm has important applications such as automatic mitigation of the timing-based side-channel attacks, i.e., by synthesizing specific valuations of unknown parameters which guarantees the absence of timing attacks. We evaluate our approach using 14 programs from the STAC benchmark, where PTA are systematically constructed from the programs based on the control flow graph and the under/over approximated execution time according to the micro-architecture. The results indicate that our method always terminates on these programs and generates useful results efficiently.

## 1 Introduction

Timed automata, introduced by Alur and Dill in [1], are as one of the most prevalent models to specify and analyze real-time systems. With timed automata, real-time behaviors are captured by bounds (clocking constraints) on the guards of system transitions and resetting clocks. However, the behaviors of a timed automaton are very sensitive to the values of these bounds, and it is rather challenging to find their correct values in practice. Therefore, it is interesting to reason parametrically by considering these bounds as unknown parameters (and subsequently synthesize their values based on the desirable properties). Timed automata with such unknown constant parameters are called Parameterized Timed Automaton (PTA) [2]. Given a desirable property, there are existing methods and tools to automatically synthesize the valuation of the parameters [3] which guarantees that the property is satisfied.

In this work, we study the language inclusion checking problem of PTA, which to the best of our knowledge has not been studied before. That is, given one PTA which models the implementation ($\mathcal{AP}$) and one which models the specification ($\mathcal{AS}$), the problem is to synthesize parameter valuations such that the language of $\mathcal{AP}$ is included in the language of $\mathcal{AS}$. In other words, the problem is to fix the valuations of the parameters such that the implementation satisfies the specification. This problem is partially inspired by the problem of mitigating timing side-channel attacks (hereafter timing attacks, which are a way of inferring system secrets through observing timed system behaviors [4]). In particular, two essential properties that formalize the notion of timing-attack-freeness, i.e., non-interference [5] and non-deducibility [6], can be reduced to the language inclusion checking of PTA. Synthesizing the parameter's valuation in the language inclusion checking problem of PTA thus potentially allows mitigating timing attacks systematically.

In particular, we propose a semi-algorithm for solving the language inclusion checking problem of PTA based on zone abstraction. We further improve the performance of the semi-algorithm with antichain-based simulation reduction. As one application of the semi-algorithm, we formulate the problem of mitigating timing attacks as a language inclusion checking problem between PTA (a.k.a. a parameter synthesis problem). That is, we synthesize constraints on the parameters to make the model under such constraints satisfy the non-interference and non-deducibility. We show that the synthesis can be reduced to the parametric timed language inclusion checking between PTA. Although our semi-algorithm is not guaranteed to always terminate, we show that it produces an over-approximation of all safe parameter valuations if it is interrupted at any time. Furthermore, the synthesized constraint is exact when it terminates.

To the best of our knowledge, this is the first work on language inclusion checking of PTA. It is also the first work to apply such an approach to the mitigation of timing attacks. Several existing works on mitigating timing attacks demand hardware modifications. Among these, SafeSpec [7] conceals the side effects of speculation in temporary structures, InvisiSpec [8] aims to make the transient loads invisible to the cache hierarchy, and DAWG [9] works by isolating cache behaviors in a protected domain. Concurrently, software-based approaches for mitigating timing attacks have been proposed. These approaches have considered the usage of program repair and symbolic model checking [10], introducing new programming language [11] or using source code obfuscation to provide an illusion of program execution at runtime [12]. In contrast to these approaches, we model programs in the form of PTA and guarantee that our synthesized design is free of timing attacks. The proposed approach is the first one to provide provable guarantees without additional language or hardware support.

Our approach has been implemented as a self-contained software toolkit named KALI. To demonstrate the applicability of KALI, we evaluated KALI using 14 programs from the DARPA Space/Time Analysis for Cybersecurity (STAC) benchmark. Specifically, we replace timing delays in these programs with unknown parameters and apply KALI to generate constraints on these parameters. The results show that KALI always terminates in our experiments within the given time limit. For all of the programs in the experiments, we successfully derive meaningful constraints on the timing parameters such that timing attacks are mitigated.

The remainder of the paper is organized as follows. We first formalize our problem in Section 2. In Section 3, we present the details of our approach. In Section 4, we present our implementation and evaluation of our method. We review related work in Section 5 and conclude in Section 6.

## 2 Models, Properties and Problem Definition

In this section, we define our problem based on a system model in the form of parametric timed automata (PTA [2]). In our *attacker model*, we assume the attacker has a complete knowledge of the system model, i. e., a white-box. The attacker's goal is to infer certain secrets only through observing high-level (timed) events.

### 2.1 Models

We assume a set $\mathbb{X} = \{x_1, \dots, x_H\}$ of *clocks*, i. e., real-valued variables evolving at the same rate. A clock valuation is a function $w : \mathbb{X} \to \mathbb{R}_{\geq 0}$. We write $\vec{0}$ to denote the clock valuation assigning 0 to all clocks. Given a constant $d \in \mathbb{R}_{\geq 0}$, $w + d$ denotes the clock valuation s.t. $(w + d)(x) = w(x)+d$ for all $x \in \mathbb{X}$. Intuitively, $w+d$ is the clock valuation after $d$ time units have elapsed. Given a set of clocks $R \subseteq \mathbb{X}$, we define the *reset* of a valuation $w$, denoted by $[w]_R$, as follows: $[w]_R(x) = 0$ if $x \in R$, and $[w]_R(x) = w(x)$ otherwise. That is, $[w]_R$ is the clock valuation after resetting to 0 all clocks in $R$.

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

**Definition 1** (PTA). *A PTA is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, \Xi)$, where $\Sigma$ is a finite set of synchronization events including a special $\tau$ event which represents a silent (unobserved) event; $L$ is a finite set of locations; $\ell_0 \in L$ is the initial location; $\mathbb{X}$ is a finite set of clocks; $\mathbb{P}$ is a finite set of parameters; $I$ is the invariant, assigning to every $\ell \in L$ a guard $I(\ell)$; and $\Xi$ is a finite set of edges (or transitions) $\xi = (\ell, g, e, R, \ell')$ where $\ell, \ell' \in L$ are the source and target locations, $e \in \Sigma$, $R \subseteq \mathbb{X}$ is a set of clocks to be reset, and $g$ is a guard.*

Given a parameter valuation $v$, we denote by $v(\mathcal{A})$ the non-parametric structure where all occurrences of a parameter $p_i$ have been replaced by $v(p_i)$. For instance, Fig. 1a shows a simple PTA (inspired by [13, Fig. 1b]) where the nodes are the locations, arrows represent edges which are labeled with events and transition guards (and optionally clocks to be reset during the transition), and $p_1$ and $p_2$ are parameters.

In this work, we focus on the finite prefix-closed timed language of PTA. Thus, an invariant $I$ at location $\ell$ can always be moved to the guard of every incoming and outgoing transition of $\ell$, without changing the semantics of the PTA. For example, Fig. 1b shows the PTA obtained by moving all of the invariants of the PTA shown in Fig. 1a to the transition guards. The state invariants are always moved to the transition guards in our approach.

**Definition 2** (Projection). *Let $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, \Xi)$ be a PTA and $\Pi \subseteq \Sigma$ be a set of events. The projection of $\mathcal{A}$ to $\Pi$, written as $\mathcal{A}{\upharpoonright}\Pi$, is a PTA $(\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, \Xi')$ such that for every edge $\xi = (\ell, g, e, R, \ell') \in \Xi$, $\xi$ is in $\Xi'$ if $e \in \Pi$. Otherwise, $\xi' = (\ell, g, \tau, R, \ell')$ is in $\Xi'$.*

**Definition 3** (Pruning). *Let $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, \Xi)$ be a PTA and $\Pi \subseteq \Sigma$ be a set of events. The* pruning *of $\Pi$ from $\mathcal{A}$, written as $\mathcal{A}/\Pi$, is a PTA $(\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, \Xi')$ such that for every edge $\xi = (\ell, g, e, R, \ell') \in \Xi$, $\xi$ is in $\Xi'$ if $e \notin \Pi$.*

Fig. 2a shows the result of $\mathcal{A}{\upharpoonright}\{l\}$ where $\mathcal{A}$ is shown in Fig. 1b, and Fig. 2b is $\mathcal{A}{\upharpoonright}\{h\}$. Pruning event $h$ from the PTA $\mathcal{A}$ shown in Fig. 1b, written as $\mathcal{A}/\{h\}$, removes locations $\ell_2$ and $\ell_3$, as shown in Fig. 3.

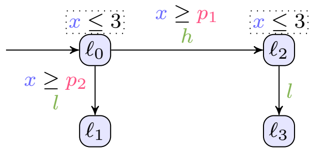
**Definition 4** (Parallel composition). *Let $\mathcal{A}_i = (\Sigma^i, L^i, \ell_0^i, \mathbb{X}^i, \mathbb{P}^i, I^i, \Xi^i)$ where $i \in \{0, 1\}$ be two PTAs such that all components are disjoint except $\Sigma^0$ and $\Sigma^1$, the* parallel composition *of $\mathcal{A}_0$ and $\mathcal{A}_1$, written as $\mathcal{A}_0 \parallel \mathcal{A}_1$, is a PTA $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, \Xi)$ such that $\Sigma = \Sigma^0 \cup \Sigma^1$; $L = L^0 \times L^1$; $\ell_0 = (\ell_0^0, \ell_0^1)$; $\mathbb{X} = \mathbb{X}^0 \cup \mathbb{X}^1$; $\mathbb{P} = \mathbb{P}^0 \cup \mathbb{P}^1$; $I$ is defined such that $I((\ell^0, \ell^1)) = I^0(\ell^0) \cap I^0(\ell^1)$; and $e$ is the smallest set which satisfies the following:*

- *if $(\ell_i, g_i, e, R_i, \ell_i') \in \Xi^0$ and $(\ell_j, g_j, e, R_j, \ell_j') \in \Xi^1$ and $e \in \Sigma^i \cap \Sigma^j$, $((\ell_i, \ell_j), g_i \cap g_j, e, R_i \cup R_j, (\ell_i', \ell_j')) \in \Xi'$;*
- *if $(\ell_i, g_i, e_i, R_i, \ell_i') \in \Xi^i$ and $e_i \notin \Sigma^i \cap \Sigma^{1-i}$, $((\ell_i, \ell_j), g_i, e_i, R_i, (\ell_i', \ell_j)) \in \Xi'$ for all $\ell_j \in L^{1-i}$.*

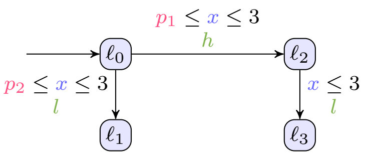
**Definition 5** (PTA Semantics). *Let $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, \Xi)$ be a PTA and a parameter valuation $v$, the semantics of $v(\mathcal{A})$ is given by the timed transition system $(S, s_0, \to)$, with*

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

We write $(\ell, w) \xrightarrow{(\xi,d)} (\ell', w')$ for a combination of a delay and discrete transition if $\exists w'' : (\ell, w) \overset{d}{\mapsto} (\ell, w'') \overset{\xi}{\mapsto} (\ell', w')$. The sequence of timed events $\pi = \langle (e_0, d_0), (e_1, d_1 + d_0), \cdots, (e_n, \sum_{i\in 0\cdots n} d_i)\rangle$ for all $i$ is a (timed) *trace* of $v(\mathcal{A})$.

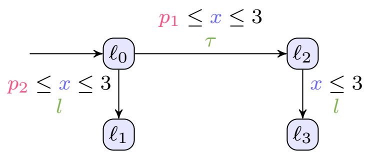


(a) A PTA example: $\mathcal{A}_{ex}$

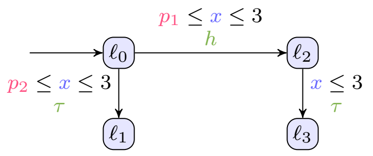


(b) Transformation

Figure 1: A PTA example: $\mathcal{A}_{ex}$



(a) $\mathcal{A}_{ex}\lceil\{l\}$



(b) $\mathcal{A}_{ex}\lceil\{h\}$

Figure 2: Projection examples

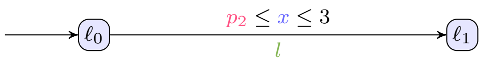


Figure 3: A PTA pruning example: $\mathcal{A}_{ex}/\{h\}$

We define the *language* of $v(\mathcal{A})$ to be the set of traces of $v(\mathcal{A})$, denoted as $\mathcal{L}(v(\mathcal{A}))$. Given a trace $\pi$ of $v(\mathcal{A})$, we can project it to $v(\mathcal{A}_0)$ by keeping only those timed events whose events are in $\Sigma_0$. In general, given a set of events $\Pi$ and a trace $\pi$, we write $\pi\lceil\Pi$ to be the sequence of timed events whose events are in $\Pi$. Furthermore, we write $head(\pi)$ to be the first event in $\pi$ and $tail(\pi)$ to be the remaining sequence of events in $\pi$.

A linear term over $\mathbb{X} \cup \mathbb{P}$ is of the form $\sum_{1\le i\le H} \alpha_i x_i + \sum_{1\le j\le M} \beta_j p_j + d$, 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) 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 $w$, $w(v(C))$ denotes the expression obtained by replacing each clock $x$ in $v(C)$ with $w(x)$. We say that $v$ *satisfies* $C$, denoted by $v \models C$, if the set of clock valuations satisfying $v(C)$ is nonempty. Given a parameter valuation $v$ and a clock valuation $w$, we denote by $w|v$ the valuation over $\mathbb{X}\cup\mathbb{P}$ such that for all clocks $x$, $w|v(x) = w(x)$ and for all parameters $p$, $w|v(p) = v(p)$. We use the notation $w|v \models C$ to indicate that $w(v(C))$ evaluates to true. We say that $C$ is *satisfiable* if $\exists w, v$ s.t. $w|v \models C$.

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, $w'|v \models C^{\nearrow}$ iff $\exists w : \mathbb{X} \to \mathbb{R}_+, \exists d \in \mathbb{R}_+$ s.t. $w|v \models C \wedge w' = w + d$.

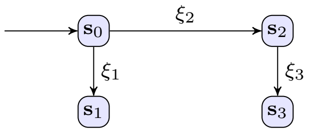


Figure 4: Parametric zone graph of Fig. 1

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 $R$, and keeping the other clocks unchanged. 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 [14]).

**Definition 6** (Symbolic semantics)**.** *Given a PTA $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, \Xi)$, the symbolic semantics of $\mathcal{A}$ is the labeled transition system called* parametric zone graph *$\mathcal{PZG} = (\Xi, \mathbf{S}, \mathbf{s}_0, \Rightarrow)$, with*

- *$\mathbf{S} = \{(\ell, C) \mid C \subseteq I(\ell)\}$, $\mathbf{s}_0 = (\ell_0, (\bigwedge_{1\le i\le H} x_i = 0)^{\nearrow} \wedge I(\ell_0))$, and*
- *$((\ell, C), \xi, (\ell', C')) \in \Rightarrow$ if $\xi = (\ell, g, e, R, \ell')$ and $C' = ([(C \wedge g)]_R \wedge I(\ell'))^{\nearrow} \wedge I(\ell')$ with $C'$ satisfiable.*

That is, in the parametric zone graph, nodes are symbolic states, and arcs are labeled by *edges* of the original PTA. A symbolic state is a pair $(\ell, C)$ where $\ell \in L$ is a location, and $C$ its associated constraint. This graph is (in general) *infinite* and, in contrast to the zone graph of timed automata, no finite abstraction can be built for properties of interest; this can be put in perspective with the fact that most problems are undecidable for PTAs [15].

**Example 1.** *Consider the PTA $\mathcal{A}$ in Fig. 1. The parametric zone graph of $\mathcal{A}$ is given in Fig. 4, where $\xi_1$ is the edge from $\ell_0$ to $\ell_1$ in Fig. 1, $\xi_2$ is the edge from $\ell_0$ to $\ell_2$, and $\xi_3$ is the edge from $\ell_2$ to $\ell_1$. In addition, the symbolic states are:*

$$\begin{aligned}
\mathbf{s}_0 &= (\ \ell_0\ ,\ 0 \le x \le 3 \wedge p_1 \ge 0 \wedge p_2 \ge 0\ ) \\
\mathbf{s}_1 &= (\ \ell_1\ ,\ x \ge p_2 \wedge 0 \le p_2 \le 3 \wedge p_1 \ge 0\ ) \\
\mathbf{s}_2 &= (\ \ell_2\ ,\ 3 \ge x \ge p_1 \wedge 0 \le p_1 \le 3 \wedge p_2 \ge 0\ ) \\
\mathbf{s}_3 &= (\ \ell_3\ ,\ x \ge p_1 \wedge 0 \le p_1 \le 3 \wedge p_2 \ge 0\ ).
\end{aligned}$$

## 2.2 Intuition and Problem Definition

In the following, we present two properties for capturing information leakage for timed systems. One is non-interference, and the other is non-deducibility.

Non-interference and non-deducibility properties are security properties that are widely studied [44], [45], [46]. Suppose two events $\mathbb{H}$ and $\mathbb{L}$ with high/low privacy, respectively. The intuitive meaning of non-interference is to guarantee that an external observer could never infer that $\mathbb{H}$ happens based only on observing $\mathbb{L}$. The intuitive meaning of non-deducibility is to guarantee that an external observer could never infer whether an event $\mathbb{H}$ has occurred based only on observing $\mathbb{L}$, and vice versa.

In the following, we assume a TA $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, I, \Xi)$. Let $\mathbb{L}$ be a set of *low* events and $\mathbb{H}$ be a set of *high* events such that $\tau \notin \mathbb{L}$ and $\tau \notin \mathbb{H}$ and $\mathbb{L} \cup \mathbb{H} \cup \{\tau\} = \Sigma$.

**Definition 7** (Non-interference [5]). *$\mathcal{A}$ satisfies (strong non-deterministic)* non-interference *if and only if $\mathcal{L}(\mathcal{A}/\mathbb{H}) = \mathcal{L}(\mathcal{A}\restriction\mathbb{L})$.*

**Example 2.** *Consider the PTA in Fig. 1. We assume $l \in \mathbb{L}$ and $h \in \mathbb{H}$. Let $v$ be a parameter valuation such that $v(p_1) = 1$ and $v(p_2) = 2$. $v(\mathcal{A})$ does not satisfy non-interference: intuitively, if $l$ occurs within time $[1, 2)$, then an external observer immediately knows that $h$ has occurred. Formally, $\mathcal{L}(\mathcal{A}/\mathbb{H})$ only allows the transition from $\ell_0$ to $\ell_1$ (at time $[2, 3]$) while $\mathcal{L}(\mathcal{A}\restriction\mathbb{L})$ also allows the transition from $\ell_2$ to $\ell_3$ (with the transition from $\ell_0$ to $\ell_2$ becoming silent—but still there). Let $v'$ be a parameter valuation such that $v'(p_1) = v'(p_2) = 2$. Then $v'(\mathcal{A})$ satisfies non-interference.*

It is trivial to show that $\mathcal{L}(\mathcal{A}/\mathbb{H}) \subseteq \mathcal{L}(\mathcal{A}\restriction\mathbb{L})$ and thus the problem of checking whether a timed automaton satisfies non-interference reduces to the problem of checking whether $\mathcal{L}(\mathcal{A}\restriction\mathbb{L}) \subseteq \mathcal{L}(\mathcal{A}/\mathbb{H})$.

**Definition 8** (Non-deducibility [6]). *$\mathcal{A}$ satisfies non-deducibility if and only if: $\forall tr \in \mathcal{L}(\mathcal{A}\restriction\mathbb{L}), \forall tr' \in \mathcal{L}(\mathcal{A}\restriction\mathbb{H}), \exists \pi \in \mathcal{L}(\mathcal{A}) : tr = \pi\restriction\mathbb{L} \wedge tr' = \pi\restriction\mathbb{H}$.*

**Example 3.** *We assume $l \in \mathbb{L}$ and $h \in \mathbb{H}$. Fig. 2 shows $\mathcal{A}\restriction\mathbb{L}$ and $\mathcal{A}\restriction\mathbb{H}$ given the PTA $\mathcal{A}$ shown in Fig. 1b. Let $v$ be a parameter valuation such that $v(p_1) = 2$ and $v(p_2) = 1$. $v(\mathcal{A})$ does not satisfy non-deducibility: intuitively, if $l$ occurs within time $[1, 2]$, an external observer immediately deduces that $h$ has not occurred. Let $v'$ be a parameter valuation such that $v'(p_1) = 1$ and $v'(p_2) = 1$. Then $v'(\mathcal{A})$ satisfies non-deducibility.*

Our problems are thus defined as follows.

**Synthesis problem:**
INPUT: a PTA $\mathcal{A}$
PROBLEM: Synthesize a set $\phi$ of valuations such that for all $v \models \phi$, $v(\mathcal{A})$ satisfies non-deducibility (resp. non-interference)

Hereafter, we say that $\mathcal{A}_\phi$ satisfies non-deducibility (resp. non-interference) if and only if $v(\mathcal{A})$ satisfies non-deducibility (resp. non-interference) for all $v \models \phi$.

## 3 Our Approach

In this section, we present our approach step-by-step. Without loss of generality, we fix a PTA $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, \Xi)$, a set of low events $\mathbb{L}$ and a set of high events $\mathbb{H}$. Note that a TA can be viewed as a special PTA whose $\mathbb{P}$ is an empty set and thus our approach can be readily applied to check whether a TA satisfies non-interference or non-deducibility.

### 3.1 Reduction to language inclusion checking

We first show that the synthesis problem for non-interference and non-deducibility can be reduced to the problem of language inclusion checking for PTA.

**Theorem 1.** *Let $\phi$ be a constraint over $\mathbb{P}$. $\mathcal{A}_\phi$ satisfies non-interference if and only if $\mathcal{L}(v(\mathcal{A})\restriction\mathbb{L}) \subseteq \mathcal{L}(v(\mathcal{A})/\mathbb{H})$ for all $v \models \phi$.*

*Proof.* By Definition 7, for any TA $v(\mathcal{A})$, $v(\mathcal{A})$ satisfies non-interference if and only if $\mathcal{L}(v(\mathcal{A})/\mathbb{H}) = \mathcal{L}(v(\mathcal{A})\restriction\mathbb{L})$. Since $\mathcal{L}(v(\mathcal{A})/\mathbb{H}) \subseteq \mathcal{L}(v(\mathcal{A})\restriction\mathbb{L})$ is always true (by Definitions 2 and 3), the theorem holds. □

**Lemma 1.** *Let $\phi$ be a constraint on $\mathbb{P}$. $\mathcal{A}_\phi$ satisfies non-deducibility if $\mathcal{L}((v(\mathcal{A})\restriction\mathbb{L}) \parallel (v(\mathcal{A})\restriction\mathbb{H})) \subseteq \mathcal{L}(v(\mathcal{A}))$ for all $v \models \phi$.*

*Proof.* Let $tr$ be an arbitrary trace of $\mathcal{A}\restriction\mathbb{L}$ and $tr'$ be an arbitrary trace of $\mathcal{A}\restriction\mathbb{H}$. Let $head(tr)$ be $(e, d)$ and $head(tr')$ be $(e', d')$. We construct $tr \parallel tr'$, which is a set of traces, as shown in Formula (6) (where $\frown$ is sequence concatenation): By Definition 4, any trace $\pi$ in $tr \parallel tr'$ must be in $\mathcal{L}((v(\mathcal{A})\restriction\mathbb{L}) \parallel (v(\mathcal{A})\restriction\mathbb{H}))$. Given $\mathcal{L}((v(\mathcal{A})\restriction\mathbb{L}) \parallel (v(\mathcal{A})\restriction\mathbb{H})) \subseteq \mathcal{L}(v(\mathcal{A}))$, $\pi \in \mathcal{L}(v(\mathcal{A}))$. Since, $tr = \pi\restriction\mathbb{L} \wedge tr' = \pi\restriction\mathbb{H}$, by Definition 8, $\mathcal{A}$ satisfies non-deducibility. □

**Lemma 2.** *Let $\phi$ be a constraint over $\mathbb{P}$. For all $v \models \phi$, if $\mathcal{A}_\phi$ satisfies non-deducibility, then $\mathcal{L}((v(\mathcal{A})\restriction\mathbb{L}) \parallel (v(\mathcal{A})\restriction\mathbb{H})) \subseteq \mathcal{L}(v(\mathcal{A}))$.*

*Proof.* Let $\pi$ be an arbitrary trace in $\mathcal{L}((v(\mathcal{A})\restriction\mathbb{L}) \parallel (v(\mathcal{A})\restriction\mathbb{H}))$. There must exist $tr$ in $\mathcal{A}\restriction\mathbb{L}$ and some $tr'$ in $\mathcal{A}\restriction\mathbb{H}$ such that $\pi \in tr \parallel tr'$ (for instance, $tr = \pi\restriction\mathbb{L}$ and $tr' = \pi\restriction\mathbb{H}$).

Let $tr$ be an arbitrary trace of $\mathcal{A}\restriction\mathbb{L}$ and $tr'$ be an arbitrary trace of $\mathcal{A}\restriction\mathbb{H}$. We have $tr \parallel tr' \subseteq \mathcal{L}((v(\mathcal{A})\restriction\mathbb{L}) \parallel (v(\mathcal{A})\restriction\mathbb{H}))$. By Definition 8, if $\mathcal{A}_\phi$ satisfies non-deducibility, there exists $\pi \in \mathcal{L}(\mathcal{A})$ such that $tr = \pi\restriction\mathbb{L} \wedge tr' = \pi\restriction\mathbb{H}$. It remains to show that if $\pi \in \mathcal{L}(\mathcal{A})$, $\pi\restriction\mathbb{L} \parallel \pi\restriction\mathbb{H} \in \mathcal{L}(\mathcal{A})$, which is true since $\mathbb{L}$ and $\mathbb{H}$ are disjoint. □

**Theorem 2.** *Let $\phi$ be a constraint over $\mathbb{P}$. $\mathcal{A}_\phi$ satisfies non-deducibility if and only if $\mathcal{L}((v(\mathcal{A})\restriction\mathbb{L}) \parallel (v(\mathcal{A})\restriction\mathbb{H})) \subseteq \mathcal{L}(v(\mathcal{A}))$ for all $v \models \phi$.*

*Proof.* By Lemmas 1 and 2. □

The above theorems reduce the synthesis problem to the problem of language inclusion checking between two PTAs, which we define as follows.

$$tr \parallel tr' = \begin{cases} \{\langle (e,d)\rangle \frown \pi \mid \pi \in \mathit{tail}(tr) \parallel tr'\} & \text{if } d < d' \\ \{\langle (e',d')\rangle \frown \pi \mid \pi \in tr \parallel \mathit{tail}(tr')\} & \text{if } d' > d \\ \{\langle (e,d),(e',d')\rangle \frown \pi \mid \pi \in \mathit{tail}(tr) \parallel \mathit{tail}(tr')\} \cup & \\ \quad \{\langle (e',d'),(e,d)\rangle \frown \pi \mid \pi \in \mathit{tail}(tr) \parallel \mathit{tail}(tr')\} & \text{otherwise} \end{cases} \tag{6}$$

**Language Inclusion Checking problem:**
INPUT: Two PTAs $\mathcal{A}^x = (\Sigma^x, L^x, \ell_0^x, \mathbb{X}^x, \mathbb{P}^x, I^x, \Xi^x)$ where $x \in \{P, S\}$ such that $L^P$ and $L^S$ as well as $\mathbb{X}^P$ and $\mathbb{X}^S$ are disjoint
PROBLEM: Synthesize a set $\phi$ of valuations (over $\mathbb{P}^P \cup \mathbb{P}^S$) such that, for all $v \models \phi$, $\mathcal{L}(v(\mathcal{A}^P)) \subseteq \mathcal{L}(v(\mathcal{A}^S))$

For instance, to solve the synthesis problem for non-deducibility given the PTA shown in Fig. 1, we solve the language inclusion problem from the parallel composition of Fig. 2a and Fig. 2a to Fig. 1.

In general, the solution to the language inclusion checking problem for PTA cannot be computed since it is known that the simpler language inclusion problem for TA is undecidable [1]. In [16], Wang *et al.* proposed a semi-algorithm for the language inclusion problem of TA, which often terminates in practice. Inspired by their work, we propose in the following a semi-algorithm to solve the language inclusion problem of **Parametric**-TA for solving our problem defined in Section 2. Moreover, we synthesize the valuation of the parameters in the language inclusion checking of PTA to ensure that non-interference and non-deducibility are satisfied. It can be seen as a "best-effort" procedure, which is not guaranteed to terminate but, if it does, its result is correct.

### 3.2 Language inclusion checking

Let $\mathcal{AP} = (\Sigma^P, L^P, \ell_0^P, \mathbb{X}^P, \mathbb{P}^P, I^P, \Xi^P)$ and $\mathcal{AS} = (\Sigma^S, L^S, \ell_0^S, \mathbb{X}^S, \mathbb{P}^S, I^S, \Xi^S)$ be two PTAs such that $L^P$ and $L^S$ as well as $\mathbb{X}^P$ and $\mathbb{X}^S$ are all disjoint. The language inclusion checking is to decide whether the language of $\mathcal{AP}$ is a subset of that of $\mathcal{AS}$. It is known that the problem can be converted to a reachability problem on the synchronous product of $\mathcal{AP}$ and the determinization of $\mathcal{AS}$ [47]. Therefore, the first step is to determinize $\mathcal{AS}$, and the second step is to compute the synchronous product of $\mathcal{AP}$ and the determinized $\mathcal{AS}$ using zone graph-based abstraction.

#### *3.2.1 Unfolding specification*

Conceptually, the first step is to unfold $\mathcal{AS}$ (adopted from the approach in [17]), although the unfolding is obviously done only on-the-fly in the actual implementation. The unfolding of $\mathcal{AS}$ is an infinite timed tree, which can be viewed as a PTA $\mathcal{AS}_\infty$ with infinitely many locations. $\mathcal{AS}_\infty$ has the input-determinacy property [17], with which we can determinize it when we construct the synchronous product of $\mathcal{AP}$ and $\mathcal{AS}_\infty$. In the following, we first show how the unfolding works using an example, and then define it.

Consider the PTA in the left of Fig. 5 (essentially a parameterization of [16, Fig. 1c]). The beginning of the unfolding is given on the right. Let $\langle z_0, z_1, z_2, \cdots \rangle$ be an infinite sequence of fresh clocks. At each level of the tree,

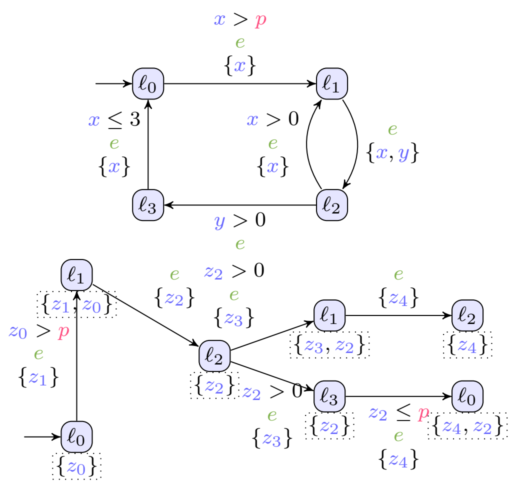


Figure 5: A PTA (left) and its unfolding (right)

a new clock from the sequence is introduced to replace the ordinary clocks. For instance, a fresh clock $z_0$ is introduced initially. Since clocks $x$ and $y$ start simultaneously as $z_0$, we can use $z_0$ to replace $x$ in the transition guard from $\ell_0$ to $\ell_1$. Similarly, a fresh clock $z_1$ is introduced afterward at level 1 of the tree, e.g., when $\ell_1$ is reached for the first time in Fig. 5. Since $x$ is reset and restarts at the same time as $z_1$, $z_1$ is used to replace $x$ in transition guards from $\ell_1$. This process continues and it is straightforward to see that the tree would contain only the freshly introduced clocks. To facilitate reduction later, each node in $\mathcal{AS}_\infty$ is associated with a set of active clocks. A clock is *active* at a node in $\mathcal{AS}_\infty$ if and only if it is a part of a constraint labeling any subsequent transitions or nodes. For instance, $z_0$ is active at the left-most node in Fig. 5 and it is no longer active at the third node from the left.

In the following, we define the unfolding of $\mathcal{AS}$ formally. Let $Z = \langle z_0, z_1, \cdots \rangle$ be an infinite sequence of clocks. The unfolding $\mathcal{AS}$ is an infinite timed tree, which can be viewed as a PTA $\mathcal{AS}_\infty = (\Sigma_\infty, L_\infty, \ell_0^\infty, \mathbb{X}_\infty, \mathbb{P}_\infty, \Xi_\infty)$ with infinitely many locations. Furthermore, we assume that $\mathcal{AS}_\infty$ is associated with a function *level* such that *level*$(n)$ is the level of node $n$ in the tree for all $n \in L_\infty$. A node $n$ in $L_\infty$ is in the form of $(\ell, A)$ where $\ell \in L^S$ and $A \subseteq Z$ is a set of active clocks. Given any node $n$, we define a function $f_n : \mathbb{X}^S \mapsto Z$ which maps ordinary clocks in $\mathbb{X}^S$ to active clocks in $Z$. In an abuse of notations, given a clock constraint $\delta$ on $\mathbb{X}^S$, we write $f_n(\delta)$ to denote the clock constraint obtained by replacing clocks in $\mathbb{X}^S$ with those in $Z$ according to $f_n$. Given any node $n = (\ell, A)$, we define $A$ to be $\{f_n(c) \mid c \in \mathbb{X}^S\}$. The initial state $\ell_0^\infty$ and transition relation $T_\infty$ are defined as follows.

- There is a level-0 node $n = (\ell_0^S, \{z_0\})$ in $L_\infty$ with *level*$(n) = 0$, $f_n(c) = z_0$ for all $c \in \mathbb{X}^S$.

- For each node $n = (\ell, A)$ at level $i$ and for each transition $(\ell, \delta, e, X, \ell') \in \Xi^S$, we add a node $n' = (\ell', A')$ in $L_\infty$ at level $i+1$ such that $f_{n'}(c) = f_n(c)$ if $c \in \mathbb{X}^S \setminus X$, $f_{n'}(c) = z_{i+1}$ if $c \in X$; $level(n') = i+1$. We add a transition $(n, f_n(\delta), e, \{z_{i+1}\}, n')$ to $\Xi_\infty$.

Note that transitions at the same level have the same newly introduced clocks. Given a node $n = (\ell, A)$ in the tree, observe that not every clock $x$ in $A$ is active as the clock may never be used to guard any transition from $q$. Hereafter, we assume that inactive clocks are always removed.

### 3.2.2 *Zone abstraction*

To solve the language inclusion problem between $\mathcal{AP}$ and $\mathcal{AS}_\infty$, we build a parametric zone graph of the synchronous product of $\mathcal{AP}$ and the determinization of $\mathcal{AS}_\infty$. This graph is $\mathcal{PZG} = (\Xi^P, \mathbf{S}, \mathrm{s}_0, \Rightarrow)$. Note that the first component is the set of edges of $\mathcal{AP}$, i. e., we label edges of the product with transitions of $\mathcal{AP}$. A state in $\mathbf{S}$ is a configuration of the form $(\ell_P, J_S, C)$ where $\ell_P \in L^P$, $J_S$ is a set of nodes in $L_\infty$, and $C$ is a parametric zone. Recall that a state of $\mathcal{AS}_\infty$ is of the form $(\ell_S, A)$, where $A$ is a set of active clocks. Given a set of states $J_S$ of $\mathcal{AS}_\infty$, we write $Act(J_S)$ to denote the set of all active clocks, i. e., $\{x \mid \exists(\ell_S, A) \in J_S \wedge x \in A\}$. $C$ constrains all clocks in $Act(J_S)$ as well as the parameters.

The initial state $\mathrm{s}_0$ of the parametric zone graph is: $(\ell_0^P, \{\ell_0^\infty\}, (Act(\ell_0^\infty) = 0 \wedge \mathbb{X}^P = 0)^\uparrow)$. Next, we define $\Rightarrow$ by showing how to generate successors of a given abstract configuration $(\ell_P, J_S, \delta)$. Let $\Xi_\infty(e, J_S)$ be the set of transitions in $\Xi_\infty$ which start with a state in $J_S$ and are labeled with event $e$. The guard conditions of transitions in $\Xi_\infty(e, J_S)$ may not be mutually exclusive. We define a set of constraints $Cons(e, J_S)$ such that each element in $Cons(e, J_S)$ is a constraint which conjuncts, for each transition in $\Xi_\infty(e, J_S)$, either the transition guard or its negation. Notice that elements in $Cons(e, J_S)$ are by definition mutually exclusive. Given $(\ell_P, J_S, \delta)$ and an outgoing transition $(\ell_P, g_p, e, X_P, \ell'_P)$ from $\ell_P$ in $\mathcal{AP}$, for each $g \in Cons(e, J_S)$ we generate a successor $(\ell'_P, J'_S, \delta')$ as follows.

- For any state $(\ell_S, A) \in J_S$ and any transition $((\ell_S, A), g_S, e, Y, (\ell'_S, A')) \in \Xi_\infty$, if $\delta \wedge g_P \wedge g \wedge g_S$ is not false, then $(\ell'_S, A') \in J'_S$.
- All states in $J_S$ are at the same level in the tree $\mathcal{AS}_\infty$ and thus all transitions in $\Xi_\infty(e, J_S)$ have the same freshly introduced clock. Let $y$ be that clock and $\delta' = ([y \cup X_P \mapsto 0](\delta \wedge g \wedge g_P))^\uparrow$ if $J_S \neq \emptyset$ otherwise $\delta' = [y \cup X_P \mapsto 0](\delta \wedge g \wedge g_P)$.

Intuitively, given any state $(\ell_P, J_S, \delta)$ in the product, $J_S$ is the set of nodes in $\mathcal{AS}$ which can be reached via the same timed word for reaching $\ell_P$ in $\mathcal{AP}$ (and $\delta$ is the constraint which must be satisfied). If $J_S$ is $\emptyset$, we have a timed word which is in the language of $\mathcal{AP}$ but not that of $\mathcal{AS}$. The following theorem establishes that the language inclusion problem is reduced to a reachability problem in $\mathcal{PZG}$. Its proof follows the proof of [16, Theorem 1].

**Theorem 3.** *$\mathcal{L}(\mathcal{AP}) \subseteq \mathcal{L}(\mathcal{AS})$ if and only if there is no reachable state of the form $(\ell_P, \emptyset, \delta)$ in $\mathcal{PZG}$.*

With the above, our goal is then to synthesize parameter valuations so that such target states are unreachable in the product, i. e., by making $\delta$ unsatisfiable.

### 3.3 Simulation reduction

We have reduced the parametric language inclusion checking problem to a reachability problem in $\mathcal{PZG}$. Next, we establish a simulation relationship between states of $\mathcal{PZG}$ so that we can apply simulation reduction in solving the problem.

**Definition 9** (simulation). *Let $(S, s_0, \Sigma, \rightarrow)$ be a labeled transition system. Let $F \subseteq S$ be a set of target states. A state $s_0 \in S$ is* simulated *by $s_1 \in S$ with respect to $F$ if $s_0 \in F$ implies $s_1 \in F$; and for all $e \in \Sigma$, $(s_0, e, s'_0) \in \rightarrow$ implies there exists $(s_1, e, s'_1) \in \rightarrow$ such that $s'_0$ is simulated by $s'_1$ with respect to $F$.*

In order to check whether a state in $F$ is reachable, if we know that $s_0$ is simulated by $s_1$, $s_0$ can be skipped during system exploration if $s_1$ has been explored already. Intuitively, this is because if a state in $F$ is reachable from $s_0$, it must be reachable from $s_1$. This is known as simulation reduction.

To compare two states $(\ell_P, J_S, \delta)$ and $(\ell'_P, J'_S, \delta')$ of $\mathcal{PZG}$, one problem is that $J_S$ and $J'_S$ as well as $\delta$ and $\delta'$ may have different clocks. Since the names of the clocks do not matter semantically, we define clock renaming functions to solve the problem. A renaming function from a set of clocks $X$ to $Y$ is a bijective function $\theta : X \rightarrow Y$ which maps every clock in $X$ to one in $Y$. We further write $\theta(A) \subseteq B$ where $A$ is a set of clocks to mean that $A$ is a subset of $B$ after clock renaming; and we write $\theta(J_S) \subseteq J'_S$ to mean that for every $(\ell, A)$ in $J_S$, there exists $(\ell', A')$ in $J'_S$ such that $\ell = \ell'$ and $\theta(J_S) = J'_S$.

The following lemma is inspired by the Anti-Chain algorithm [18].

**Lemma 3.** *Let $(\ell_P, J_S, \delta)$ and $(\ell_P, J'_S, \delta')$ be states in $\mathcal{PZG}$. Let $F = \{(\ell, \emptyset, \delta_0)\}$ be the set of target states. $(\ell_P, J'_S, \delta')$ simulates $(\ell_P, J_S, \delta)$ w.r.t. $F$ if there is a renaming function $\theta : Act(J_S) \rightarrow Act(J'_S)$ s.t. $\theta(J_S) \supseteq J'_S$ and $\theta(\delta) \implies \delta'$.*

*Proof.* The proof follows the proof of [16, Lemma 2]. □

### 3.4 Algorithm

In the following, we present our semi-algorithm for parametric language inclusion checking. Let $\mathcal{PZG}$ be the parametric zone graph of the product as explained above. Algorithm 1 constructs $\mathcal{PZG}$ on-the-fly while performing simulation reduction. It maintains three data structures. One is a set *Working* which stores states which are yet to be explored. Another is a set *Done* which contains states which have already been explored. The last one is $\phi$, which is the constraint over the parameters to be returned. Initially, *Working* is set to be $\{\mathrm{s}_0\}$ and *Done* is empty. During

**Algorithm 1:** Parametric language inclusion synthesis

```
   input : $\mathcal{PZG} = (\Xi^P, \mathbf{S}, \mathbf{s}_0, \Rightarrow)$
1  $Working \leftarrow \mathbf{s}_0$;
2  $Done \leftarrow \emptyset$;
3  $\phi \leftarrow true$;
4  while $Working \neq \emptyset$ or times out do
5      remove $(\ell_P, J_S, \delta)$ from $Working$
6      $Done \leftarrow Done \cup \{(\ell_P, J_S, \delta)\}$
7      forall $ps \in Done$ s.t. $(\ell_P, J_S, \delta)$ simulates $ps$ do
           $Done \leftarrow Done \setminus \{ps\}$ ;
8      forall $(l'_P, J'_S, \delta')$ such that
       $(\ell_P, J_S, \delta) \Rightarrow (l'_P, J'_S, \delta')$ do
9          if $J'_S = \emptyset$ then
10             $\phi \leftarrow \phi \wedge \neg(\delta'{\downarrow_{\mathbb{P}}})$ ;    /* Prevent this
               word */
11         if $\nexists ps \in Done$ s.t. $(\ell'_P, J'_S, \delta')$ is simulated by
           $ps$ then
12             $Working \leftarrow Working \cup \{(\ell'_P, J'_S, \delta')\}$
13 return $\phi$;
```

the loop from line 4 to line 8, a state is removed from *Working* and added to *Done* each time. To keep *Done* small, whenever a state $s$ is added into *Done*, all states in *Done* which are simulated by $s$ are removed.

We generate successors of $s$ at line 8. For each successor, if it is a target state, we obtain the constraint on the parameters at line 10 and its negation is conjuncted with $\phi$. Intuitively, $\delta'$ is the constraint that must be satisfied so that the transition $(\ell_P, J_S, \delta) \Rightarrow (l'_P, J'_S, \delta')$ is feasible. In order to make it infeasible (by constraining the parameters), we project the constraint onto the parameters $\delta'{\downarrow_{\mathbb{P}}}$ to obtain a constraint over the parameters which must be satisfied so that the transition is feasible. Its negation is then conjuncted with $\phi$. That is, if its negation is satisfied, this transition becomes infeasible and thus it prevents violation of language inclusion.

If $s$ is simulated by a state in *Done*, it is ignored. Otherwise, it is added into *Working* so that it will be explored later. Lastly, we return the constraint $\phi$ at line 13 after exploring all states. We remark that *Done* is an *AntiChain* [18] as any pair of states in *Done* is incomparable. The following theorem states that the semi-algorithm always produces correct results

**Theorem 4.** *If Algorithm 1 terminates and returns $\phi$, $\forall v \models \phi$. $\mathcal{L}(v(\mathcal{AP})) \subseteq \mathcal{L}(v(\mathcal{AS}))$.*

In general, Algorithm 1 does not always terminate. If it times out after a while, $\phi$ is guaranteed to be an *over-approximation* of all safe parameter valuations.

**Corollary 1.** *If Algorithm 1 times out and returns $\phi$, $\forall v \not\models \phi$. $\mathcal{L}(v(\mathcal{AP})) \not\subseteq \mathcal{L}(v(\mathcal{AS}))$.*

Based on Theorems 1 and 2, the above algorithm can be applied to solve the synthesis problem for non-interference and non-deducibility. That is, given a PTA model $\mathcal{A}$, we check $\mathcal{L}(v(\mathcal{A}){\restriction}\mathbb{L}) \subseteq \mathcal{L}(v(\mathcal{A})/\mathbb{H})$ using the algorithm and returns the constraint for non-interference; and we check $\mathcal{L}((v(\mathcal{A}){\restriction}\mathbb{L}) \parallel (v(\mathcal{A}){\restriction}\mathbb{H})) \subseteq \mathcal{L}(v(\mathcal{A}))$ using the algorithm for non-deducibility.

**Example 4.** *For example, to solve the synthesis problem for $\mathcal{A}_{ex}$ shown in Fig. 1 for non-interference, we check whether the language of $(\mathcal{A}_{ex}{\restriction}\{l\})$ shown in Fig. 2(a) is included in that of $\mathcal{L}(v(\mathcal{A})/\mathbb{H})$. The constructed $\mathcal{PZG}$ is shown in Fig. 6, where $\delta$ is $x = z_0 \wedge x \geq p_1 \wedge x \leq 3 \wedge (z_0 < p_2 \vee z_0 > 3)$, which simplifies to $p_1 \leq z_0 < p_2$. To make sure this constraint is unsatisfiable, we have $p_1 \geq p_2$.*

## 4 Evaluation

We implement a tool named Kali based on the approach. We use Z3 [19] to solve Linear Real Arithmetic constraints and Fourier-Motzkin [20] to eliminate the unrelated variables. To evaluate the relevance of Kali, we conduct a series of experiments using a PC with Intel® Core™ i5-9500 CPU at 3.10 GHz and 8.0 GiB RAM, running Linux Ubuntu 18.04.4 LTS. The tool and the models are available online at [1].

### 4.1 Modeling programs in PTAs

We apply Kali to the DARPA STAC (Space/Time Analysis for Cybersecurity) programs [2]. These programs are being released publicly to facilitate researchers to develop methods and tools for identifying STAC vulnerabilities in the programs. We systematically transfer the programs into PTA (ignoring complications due to hardware architecture such as caching for simplicity). The readers are referred to the rich literature documented in, for instance, [21]. In this work, we instead use the following simplistic assumption from [22] on the execution time of a program statement and focus on mitigating the timing attack problem. We assume that the execution time of a program statement other than *Thread.sleep(n)* is within a range $[0, \epsilon]$ where $\epsilon$ is a small integer constant (in milliseconds), whereas the execution time of statement *Thread.sleep(n)* is within a range $[n, n + \epsilon]$.

We use identifiers of the form `STAC:1` where 1 denotes the identifier in the library. There are both timing-related and space-related vulnerabilities in the STAC program. We focus on the timing related programs except those that deal with complex computations such as operations on matrices (`STAC:16:v`) or probabilities (`STAC:18:v`). We translated these programs to PTAs with the help of the control flow graphs. The loops in the programs are unrolled based on the loop guards.

**Example 5.** *We encode `STAC:1` as follows. In this program, a user guesses the value of a secret variable, the variable $x$ stands for the guessed value, and the variable secret stands for the secret value. There are two versions of `STAC:1`, one vulnerable*

1. https://github.com/z971586668/ptata/tree/master
2. https://github.com/Apogee-Research/STAC/

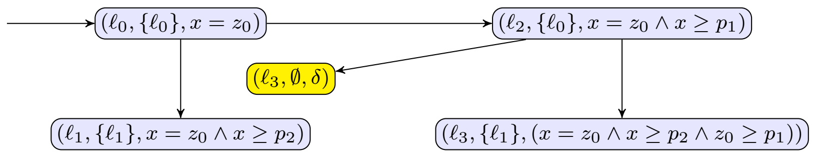

$(\ell_0, \{\ell_0\}, x = z_0)$
$(\ell_2, \{\ell_0\}, x = z_0 \wedge x \geq p_1)$
$(\ell_3, \emptyset, \delta)$
$(\ell_1, \{\ell_1\}, x = z_0 \wedge x \geq p_2)$
$(\ell_3, \{\ell_1\}, (x = z_0 \wedge x \geq p_2 \wedge z_0 \geq p_1))$

Figure 6: Non-interference synthesis example

*version and one non-vulnerable version. In the vulnerable version, if $x \leq secret$, the program sleeps for 1024 seconds; otherwise, it sleeps for 2048 seconds. In the non-vulnerable version, the program sleeps for 2048 seconds in both cases. Fig. 7 illustrates the model of the vulnerable version with 2 parameters $p_1$ and $p_2$. Since the program is executed on a server, the initialize steps ($\ell_1$, $\ell_2$, $\ell_3$) are hidden from the users, the corresponding transitions are labeled with $\tau$. Before applying* KALI, *we systematically remove all $\tau$-transitions in a way that does not alter the semantics of the PTA [41]. Note that one of the two branches is labeled with $h$.*

### 4.2 Experimental results

The statistics of the programs that we verified are shown in Table 1 where the first column is the name of the program, and the second column is the number of lines of the program. Note that multiple sets of parameters are used for behaviors depending on user inputs. The timing parameter $p_i$ denotes the number of unknown time units that a process decides to wait at a certain state (for mitigating timing attack). A few additional parameters are used to model program behaviors, e. g., parameter $x_{mi}$ is the (unknown but constant) position of the input that is different from the secret value; the parameter $x_{true}$ is the number of *true* in a random Boolean array, and the parameter $x_{len}$ is the length of the user input. Note that although these parameters are not timing parameters (i. e., they are discrete unknown constants), KALI supports such parameters naturally without any extension.

Table 1 shows the result of checking non-interference and non-deducibility using KALI, where the third and fourth column shows the time (in seconds) for the checkings, and the last column shows the synthesized constraints. The possibility of side-channel attacks is mitigated if the parameters satisfy the constraints. First, it can be observed that not only KALI terminates on each and every model, but also that it is rather efficient (within seconds). This is due to multiple reasons, i.e., the reduction techniques presented in Section 3 help to make sure the semi-algorithm is terminating; the step of systematically removing $\tau$-transitions before applying KALI often reduces the number of states in the PTAs significantly (which is especially important for the specification PTA); and the number of parameters is often limited in these programs. Second, the synthesized constraints are manually checked to be correct. The most complicated constraints are generated for `STAC:12e:v`, `STAC:12c:v`, and `STAC:12c:n`. In the following, we present some of the resultant constraints as examples. For instance, the parameter constraints returned from the verification for `STAC:1:v` is $p_1 - p_2 \leq 1024$ and $p_1 - p_2 \geq 1024$ against non-interference. That is to say, the program satisfies non-interference if and only if $p_1 = p_2 + 1024$. Referring to Fig. 7, it implies that both branches must wait for the same number of time units to mitigate timing attack. Lastly, we observe that the results are identical (after simplification) for checking against non-deducibility and non-interference. This is mainly due to how the *high* events are introduced. We are currently exploring theoretical results on when these two notions are the same so that we can reduce the verification time of non-deducibility by reducing it to the verification of non-interference.

## 5 RELATED WORK

In this section, we position our work with respect to previous approaches in the fields of (parametric) timed automata and timing attacks.

**(Parametric) TA and language inclusion checking:** This work is closely related to the line of research on defining and analyzing time-related security properties based on formalisms such as TA or PTA. In [23], Barbuti *et al.* defined the notion of timed non-interference. In [24], Cassez *et al.* proposed a related property called dense-time opacity for the discrete-event system, extending from opacity. It reveals that verifying opacity against the very restrictive class of event-recording automata [25] is already undecidable. In [26], Vasilikos *et al.* define the security of TA in terms 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 [27], André *et al.* define a notion of non-interference based on PTA and give a procedure to solve the problem without guarantee of termination. Their definition is different from ours, as their attacker can only observe the reachability of some discrete locations. Our work is inspired by the line of works on language inclusion checking for timed systems. Although the general problem of language inclusion checking for timed systems (i. e., those modeled using TA) is shown to be undecidable in [1], there have been attempts to address the problem by restricting the expressiveness of TA (e. g., [25], [28]) or through semi-algorithms which terminate often [16]. This algorithm developed in this work is inspired by [16], although [16] neither works with PTA nor is concerned with security properties such as non-interference or non-deducibility. Related to language inclusion is the problem of *learning* an unknown language: although the problem is again undecidable for TAs [1], the problem was tackled for the subclass of event-recording automata [25] in [29], [30], and extended to PTA in [31].

**Timing attacks:** Statically analyzing timing channels has been an active topic of research in the last two decades [32], [33], [34], [35], [36]. The general idea behind these works

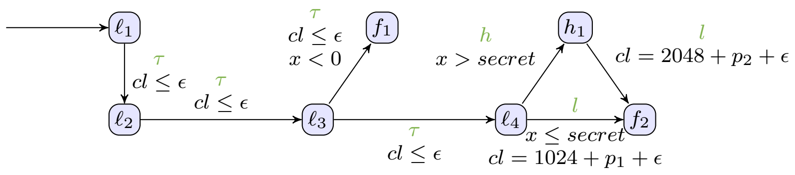


Figure 7: PTA modeling `STAC:1:v`

Table 1: Constraints synthesized and verification times

| Program | LOC | I-time (s) | D-time (s) | Constraints |
|---|---|---|---|---|
| `STAC:1:v` | 90 | 0.436 | 0.401 | $p_1 = p_2 + 1024$ |
| `STAC:1:n` | 90 | 0.700 | 0.894 | $p_1 = p_2$ |
| `STAC:3:v` | 108 | 0.447 | 0.832 | $p_{11} = p_{12}$ |
| | | 0.448 | 0.743 | $p_{22} = p_{21} + 380$ |
| | | 0.429 | 0.546 | $p_{31} = p_{32}$ |
| `STAC:3:n` | 108 | 0.441 | 0.786 | $p_{11} = p_{12}$ |
| | | 0.443 | 0.495 | $p_{11} = p_{12}$ |
| | | 0.448 | 0.490 | $p_{31} = p_{32}$ |
| `STAC:4:v` | 131 | 0.528 | 0.779 | $p_1 - p_2 + 1.5 \times x_{mi} = 15$ |
| `STAC:4:n` | 133 | 0.501 | 0.475 | $p_1 - p_2 + 0.15 \times x_{mi} = 15$ |
| `STAC:6:n` | 103 | 0.908 | 0.873 | $p_1 = p_2$ |
| `STAC:6:v` | 110 | 0.670 | 0.541 | $-p_1 + p_2 + 1.14 \times x_{mi} = 11.4$ |
| `STAC:11A` | 103 | 0.873 | 0.896 | $p_1 - p_2 = 1000$ |
| `STAC:11B` | 107 | 0.448 | 0.647 | $10 \times x_{true} = p_1 - p_2$ |
| `STAC:12e:v` | 102 | 1.009 | 3.339 | $p_4 - p_3 = 5 \wedge p_1 = p_2 = p_4$ |
| `STAC:12c:v` | 102 | 0.762 | 3.814 | $p_4 - p_3 = 5 \wedge p_1 = p_2 = p_4$ |
| `STAC:12c:n` | 117 | 0.737 | 3.793 | $p_3 - p_1 = 0.08 \wedge p_2 = p_3 = p_4$ |
| `STAC:14:n` | 109 | 0.363 | 0.332 | $0.01 \times x_{mi} - p_1 + p_2 = 0.01 \times x_{len}$ |

is to statically model the side-channel behavior, including micro-architectural side channels. Then, based on these models, the leakage of information from a program is analyzed. Existing works have also focused on testing the presence of timing channels, including finding the violations of the constant-time programming paradigm [37], maximizing the timing difference in secret-dependent executions [38] or finding timing-leakage in the presence of speculation [39]. However, none of these works synthesize provable mitigation to seize timing attacks, no focus on mitigating timing attacks. Contrary to the works that focus on the synthesis of side-channel attacks, such as recent work using symbolic models [42], our work focuses on the synthesis of mitigation. Our proposed approach is orthogonal to the line of research based on the constant-time programming paradigm [43], as our proposed framework does not rely on any specific programming styles or rules. In particular, we model arbitrary programs via PTAs and synthesize parameters that result in provable mitigation.

## 6 Conclusion

In this work, we propose a novel (and first) semi-algorithm for the *parametric timed language inclusion checking* of PTA, which overcomes the limitations of TA in the verification of systems with incomplete specifications. To this end, we propose new zone abstraction and simulation reduction techniques for solving the problem. We then apply the language inclusion checking to the mitigation of timing attacks. In particular, we reduce the mitigation of timing attacks to a language inclusion checking problem between two PTAs. The synthesis results on the parameters provide a provable guarantee against timing attacks. This checking is an important one not only for mitigating timing attacks but also in the general model checking community. We therefore believe that our procedure implemented in our toolkit can have applications beyond the security problems we considered in this work. Future works include more aggressive state space reduction techniques to ensure better scalability of our algorithm.

## References

[1] Alur, Rajeev, and David L. Dill. A theory of timed automata. Theoretical computer science 126.2 (1994): 183-235.
[2] Alur, Rajeev, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. Proceedings of the twenty-fifth annual ACM symposium on Theory of computing. 1993.
[3] Jovanovic, Aleksandra, Didier Lime, and Olivier H. Roux. Integer Parameter Synthesis for Timed Automata. TACAS. Vol. 7795. 2013.
[4] Dhem, Jean-Francois, et al. A practical implementation of the timing attack. International Conference on Smart Card Research and Advanced Applications. Springer, Berlin, Heidelberg, 1998.
[5] Benattar, Gilles, et al. Control and synthesis of non-interferent timed systems. International Journal of Control 88.2 (2015): 217-236.
[6] Sutherland, David. A model of information. Proceedings of the 9th national computer security conference. Vol. 247. 1986.
[7] Khasawneh, Khaled N., et al. Safespec: Banishing the spectre of a meltdown with leakage-free speculation. 2019 56th ACM/IEEE Design Automation Conference (DAC). IEEE, 2019.
[8] Yarom, Yuval, and Katrina Falkner. FLUSH+ RELOAD: A high resolution, low noise, L3 cache side-channel attack. 23rd USENIX Security Symposium (USENIX Security 14). 2014.
[9] Kiriansky, Vladimir, et al. DAWG: A defense against cache timing attacks in speculative execution processors. 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2018.
[10] Wu, Meng, et al. Eliminating timing side-channel leaks using program repair. Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2018.

[11] Bond, Barry, et al. ”Vale: Verifying high-performance cryptographic assembly code.” 26th USENIX Security Symposium (USENIX Security 17). 2017.

[12] Rane, Ashay, Calvin Lin, and Mohit Tiwari. Raccoon: Closing digital side-channels through obfuscated execution. 24th USENIX Security Symposium (USENIX Security 15). 2015.

[13] Gardey, Guillaume, John Mullins, and Olivier H. Roux. Non-interference control synthesis for security timed automata. Electronic Notes in Theoretical Computer Science 180.1 (2007): 35-53.

[14] Schrijver, Alexander. Theory of linear and integer programming. John Wiley & Sons, 1998.

[15] André, Étienne. What's decidable about parametric timed automata?. International Journal on Software Tools for Technology Transfer 21.2 (2019): 203-219.

[16] Wang, Ting, et al. Are timed automata bad for a specification language? language inclusion checking for timed automata. International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2014.

[17] Baier, Christel, et al. When are timed automata determinizable?. International Colloquium on Automata, Languages, and Programming. Springer, Berlin, Heidelberg, 2009.

[18] De Wulf, Martin, et al. Antichains: A new algorithm for checking universality of finite automata. International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, 2006.

[19] De Moura, Leonardo, and Nikolaj Bjørner. Z3: An efficient SMT solver. International conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2008.

[20] Dantzig, George B. Fourier-Motzkin elimination and its dual. STANFORD UNIV CA DEPT OF OPERATIONS RESEARCH, 1972.

[21] Lv, Mingsong, et al. Combining abstract interpretation with model checking for timing analysis of multicore software. 2010 31st IEEE Real-Time Systems Symposium. IEEE, 2010.

[22] André, Étienne, and Jun Sun. Parametric timed model checking for guaranteeing timed opacity. International Symposium on Automated Technology for Verification and Analysis. Springer, Cham, 2019.

[23] Barbuti, Roberto, and Luca Tesei. A decidable notion of timed non-interference. Fundamenta Informaticae 54.2-3 (2003): 137-150.

[24] Cassez, Franck. The dark side of timed opacity. International conference on information security and assurance. Springer, Berlin, Heidelberg, 2009.

[25] Alur, Rajeev, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. Theoretical Computer Science 211.1-2 (1999): 253-273.

[26] Vasilikos, Panagiotis, Flemming Nielson, and Hanne Riis Nielson. Secure information release in timed automata. International Conference on Principles of Security and Trust. Springer, Cham, 2018.

[27] André, Étienne, and Aleksander Kryukov. Parametric non-interference in timed automata. 2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS). IEEE, 2020.

[28] Abdulla, Parosh Aziz, et al. Universality analysis for one-clock timed automata. Fundamenta Informaticae 89.4 (2008): 419-450.

[29] Grinchtein, Olga, Bengt Jonsson, and Martin Leucker. Learning of event-recording automata. Theoretical Computer Science 411.47 (2010): 4029-4054.

[30] Lin, Shang-Wei, et al. Learning assumptions for compositional verification of timed systems. IEEE Transactions on Software Engineering 40.2 (2013): 137-153.

[31] André, Étienne, and Shang-Wei Lin. Learning-based compositional parameter synthesis for event-recording automata. International Conference on Formal Techniques for Distributed Objects, Components, and Systems. Springer, Cham, 2017.

[32] Doychev, Goran, et al. Cacheaudit: A tool for the static analysis of cache side channels. ACM Transactions on Information and System Security (TISSEC) 18.1 (2015): 1-32.

[33] Köpf, Boris, and David Basin. An information-theoretic model for adaptive side-channel attacks. Proceedings of the 14th ACM conference on Computer and communications security. 2007.

[34] Köpf, Boris, Laurent Mauborgne, and Martín Ochoa. Automatic quantification of cache side-channels. International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, 2012.

[35] Pasareanu, Corina S., Quoc-Sang Phan, and Pasquale Malacaria. Multi-run side-channel analysis using Symbolic Execution and Max-SMT. 2016 IEEE 29th Computer Security Foundations Symposium (CSF). IEEE, 2016.

[36] Wang, Shuai, et al. Identifying cache-based side channels through secret-augmented abstract interpretation. 28th USENIX Security Symposium (USENIX Security 19). 2019.

[37] He, Shaobo, Michael Emmi, and Gabriela Ciocarlie. ct-fuzz: Fuzzing for Timing Leaks. 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST). IEEE, 2020.

[38] Nilizadeh, Shirin, Yannic Noller, and Corina S. Pasareanu. DifFuzz: differential fuzzing for side-channel analysis. 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019.

[39] Guo, Shengjian, et al. SpecuSym: Speculative symbolic execution for cache timing leak detection. Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020.

[40] Köpf, Boris, and Markus Dürmuth. A provably secure and efficient countermeasure against timing attacks. 2009 22nd IEEE Computer Security Foundations Symposium. IEEE, 2009.

[41] Sun, Jun, et al. Verifying stateful timed CSP using implicit clocks and zone abstraction. International Conference on Formal Engineering Methods. Springer, Berlin, Heidelberg, 2009.

[42] Phan, Quoc-Sang, et al. Synthesis of adaptive side-channel attacks. 2017 IEEE 30th Computer Security Foundations Symposium (CSF). IEEE, 2017.

[43] Almeida, José Bacelar, et al. Verifying constant-time implementations. 25th USENIX Security Symposium (USENIX Security 16). 2016.

[44] Roscoe, A. W., J. C. P. Woodcock, and Lars Wulf. Non-interference through determinism. European Symposium on Research in Computer Security. Springer, Berlin, Heidelberg, 1994.

[45] McLean, John, Jon Millen, and Virgil Gligor. Non-interference, who needs it?. Proceedings of the IEEE Workshop on Computer Security Foundations. 2001.

[46] Focardi, Riccardo, Anna Ghelli, and Roberto Gorrieri. ”Using non interference for the analysis of security protocols.” Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols. September, 1997.

[47] Ouaknine, Joël, and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004.. IEEE, 2004.