École doctorale IAEM Lorraine

Contributions théoriques et algorithmiques pour l'analyse de propriétés de sûreté et de sécurité dans les systèmes temporisés sous incertitude

THÈSE

présentée et soutenue publiquement le 3 octobre 2023

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Dylan Marinho

Composition du jury

Président :	Véronique Cortier	Directrice de recherche
Rapporteurs :	Patricia Bouyer-Decitre	Directrice de recherche
	Thierry Jéron	Directeur de recherche
Examinateurs :	Thao Dang	Directrice de recherche
	Frédéric Herbreteau	Maître de conférences
	Swen Jacobs	*Tenure-track faculty*
Encadrant :	Étienne André	Professeur des universités
	Stephan Merz	Directeur de recherche

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

À mes parents,

À mon frère,

Je pourrais vivre encore mille ans
Ça n'serait pas suffisant, tu sais,
Pour te dire tout ce que je t'aime, maman.
Pourtant j'étais pas facile tout le temps,
Il t'a fallu du cran, c'est vrai
Mais rien n'est trop pour une reine, maman.

Ça traversera le temps
Comme un immortel amour
Merci à vous mes parents
J'aime la vie grâce à vous
J'aime la vie grâce à vous

J'essaie de te suivre dans tes pas
Ton courage de soldat, écoute
Sais-tu tout ce que tu m'inspire, papa
Je n'ai pas autant de force que toi
Je plie parfois sous le poids, c'est vrai
Mais j'espère que tu es fier de moi

Ça traversera le temps
Comme un immortel amour
Merci à vous mes parents
J'aime la vie grâce à vous
J'aime la vie grâce à vous

— Mickaël Pouvin, Grâce à vous

CONTENTS

Introduction and preliminaries

II Execution-time opacity

General conclusion

ABSTRACT

Real-time systems can be used in a wide range of applications, such as transport, telecommunications and industry. However, accidents can happen, and it is necessary to have confidence in these systems in order to avoid them. It is therefore necessary to formally prove that their behavior will comply with a specification. This specification can be of two kinds: with safety properties, showing that the system will always behave as expected, and security properties, showing that it will be resistant to certain attacks. For this, the formalism of timed automata (TAs) [AD94] is fairly common. However, this modeling may be imperfect, due to the nature of the system, needed simplifications or imprecisions. We therefore study these timed systems under uncertainty, i.e. using parameters. The natural extension studied is the formalism of parametric timed automata (PTAs) [AHV93].

First, we focus on efficient verification methods for PTAs. A benchmark library for parametric timed model-checking is presented, allowing us to compare different algorithms for PTAs. Next, we study the case of state merging in the parametric zone graph (PZG) of a PTA: if the union of the constraints of two states with the same location is convex, then these two states can be merged. We propose an algorithm, implement it and compare different heuristics. We show that, in practice, this method reduces the computation time by an average of 62%.

Next, we introduce a notion of opacity on PTAs. In our formalism, an attacker seeks to determine a secret (expressed in terms of visiting a location) knowing only the total execution time of the system (as well as the model). We formally define this notion and study two types of problem: deciding that a system expressed as a TA is opaque, and determining the parameter valuations of a PTA to ensure the opacity of the associated TA. We then extend this definition to the case of secrets with expiration: in this formalism, after a certain delay, finding a secret is useless for the attacker. We then address the decision problem as well as the problem of computing the expiration date to ensure that a TA is opaque. A parameterized extension is also studied, with the synthesis of parameters in a PTA. For the different problems, we show decidability results and propose some algorithms to solve them. We finally present a first version of untimed control associated with our opacity formalism. In this work, we seek to highlight a set of actions so that a PTA restricted to this set is opaque; an algorithm and an implementation are proposed.

REMERCIEMENTS

Cette thèse est le fruit d'un long travail, qui n'aurait pu être possible sans le soutien de nombreuses personnes.

Je tiens en premier lieu à remercier mon directeur de thèse, Étienne André, pour tous nos échanges, ses conseils et son accompagnement durant ces trois années. Il a été un directeur de thèse exemplaire, toujours disponible malgré un contexte parfois difficile. Je remercie également Stephan Merz d'avoir aimablement accepté de rejoindre la direction de thèse et de nous avoir épaulé pour toutes les démarches administratives ; cependant, au delà d'un simple soutien administratif, c'est notamment en dirigeant avec patience et douceur l'équipe VeriDis, qu'il a créé un environnement de travail idéal.

Je remercie également tous les personnes qui ont dédié une partie de leur temps à l'évaluation de cette thèse : Patricia Bouyer-Decitre et Thierry Jéron qui ont accepté d'en être les rapporteurs et ont fourni des retours très constructifs, ainsi que Véronique Cortier, Thao Dang, Frédéric Herbreteau et Swen Jacobs pour les discussions très enrichissantes que nous avons eues.

Cette thèse est aussi le fruit de nombreuses collaborations, avec les différents membres du projet ProMiS, et en premier lieu Didier Lime et Sun Jun (sans oublier Étienne André), mais aussi Laure Petrucci, Jaco van de Pol et Engel Lefaucheux. Sans les (quelques) heures de réflexions communes devant des tableaux, parfois un peu trop blancs, cette thèse n'aurait certainement pas vu le jour.

Je dois également remercier toutes les personnes composant l'équipe Mosel/Veridis, tant pour les échanges scientifiques que pour leur soutien, leurs conseils ou nos discussions à table : Marie, Stephan, Horatiu, Dominique, Sophie, Engel, Johan. Je n'oublie pas non plus ceux qui ont été mes co-bureaux (ou presque) durant ces trois années : ceux qui étaient déjà là à mon arrivée et qui m'ont accueilli – Pierre, Hans, Rosalie – et ceux qui nous ont ensuite rejoint, Benjamin, Thomas et Alessio.

J'ai aussi eu la chance de côtoyer quelque temps les membres composant l'équipe LoVe du LIPN, que je remercie pour leur accueil et leurs conseils, particulièrement au cours de ma dernière année, ainsi que Sophie et Sylvie avec qui j'ai pu partager un bureau (et un coin d'étagère).

Cette thèse est également l'aboutissement de plusieurs années d'études, ponctuées et motivées par l'engagement de plusieurs enseignants qui ont su transmettre avec passion leur amour des mathématiques ou de l'informatique (voire même d'autres disciplines). Je tiens à tous les remercier, c'est grâce à eux que j'ai pu arriver à cette étape de ma vie. Plus particulièrement, je remercie Audrey Poirier et Ludovic Bossard qui ont joué un rôle particulièrement important dans ce parcours.

Merci à tous mes amis pour tous les bons moments de détente, si importants, de discussions, d'échanges, de partage. Merci Véronique, Mathis, Anthony (notamment pour ces nuits parfois un peu trop blanches), Olivier (et Emma, et Alexandre), Jean-Daniel (et Auriane, et Alexandre), Marine, et tous ceux que j'oublie. Merci également à ceux qui ont ponctué des étapes de ma vie, même si le temps et les circonstances ont fait que nous nous sommes éloignés.

Enfin, merci à ma famille, et en particulier à mes parents et à mon frère, pour leur soutien depuis toujours. Je ne pourrai jamais assez les remercier pour tout ce qu'ils m'ont apporté et pour leur soutien indéfectible. Enfin, merci Pierre d'avoir été si présent au cours de ces trois années (et de celles qui les ont précédées), d'avoir occupé une place assez difficile, je le sais, au cours de ces années, celle de me supporter et de me soutenir jour après jour ; "*parce que c'est toi, parce que t'es là, je n'ai plus peur du dimanche soir.*" (Grand Corps Malade)

PUBLISHED ARTICLES

The results presented in this thesis have been published in the following publications:

[ABLM22] Étienne André, Shapagat Bolat, Engel Lefaucheux, and Dylan Marinho. “strategFTO: Untimed control for timed opacity”. In: *Proceedings of the 8th ACM SIGPLAN International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2022)* (Dec. 7, 2022). Ed. by Cyrille Artho and Peter Ölveczky. Auckland, New Zealand: ACM, 2022, pp. 27–33. DOI: 10.1145/3563822.3568013.

[ALLMS23] Étienne André, Engel Lefaucheux, Didier Lime, Dylan Marinho, and Jun Sun. “Configuring Timing Parameters to Ensure Execution-Time Opacity in Timed Automata”. In: *Proceedings of the First Workshop on Trends in Configurable Systems Analysis (TiCSA@ETAPS 2023)* (Apr. 23, 2023). Ed. by Maurice H. ter Beek and Clemens Dubslaff. Vol. 392. EPTCS. Paris, France, 2023, pp. 1–26. DOI: 10.4204/EPTCS.392.1.

[ALM23] Étienne André, Engel Lefaucheux, and Dylan Marinho. “Expiring opacity problems in parametric timed automata”. In: *Proceedings of the 27th International Conference on Engineering of Complex Computer Systems (ICECCS 2023)* (June 12–16, 2023). Ed. by Yamine Ait-Ameur and Ferhat Khendek. Vol. 13260. Toulouse, France: Springer, 2023, pp. 451–469. DOI: 10.1109/ICECCS59891.2023.00020.

[ALMS22] 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* 31.4 (2022), 64:1–64:36. DOI: 10.1145/3502851.

[AMP21] Étienne André, Dylan Marinho, and Jaco van de Pol. “A Benchmarks Library for Extended Parametric Timed Automata”. In: *Proceedings of the 15th International Conference on Tests and Proofs (TAP 2021), Held as Part of STAF 2021* (June 21–22, 2021). Ed. by Frédéric Loulergue and Franz Wotawa. Vol. 12740. Lecture Notes in Computer Science. Virtual Event: Springer, 2021, pp. 39–50. DOI: 10.1007/978-3-030-79379-1_3.

[AMPP22] Étienne André, Dylan Marinho, Laure Petrucci, and Jaco van de Pol. “Efficient Convex Zone Merging in Parametric Timed Automata”. In: *Proceedings of the 20th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2022)* (Sept. 13–15, 2022). Ed. by Sergiy Bogomolov and David Parker. Vol. 13465. Lecture Notes in Computer Science. Warsaw, Poland: Springer, 2022, pp. 200–218. DOI: 10.1007/978-3-031-15839-1_12.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF DEFINITIONS

LIST OF THEOREMS

ACRONYMS

*∀CTL**

Universal fragment of CTL∗ (Appendix A on page 183) 56

(P)TA

(Possibly parametric) TA 9

BFS

Breadth-first search 22

CTL

Computation tree logic (Appendix A on page 182) 182

DFA

Deterministic finite-state automaton (Definition 3.1 on page 25) 25

ERA

Event-recording automaton 7

ET-opacity

Execution-time opacity (Definition 7.1 on page 80) 9

ET-opaque

Execution-time opaque 10

exp-ET-opacity

Expiring execution-time opacity (Definition 8.1 on page 114) 10

exp-ET-opaque

Expiring execution-time opaque 111

HA

Hybrid automaton 22

IM

Inverse method (also called trace preservation synthesis (TPS)) 42

IPTA

IMITATOR parametric timed automaton (Section 4.1 on page 43) 43

L/U-PTA

Lower/upper parametric timed automaton (Definition 3.16 on page 36) 21

LTL

Linear temporal logic (Appendix A on page 181) 58

LTS

Labeled transition system (Definition 3.2 on page 26) 7

MRA

Multi-rate automaton 41

PGA

Parametric timed game automaton 21

PN

Petri net 4

PPL

Parma Polyhedra Library 37

PTA

Parametric timed automaton (Definition 3.11 on page 32) . . vii

PZG

Parametric zone graph (Definition 3.15 on page 34) vii

RTA

Real-time automaton 8

SNNI

Strong non-deterministic non-interference 19

TA

Timed automaton (Definition 3.5 on page 28) vii

TGA

Timed game automaton 21

TPS

Trace preservation synthesis 42

TTS

Timed transition system (Definition 3.4 on page 27) 26

U-PTA

Upper parametric timed automaton 93

INTRODUCTION AND PRELIMINARIES

If you can keep your head when all about you
Are losing theirs and blaming it on you;
If you can trust yourself when all men doubt you,
But make allowance for their doubting too;
If you can wait and not be tired by waiting,
Or, being lied about, don't deal in lies,
Or, being hated, don't give way to hating,
And yet don't look too good, nor talk too wise;

If you can dream—and not make dreams your master;
If you can think—and not make thoughts your aim;
If you can meet with triumph and disaster
And treat those two impostors just the same;
If you can bear to hear the truth you've spoken
Twisted by knaves to make a trap for fools,
Or watch the things you gave your life to broken,
And stoop and build 'em up with wornout tools;

If you can make one heap of all your winnings
And risk it on one turn of pitch-and-toss,
And lose, and start again at your beginnings
And never breathe a word about your loss;
If you can force your heart and nerve and sinew
To serve your turn long after they are gone,
And so hold on when there is nothing in you
Except the Will which says to them: "Hold on";

If you can talk with crowds and keep your virtue,
Or walk with kings—nor lose the common touch;
If neither foes nor loving friends can hurt you;
If all men count with you, but none too much;
If you can fill the unforgiving minute
With sixty seconds' worth of distance run–
Yours is the Earth and everything that's in it,
And–which is more–you'll be a Man, my son!

— Rudyard Kipling, If

1

INTRODUCTION

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth

Complex concurrent systems are used in many areas, such as transports, communications or industrial units, and become more and more intricate. They are composed of many entities, interacting to each other, with complex behaviors. Their ubiquity (e. g., for telecommunication items), their cost (e. g., for space rockets) or their criticality (e. g., autonomous cars or planes) require to have confidence in such systems.

However, accidents often occur due to the lack of behavioral studies or robust verification. That is, when a system faces some perturbations (such as switching guidance mode or weather conditions), issues can appear. For example, the launching of the new SpaceX's rocket had to be delayed on 17 April 2023, due to pressurization issues. Three days after, the second attempt lead to the destruction of the Starship rocket, as well as the launching pad, when the engines broke down one after the other. One can also raise the Chernobyl disaster, on 26 April 1986, whose one of the main causes is a misconducted test experiment. More recently, the Fukushima Daiichi disaster, on 11 March 2011, is due to an error in the cooling system after reactors' electricity provision failed.

Modeling and verification of such systems can help to avoid these undesirable crashes, even if they can never totally avoid them: every model includes some approximations or assumptions under which a system is verified. Nevertheless, they offer a rigorous procedure for limiting the possibility of unexpected behavior. Throughout this thesis, we will be interested in the verification of safety properties – asking if a system will behave well – as well as security properties – asking if a system could withstand a thoughtless user trying to attack the system to reveal some secrets.

1.1 TIMING LEAKS

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. They can appear when the time needed to execute some instructions, or the whole program, depends on private data, such as user-input variables or the status of the memory.

Diverse examples of attacks based on timing leaks are presented in the literature with various application cases. For example, [CHSJX22] proves that it is possible to break the SM2 signature algorithm (the Chinese standard for public key cryptography) with a deduction on the most significant zero bits based on the execution time. Among well-known side-channel attacks based on time, the Spectre vulnerability was revealed in early 2018 [Koc+20]. The main vulnerability comes from the out-of-order properties of modern complex cores, allowing to enforce the presence in the cache of forbidden values. However, it permits information leakage since an attacker could retrieve these values only checking the needed time to access a given one. Five years after its disclosure, this vulnerability persists to be a research field, yielding to recent publications. Therefore, it remains as a challenge to automatically detect, or mitigate, such vulnerabilities.

We aim to propose a method based on the formal verification of timed systems.

1.2 FORMAL VERIFICATION

When we need to obtain formal guarantees that a system satisfies some properties (such as not allowing leaks of information), we have to go beyond testing methods, sometimes used in practice. Indeed, checking that a system has had a good behavior for some executions cannot be a proof that it will never misconduct. However, having guarantees that, for any environment and for all executions, some properties will be verified can give confidence in a system. To this end, one may use formal verification techniques, such as model checking [BK08].

1.2.1 *Model checking*

Given a system and a property, model checking aims to confirm or deny that the model verifies the property. It is a very active field of research and has a very high importance in modern computer science, with the granting of the Turing Award in 2007 to three researchers for their work on this topic. Depending on the formalism used to describe

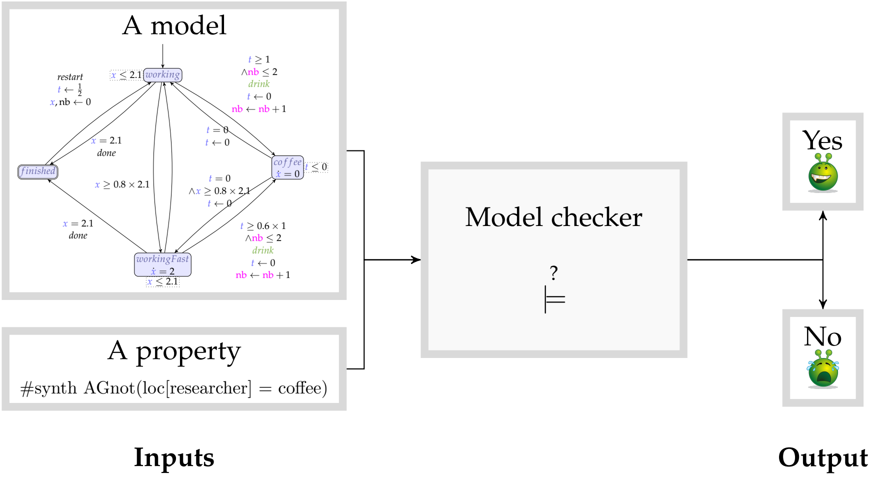


Figure 1.1: Model checking overview

the model and the property, different techniques can be applied to automatically check the validation of the property. Classical formalisms include (directed) graphs, automata and their extensions (such as timed automata (TAs) [AD94]) or Petri nets (PNs) [Pet62]. Properties can be expressed using various formalisms, such as temporal logics, allowing to describe, for example, the order on which the events may appear or their (relative) timestamps.

1.2.2 *Timed model checking*

In timed model checking, models and properties include timing aspects. This extension allows to verify specifications when events must (or must not) occur within or after a certain duration.

We use the well-known modeling of TAs to describe our systems. In few words, TAs extend classical finite-state automata with a set of clocks (which are real-valued variables evolving at the same rate). These clocks can be compared to integers within invariants or guards (constraints to verify to stay in a location or to enable a transition). The formalism of TAs benefits from nice decidability results, justifying their use to model several classes of systems and to verify their properties. We can mention, for example, the decidability of the reachability properties [AD94] with a fair complexity (PSPACE-complete). TAs are formally defined in Chapter 3.

In Figure 1.2, we present a model of a coffee machine within the formalism of TAs. Clocks x and y are used to represent the time and are compared to integers to restrict the timing behavior of the model. An execution of this system begins in the initial location ℓ_0 ("idle"). After an arbitrarily long time (there are no conditions for staying in or leaving ℓ_0), one can move (instantly) to ℓ_1 ("add sugar") while clocks x

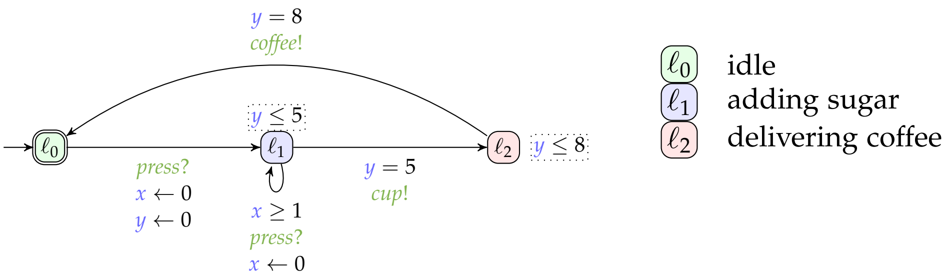


Figure 1.2: A coffee machine modeled with a TA

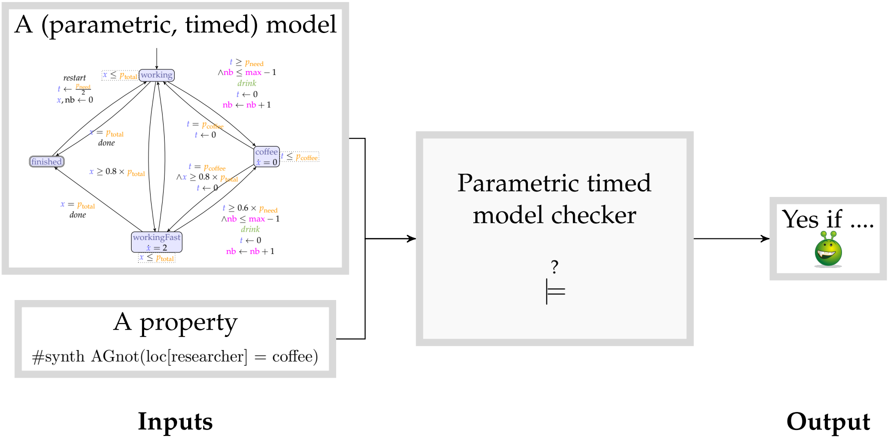


Figure 1.3: Parametric timed model checking overview

and y are reset. It is possible to stay in ℓ_1 for five time units (invariant "$y \leq 5$") and use the *press*? action to add a sugar; this transition can be taken when $x \geq 1$ and x is reset along it. When $y = 5$, the transition "*cup*!" must be taken and the run reaches ℓ_3 ("delivering coffee"). After three time units (to enable the *coffee*! transition, guarded by "$y = 8$"), the system can reach the final location ℓ_0.

Timed model checking allows to verify timing properties over this kind of models. Therefore, one is able to verify properties such as "It is possible to have a coffee without any sugar" or "It is always possible to have a coffee in less than 5 time units". Its global overview is depicted in Figure 1.1, with timing models and (timing) properties as inputs.

Among the timed model checkers, we can mention UPPAAL [LPY97], PAT [SLDP09] and TCHECKER [HPT].

1.2.3 *Parametric timed model checking*

Parametric (timed) model checking goes beyond classic (timed) model checking by allowing to have not only binary yes/no answers, but to ask to compute *parameter valuations* ensuring a property. Its overview is then a little bit different, as displayed in Figure 1.3.

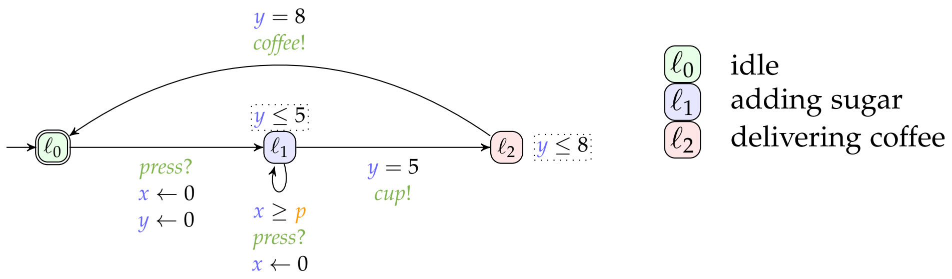


Figure 1.4: A parametrization of the coffee machine of Figure 1.2

Using parameter synthesis handles important needs. First, one may be interested by robustness properties: numbers may not be implemented exactly and assuming exact values may lead to malfunctions. Moreover, it allows to fully explore a system, and not only its correctness according to some values. Finally, one can also be interested in finding the good values to design a system according to some requirements.

The classical parametric extension of TAs is parametric timed automata (PTAs) [AHV93], where TAs are extended with a set of parameters (which are unknown constants of the system). Their semantics consist in the union, for all parameter valuations, of the valuated TA semantics. However, these semantics are commonly expressed using parametric zones, i. e., a data structure representing a dense set of clock and parameter values, often represented by a polyhedron. We define formally PTAs and their semantics in Chapter 3. Figure 1.4 presents the same coffee machine as previous Figure 1.2, with a parameter p controlling the delay between two presses in the "add sugar" button. With these models, one may ask to know "For which values of p is it possible to have a coffee with six doses of sugar?".

A well-known model checker supporting PTAs is IMITATOR [And21a].

1.3 THE NOTION OF OPACITY

In its most general form on partially observed labeled transition systems (LTSs) [Kel76], 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 [BKMR08]. This secret is completely generic and, depending on its actual definition, properties and their decidability can differ.

Franck Cassez proposed in [Cas09] a first definition of timed opacity for TAs: 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. It is then proved in [Cas09] that it is undecidable whether a TA is opaque, even for the restricted class of event-recording automata (ERAs) [AFH99] (a subclass of TAs). This notably relates to the undecidability of timed language inclusion for TAs.

The aforementioned negative result leaves hope only if the definition or the setting is changed, which was done in two main lines of works.

First, in [WZ18; WZA18], the input model is simplified to real-time automaton (RTA), a severely restricted formalism compared to TAs. With these models, timed aspects are only considered by interval restrictions over the total elapsed time along transitions: RTAs can be seen as a subclass of TAs with a single clock, reset at each transition. In this setting, (initial-state) opacity becomes decidable.

Second, in [AETYM21], the authors consider a time-bounded notion of the opacity of [Cas09], where the attacker has to disclose the secret before an upper bound, using a partial observability. This can be seen as a secrecy with an *expiration date*. The rationale is that retrieving a secret "too late" is useless; this is understandable, e. g., when the secret is the value in a cache; if the cache was overwritten since, then knowing the secret is probably useless in most situations. In addition, the analysis is carried out over a time-bounded horizon; this means there are two time bounds in [AETYM21]: one for the secret expiration date, and one for the bounded-time execution of the system. The authors prove that whether a system is time-bounded opaque under a bounded time horizon, with a notion close to our weakness definition (Definition 8.1, with unidirectional language inclusion), is decidable for TAs. A construction and an algorithm are also provided to solve it; a case study is verified using SpaceEx [Fre+11].

A more extended presentation of related works is presented in Chapter 2.

1.4 PROBLEM STATEMENT

This thesis aims to propose novel efficient verification techniques for safety and security properties in timed systems under uncertainty.

This goal is broken down into two sub-problems.

ON EFFICIENT VERIFICATION

- How can we efficiently verify a (parametric) timed system?
- How can we compare different methods of (parametric) timed model checking?

ON SECURITY PROPERTIES

- For which classes of models can we decide whether an attacker is able to deduce private information from a timed system with the observation of its execution time?
- If secrets have an expiration date, can we have guarantees that an attacker is unable to deduce private information from the total execution time?
- Can we control a system so that it remains secure?
- How can we fix a non-secure system?

1.5 CONTRIBUTIONS

In this thesis, we present five different scientific contributions: two about zone merging in PTAs (Section 1.5.1) and three concerning execution-time opacity (ET-opacity) (Section 1.5.2). As an additional contribution, we published the technical code and data resulting from the theoretical and algorithmic contributions from this thesis in long-term open access venues (Section 1.5.3).

1.5.1 *Zone merging in parametric timed automata*

PTAs are a powerful formalism for reasoning on concurrent real-time systems with unknown or uncertain timing constants. Therefore, we also focus on efficient techniques for parametric timed verification. To this end, we work on a dedicated benchmark library before studying efficient techniques to construct and explore PTA state space.

A benchmark library for parametric timed model checking

In order to test the efficiency of new algorithms, a fair set of benchmarks is required. We present an extension of the IMITATOR benchmark library, that accumulated over the years a number of case studies from academic and industrial contexts. We reformat the library into a better-structured repository, significantly improving its organization. We also extend the library with several dozens of new benchmarks; these benchmarks highlight several new features: liveness properties, extensions of (possibly parametric) TAs ((P)TAs) (including stopwatches or multi-rate clocks), and unsolvable toy benchmarks. These latter additions help to emphasize the limits of state-of-the-art parameter synthesis techniques, with the hope to develop new dedicated algorithms in the future.

This work was conducted in collaboration with Étienne André and Jaco van de Pol, and published in [AMP21a].

Efficient convex zone merging

Reducing the state space of PTAs is a significant way to reduce the inherently large analysis times. We study different convex zone merging reduction techniques based on convex union of constraints (making up parametric zones), allowing to decrease the number of states while preserving the correctness of verification and synthesis results. We perform extensive experiments, and identify the best heuristics in practice, bringing a significant decrease in the computation time on the aforementioned benchmark library.

This work was conducted in collaboration with Étienne André, Laure Petrucci and Jaco van de Pol, and published in [AMPP22a].

1.5.2 *Execution-time opacity*

In our setting, we define a form of opacity in which the observation is only *the time to reach a given final location*. Therefore, we are interested in knowing if an attacker could deduce some private information knowing only the execution time of a model. It is ET-opaque if there exist two runs reaching its final location, one visiting a given (private) location and one not visiting it, of the same duration. That is, for this particular execution time, the system is ET-opaque if one cannot deduce whether the system visited the private location.

Guaranteeing execution-time opacity

We address the following ET-opacity problem: given a timed system with 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 visited the private location. We are interested in the existence of a such execution time: if there exists one, the system is ∃-ET-opaque. We also consider the full ET-opacity decision problem, asking whether the system is ET-opaque for all execution times. We show that these problems are decidable for TAs but become undecidable when one adds parameters, yielding PTAs. We identify a subclass of PTAs with some decidability results. We then devise an algorithm for synthesizing PTA parameter valuations guaranteeing that the resulting TA is ET-opaque. We finally show that our method can also apply to program analysis.

This work was conducted in collaboration with Étienne André, Didier Lime and Sun Jun, and published in [ALMS22].

Expiring execution-time opacity problems

We propose a definition of expiring execution-time opacity (exp-ET-opacity): in addition to the first definition of ET-opacity, the secrecy is violated only when the private location was entered "recently", i. e., within a given time bound (or expiration date) prior to system completion. This has an interesting parallel with concrete applications, notably cache deducibility: it may be useless for the attacker to know the cache content too late after its update. We study exp-ET-opacity problems in TAs: we consider the set of expiration dates for which a system is ET-opaque and show when they can be effectively computed for TAs. We then study the decidability of several parameterized problems, when not only the bounds, but also some internal timing constants become timing parameters of unknown constant values within the framework of PTAs.

This work was conducted in collaboration with Étienne André and Engel Lefaucheux, and published in [ALM23].

Untimed control for execution-time opacity

We focus on the untimed control problem: exhibiting a controller, i. e., a set of allowed actions, such that the system restricted to those actions is fully ET-opaque (i. e., ET-opaque for all its execution times). We first show that this problem is not more complex than the full ET-opacity decision problem, and then we propose an algorithm, implemented and evaluated in practice. We introduce a prototype tool strategFTO implementing this algorithm with some heuristics.

This work was conducted in collaboration with Étienne André, Shapagat Bolat and Engel Lefaucheux, and published in [ABLM22b].

1.5.3 *Open-access data*

The code and data resulting from the aforementioned theoretical contributions are published as long-term open access data.

Productions and experiments (Chapters 4, 5 and 7) are registered using Zenodo with Digital Object Identifiers (DOIs). When possible, they were presented as artifacts joined to the linked publications and were reviewed by the artifact evaluation committees of the conferences where the results were published. The experiments, scripts, input and result files of the following works are available.

- The IMITATOR library (Chapter 4) at https://doi.org/10.5281/zenodo.4730980 [AMP21b], which received the functional badge from the TAP'21 artifact evaluation committee.
- The comparison of merging heuristics (Chapter 5) at https://doi.org/10.5281/zenodo.6806915 [AMPP22b], which received the "evaluated" badge from the FORMATS'22 artifact evaluation committee.

- The scalability study of strategFTO (Chapter 9) at https://doi.org/10.5281/zenodo.7181848 [ABLM22a].

The source code of tools I developed for this thesis (Chapters 4, 5 and 9) is accessible on GitHub:

- the content of the IMITATOR library, at https://github.com/DylanMarinho/IMITATOR-library;
- the implementation of the merging techniques (Chapter 5) as options of IMITATOR, at https://github.com/imitator-model-checker/imitator;
- strategFTO (Chapter 9) at https://github.com/DylanMarinho/Controlling-TA.

1.6 OUTLINE OF THE MANUSCRIPT

After the general introduction in the present chapter, this thesis begins in Chapter 2 by a presentation of other works related to the problems we study. Then, we recall (parametric) timed model checking and general preliminary definitions in Chapter 3.

In Part I, we first present our benchmark library for parametric timed model checking in Chapter 4 and consider heuristics for efficient merging techniques in Chapter 5.

In Part II, we introduce and study ET-opacity. Chapter 6 introduces preliminary concepts for the following chapters. We study ET-opacity problems in Chapter 7, extended with expiring secrecy in Chapter 8. Finally, we focus on untimed control for ET-opacity in Chapter 9.

We conclude the thesis in Chapter 10.

2

RELATED WORKS

Bring the past only if you're going to build from it.

— Domenico Cieri Estrada

In this Chapter 2, we present works related to the problems we study.

Organization of the chapter

In Section 2.1, we present works concerning the mitigation of timing leaks not considering solutions based on verification and model-checking.

In Section 2.2, we present an overview of works considering hyperproperties.

In Section 2.3, works over timed model checking are recalled, including an overview over model checkers supporting TAs and their use. Section 2.4 extends the same overview over PTAs. In Section 2.5, we present an overview of security problems with TAs. In Section 2.6, we present works about the control of real-time systems.

Section 2.7 recalls work about the reduction of the state space of PTAs and Section 2.8 about libraries for (timed) model checking.

In Section 2.9, we present an overview of *parameterized* verification, where the parameter lies in the number of system components (e. g., number of processes, of clients, of agents. . .), with the aim at verifying the system for any such number of components.

2.1 TIMING LEAKS

Defeating timing channel leaks is challenging and can be done with two main approaches: (*i*) closing or mitigating the channel, and (*ii*) reducing the adversary power.

A particular example on mitigating the timing channel includes constant-time software techniques, which aim to implement software where the execution time does not depend on secret values. For example, [ABBDE16] proposes an approach to verify the constant-time policy application of a program. [BPT19] uses abstract interpretation to report time leakage in C programs; an implementation using the CompCert compiler toolchain [Ler09] and the Verasco static analyzer [JLBLP15] is presented.

In [WGSW18], the authors propose to eliminate timings leaks with program transformation. In practice, to an input program and a list of secret variables, their tool generates a transformed program where branches and loads are balanced, for example, with the insertion of dummy variables. That is, the output program is free of timing vulnerabilities. This method was implemented in LLVM and validated on a set of cryptographic applications.

[ELFL21] introduces guidelines to design immune cores to micro-architectural timing leaks. The authors propose two implementations using the RISC-V instruction set. [JWL18] and [GR21] propose to ensure opacity by function insertion, including fictitious events in the system.

2.2 HYPERPROPERTIES

Temporal logics for hyperproperties are introduced in [Cla+14] to characterize properties of sets of paths. Their allow to easily define security policies, as non-interference, adding the ability to quantify a property over multiple paths.

Model checking techniques for hyperproperties are presented in [Cla+14; FRS15; FV19] and monitoring is discussed in [FHST18; Hah19; FHST19; FHST20; CH23]. Some tools have been created to verify hyperproperties, as Weaver [FV19] and RVHyper [FHST18].

Time hyperproperties are considered in [BPS20; HZJ21]. In [BPS20] a particular focus is done on timing leaks and the authors provide a model checking algorithm.

2.3 TIMED MODEL CHECKING

2.3.1 *Timed automaton verification*

Since [AD94], the reachability emptiness (EF-emptiness) problem for TAs is shown to be decidable and PSPACE-complete. Some studies were conducted to exhibit subclasses with simpler complexity, e. g., TAs with one or two clocks [LMS04]. Cycle detection [AM04] is also decidable.

Despite these results, the universality problem (asking if the TA accepts all traces) is undecidable [AD94]. Moreover, inclusion and equivalence problems (inclusion or equality of TA languages) are undecidable.

These problems become decidable for the subclass of deterministic TAs. Note that, deterministic TA are strictly less expressive than TA, and it is undecidable to know if there exists a determinization of a given TA [Trio6]. The subclass of ERAs, where a clock is associated to each action and reset when it occurs, also provides decidability results (e. g., [AFH99]).

A survey of decidability results with TAs was proposed by [AM04].

2.3.2 *Model checkers supporting the timed automaton formalism*

Kronos [Yov97] is a software tool designed to be integrated into environments for real-time systems. It was used for the analysis of several processes. In [Ber+01], Kronos is joined to other tools to propose Taxys. This tool allows to capture the temporal behavior of an application, including the software, the computer and its environment.

TChecker [HPT] is an open-source model checker designed to experiment verification algorithms of real-time systems. That is, it produces not only an answer to the verification, but also statistics over the used algorithms. It provides the verification of emptiness and Büchi emptiness problems over TAs and comes with a collection of libraries to perform algorithm comparisons.

PAT [SLDP09] is a model checker designed to compose, simulate and reason on real-time systems with concurrency. It comes with user-friendly interfaces and implements various techniques of model checking.

Uppaal [LPY97] is an integrated tool for modeling, validating and verifying real-timed systems modeled in a TA formalism augmented with some features, as data types. Many features are provided by the authors, such as a graphical system editor, a simulator, the verification of safety and liveness properties and the generation of diagnostic traces.

2.3.3 *Using timed automata to ensure safety*

There have been many applications of model checking TAs to many areas, such as robotics, scheduling and protocols. In the following, we just cite a few as examples of applications.

In [HSLL97], the authors proposes to formally verify with Uppaal a message transmission controller used by an audio/video company (Bang&Olufsen). This controller aims to detect collisions between communications in a single bus. In practice, an error trace is generated by Uppaal and the mistake in the implementation is highlighted.

[LMNS05] presents the verification of an industrial case study with Uppaal-TRON, an extension of Uppaal for black-box conformance testing.

In [BC11], the authors uses TAs to model a hardware structure. Synchronizing this model with a control flow graph (CFG) of a binary program, they compute the worst case execution time (WCET) using Uppaal. Experiments are performed on the ARM9020T real platform.

In [SBP19], the authors propose a framework based on the verification of TAs using Uppaal to ensure cache coherence.

In [LKP19], TAs are used for the verification of time properties of Java programs. The authors introduce a formal definition of the Java time semantics and automatically extract TAs verified using Uppaal. A study over twenty programs is presented, leading to the identification of eight errors in these projects.

In [RFRJM19], runtime enforcement of (possibly timed) properties in TAs is presented. The main goal is to modify the execution of a system during its execution to ensure the verification of a given property. [NSJMM22] deals with the verification of program requirements and reparation using satisfiability modulo theories queries and TAs; an implementation is provided.

[FH20] proposes to perform timed formal verification of robotic specifications. To this end, the authors model and check these requirements from a robotic framework using TAs and Uppaal, being mainly focused on schedulability.

In [VNNK19], the authors study the impact of clock granularity (clocks that ticks periodically) on timing channel attacks. They model an adversary with a parametric model in the granularity of the clock, connected with a model of the system, using TAs. Their technique allows to derive insights about the effectiveness of attacks and countermeasures. A manual proof on a case study using RSA signatures is provided, without any implementation.

2.4 PARAMETRIC TIMED MODEL CHECKING

2.4.1 *Parametric timed automaton verification*

Since their definition in 1993 [AHV93], number of (un)decidability results of problems based on PTAs have been proved.

First, the membership problem (i. e., the emptiness of the language of a valuated PTA) is shown to be undecidable in the general case (if there are irrational parameter values) [Mil00], even if it is decidable with simpler formalisms (integer-valued parameters or discrete time) [AD94].

The EF-emptiness problem ("is a location reachable for some valuation?") is shown to be undecidable by the seminal paper on PTAs [AHV93]. Other classical problems are also undecidable in general: AF-emptiness [ALR16; JLR15], AG-emptiness [ALR16] and EG-emptiness [AL17a].

Studying the properties over a bounded time can lead to some decidability results (e. g., language inclusion of valuated PTAs [OW10]), but other results remain undecidable (e. g., EF-emptiness [Jov13]).

Since [AHV93], different works were conducted to obtain bounds for decidability results on the EF-emptiness problem. They aim to ask for the minimal assumptions over the model to ensure the decidability. For example, it becomes decidable if the PTA contains only one clock compared to parameters (called "parametric clock") and integer-valuated parameters, with arbitrarily many non-parametric clocks.

A survey of decidability results with PTAs was proposed by [And19b].

2.4.2 *Model checkers supporting the parametric timed automaton formalism*

Considering hybrid systems (a class going beyond PTAs), the first model checker was HyTech [HHWT95]. However, it is not maintained anymore. SpaceEx [Fre+11] is another tool for hybrid systems, not dedicated to parameter synthesis and mainly focused on safety and reachability properties.

IMITATOR [And21a] is a tool for PTA verification. It supports the PTA formalism, extended with many features such as multi-rate clocks, global variables and data structures. It implements the synthesis of parameters ensuring safety, reachability, liveness or robustness properties for example.

Finally, Roméo [LRST09] is a tool supporting parametric time PNs (a formalism close to PTAs, with a similar level of expressiveness). It allows the synthesis of parameters.

2.4.3 *Using parametric timed automata to ensure safety*

As for TAs, there have been many applications of model checking PTAs. In the following, we just cite a few as examples of applications; [And19b] presents a more global overview.

[ALRS21] investigates the translation of attack-fault trees, which allow to model how a safety or security property can be refined into smaller sub-goals, to (extensions of) PTAs. This is done in order to synthesize internal constants, as time costs, using IMITATOR to prevent attack to be successful. [PKPS19] considers reductions of attack-defense trees (ADTrees) and presents experiments using IMITATOR.

In [FJ13], PTAs are used for timing analysis of mixed music scores. [CAS01] considers a protocol on data transfer using timing constraints to elect a leader.

[CJL17] proposes a new generic algorithm to verify general real-timed systems, formalized in different formalisms including PTAs. The authors' method is based on trace abstraction, differing from classical computation: many tools (such as the model checkers presented before) rely on polyhedra libraries. Experiments are proposed with several tools, including IMITATOR, and show that this technique solves instances which are not solvable by state-of-the-art analysis tools.

Solutions using PTAs for the Formal Methods for Timing Verification Challenge (FMTV 2015) are proposed in [SAL15]. These challenges aim to compute maximum/minimum latencies on periodical systems. The authors give solutions to each challenge, providing only upper-bounds for the most difficult. IMITATOR is used for the parameter synthesis.

In [LGSBL19], the authors present a method based on (extensions) of TAs and PTAs to perform the verification product-line engineering tools. They define a notion of coverage with minimum and maximum delays, asking every location to be accessible within a given interval of time. Their method is implemented and evaluated using IMITATOR.

In [KLS20], the authors deal with the modeling and verification of a Thales consensus algorithm. This algorithm has an arbitrary number of processes, which can possibly fail and restart at any time; their communications are asynchronous and periodic. The proposed method is based on the modeling of the source code, as well as the execution interleavings.

[KSA22] proposes to model and verify smart applications in SmartThings using the formalism of PTAs. To this end, the authors present a practical framework, named PSA, to identify safety violations, with a translation of application programming interfaces to PTAs and toolkits to model the devices and their environment. The framework PSA is evaluated using the IMITATOR model checker on practical examples from the Samsung SmartThings IOT ecosystem.

In [ACFJL21], the scheduling of a flight control system of a space launcher is implemented into a (extended) PTA. This problem aims to decide which task is run by the processor at each moment, considering its importance and urgency. IMITATOR is used to perform experiments.

2.5 SECURITY PROBLEMS IN TIMED AUTOMATA

2.5.1 *Diagnosability*

The *diagnosis* of TAs is one of the dominant research directions aimed at analyzing information leakage from a safety perspective. Its goal is to detect, by observing the system, whether some faulty behavior occurred. As such, it is some form of dual to opacity. Diagnosis was first introduced for TAs in [Tri02]. Diagnosability of a system is shown there to be decidable, though the actual diagnoser may be quite complex ([BCD05] presents subclasses of TAs allowing simpler diagnoser). [CT13] presents a summary of the main results on the diagnosis of TAs.

2.5.2 *Non-interference*

The notion of opacity is also closely related to the line of work on defining and analyzing information flow in TAs. It is well-known (e. g., [Koc96; FS00; BB07; KPJJ13; BCLR15]) 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 *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). It allows to quantify the frequency of an attack; this can be seen as a measure of the strength of an attack, depending on the frequency of the admissible actions. Different equivalences lead to different flavors of non-interference. In [BDFST02; BT03], a first notion of *timed* non-interference is proposed for TAs. This notion is extended to PTAs in [AK20], with a semi-algorithm.

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. Furthermore, they show that the problem of determining whether a TA 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 [BKMR08]. Other equivalence relations (namely (timed) cosimulation, (timed) bisimulation, sets of states) are not as easily relatable to opacity. No implementation is provided in [GMR07].

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 TAs; however, we mainly focus on leakage as execution time, and we *synthesize* internal parts of the system (clock guards), in contrast to [NNV17] where the system is fixed.

In [VNN18], Vasilikos *et al.* define the security of TAs 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 [GSB18], 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 [LPY97] is used to obtain the answer.

In [BCLR15], Benattar *et al.* study the control synthesis problem of TAs for SNNI. That is, given a TA, 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 PTAs, i. e., timed systems with unknown design parameters, and focuses on synthesizing parameter valuations which guarantee information flow property. Compared to [BCLR15], 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.

A survey on security problems in TAs is proposed by [AA23].

position of our work To the best of our knowledge, our approach is the first work on parametric model checking for TAs for information flow property. We therefore present a notion of ET-opacity and exp-ET-opacity in Chapters 7 and 8, with decidability results on TAs and PTAs and a notion of control in Chapter 9. In addition, and in contrast to most of the aforementioned works, our approach comes with an implementation.

2.6 CONTROLLING REAL-TIME SYSTEMS

2.6.1 *Controlling systems*

In controller synthesis problems, the goal is not to verify the correctness of a system (according to a property), but to compute a *controller*, which restricts the choices of the system, to ensure the verification of the property. These problems are often modeled with 2-player games, with the synthesis of a winning strategy for the controller against the environment.

[FM15] deals with the runtime verification of opacity properties in LTSs. In this work, the goal is not only to observe a system, but to allow the monitor to modify the system behavior at runtime. Particularly, the authors introduce and study K-step opacity, with different levels, from simple opacity to a strong notion. Their techniques are implemented in a tool, TAKOS: this tool allows to model-check opacity and to synthesize runtime verifiers.

2.6.2 *Controller synthesis in timed systems*

Timed game automata (TGAs) [MPS95] are a common formalism to express 2-player games on TAs; they are extension of TAs allowing to reason over the moves of the two players, with controllable and uncontrollable actions. The problem of controller synthesis for reachability for TGAs is decidable [MPS95], an implementation is proposed in [CDFLL05; Beh+07], with case studies in [CJLRR09; JRLD07].

Controller synthesis is extended to probabilistic timed automata [Bea03] in [JKNP17] and to time PNs in [LLR23]; robustness is considered in [SBMR13].

[Bac+21] presents a controller synthesis over energy TAs (an extension of TAs where states have rates, or prices) with uncertainties; the uncertainty is here modeled by two functions, assigning imprecisions to rates of states to updates of transitions. The authors' approach consists in the translation of the controller problem into arithmetic expressions. Case studies are presented using a tool chain including MATHEMATICA v.11.2 [Inc17] and MJOLLNIR [Mon10].

2.6.3 *Controller synthesis in parametric systems*

The controller synthesis problem consists in synthesizing (or deciding the existence of) both the controller and the parameter valuations for which a given location of the system is reachable.

[JLR19] introduces the formalism of parametric timed game automata (PGAs) [JLR19], extending TGAs with parameters (on the same way as PTAs extend TAs). In this formalism, controller synthesis is undecidable (since the reachability problem is undecidable for PTAs). A subclass of TGAs, inspired by the class of lower/upper parametric timed automata (L/U-PTAs) [HRSV02], leads to decidable results. A semi-algorithm is also provided (without guarantee of termination) to solve controller synthesis in PGAs. In [JLR22], the same authors restrict PGAs to integer parameters. Algorithms are proposed and implemented in Roméo.

2.7 REDUCE THE STATE SPACE OF (POSSIBLY PARAMETRIC) TAs

2.7.1 *Extrapolation, simulations and inclusion*

Beyond merging (which we will study in this manuscript), various heuristics were proposed to efficiently reduce the state space of TAs.

EXTRAPOLATION AND SIMULATIONS Extrapolation and abstractions were proposed in [AD94; BBLP06; HSW13; HSW16] for TAs, and then extended to PTAs in [ALR15; BBBČ16; AA22]. These techniques aim to obtain a finite simulation of the state space of a model while preserving reachability properties. There are mainly based on the largest constants appearing in the constraints of a (P)TA and complex heuristics developed over years. Exploration orders were discussed in [HT15] for TAs and then in [ANP17] for PTAs. The efficiency of model checking liveness properties for TAs is discussed notably in [HSW12; HSTW20].

INCLUSION Zone inclusion (subsumption) for liveness checking is discussed for TAs in [LODLDP13] and for PTAs in [AAPP21]. We can consider inclusion/subsumption as a special case of merging.

OVER-APPROXIMATION In addition, computing efficiently exact or over-approximated successors of "zones" in the larger class of hybrid automata (HAs) [Hen96] is an active field of research (e. g., [CÁF11; CSÁ14; SNÁ17; BFFPS20]). Beyond the target formalism (PTAs instead of HAs), a main difference is that we are concerned here exclusively with an *exact* analysis.

These techniques are orthogonal to the merging technique, and they can be combined, except for inclusion as a particular case of merging.

2.7.2 *Merging zones*

Merging was first proposed for TAs in [Dav05], and then extended to the "inverse method" for PTAs in [AFS13].

In [BSBM06], Ben Salah *et al.* show that it is safe to perform the convex merging of various constraints, when they are the result of an interleaving. The exploration is done in a breadth-first search (BFS) manner, and states are merged at each depth level. In [BSBM09], the same authors propose a compositional verification framework, where merging is briefly mentioned to ensure exactness instead of an over-approximated approach based on the convex hull: it is mentioned that "preservation of the timed semantics by this step" is guaranteed whenever the authors "restrict the merging of transitions and states to those whose corresponding unions are convex" [BSBM09].

POSITION OF OUR WORK In Chapter 5, we propose to extend the constraint merging principle for reachability synthesis in PTAs. To perform efficient computation times, we propose and investigate different heuristic techniques.

2.8 BENCHMARK LIBRARIES FOR MODEL CHECKING

No library is really interested in a synthesis of timing parameters on TAs (including PTAs). We present here some libraries considering models close to those we study.

RTLib [SGQ16] is a library of real-time systems modeled as TAs. Contrary to our goal and proposed solution, it does not consider parametric models.

Two libraries for hybrid-systems benchmarks were proposed in [FI04; Che+15]. Despite being more expressive than PTAs in theory, these formalisms cannot be compared in practice: most of them do not refer to timing parameters. Moreover, these libraries only focus on reachability properties and non-parameterized benchmarks.

The PRISM benchmark suite [KNP12] collects probabilistic models and properties. Despite including some timing aspects, time is not the focus there.

The collection of Matlab/Simulink models [HAF15] focuses on timed model checking, but has no parametric extensions. Two of our benchmarks (accel and gear) originate from a translation of their models to (extensions of) PTAs.

The JANI specification [Bud+17] defines a representation of automata with quantitative extensions and variables; their syntax is supported by several verification tools. A library of JANI benchmarks is also provided; such benchmarks come from PRISM, Modest, Storm and FIG, and therefore cannot be applied to parameter synthesis for timed systems.

Also, a number of model checking competitions started in the last two decades, accumulating over the years a number of sets of benchmarks, such as the ARCH (Applied Verification for Continuous and Hybrid Systems) "friendly competition" [Fre+19; ARCH21], the PNs model checking contest (MCC) [Amp+19; MCC21], the MARS (Models for Formal Analysis of Real Systems) workshop repository [MARS21], or the WATERS workshop series [QV15].

In [And19a; And18], a first version of the PTA library was introduced, including academic benchmarks, industrial and toy case studies.

POSITION OF OUR LIBRARY Our library presented in Chapter 4 aims at providing benchmarks for *parameter* synthesis for (extensions of) TAs. Notably, we go beyond the TA syntax (offering some benchmarks with multi-rate clocks, stopwatches, timing parameters, additional global variables), while not offering the full power of HAs (differential equations, complex flows). To the best of our knowledge, no other set of benchmarks addresses specifically the *synthesis* of timing parameters.

2.9 PARAMETERIZED VERIFICATION

Parametrized verification aims to validate a model regardless of the value of a parameter. For example, in [BF13], the authors verify a network of identical probabilistic timed processes, where the parameter is the number of processes. It is therefore unknown and might be constant over time or might change, with disappearance or creation of processes. In [BF13], modeling is performed using probabilistic TAs and the authors study different (un)decidability results.

Parametrized verification of TAs is introduced in [AJ03]. The problem of checking a safety property is in general undecidable for TA networks [ADM04], but becomes decidable for TAs with one clock [AJ03].

[BFS14] proposes parametrized verification of a network with a parametrized number of entities where the communication topology can be reconfigured at any time and entities can change probabilistically. These problems are studied with reduction to 2-player games. [ADRST16] also studies parametrized verification with a network of processes whose protocol is defined by a TA, where the parametrization considers the initial configuration of the network.

In [ADFL19], the authors propose to use parametrized verification on PTAs. That is, they consider a combination of both timing parameters and a parametric network size.

A survey on parametrized verification is proposed in [AD16].

3 TIMED MODEL-CHECKING

When we were children, we used to think that
when we were grown-up we would no longer be vulnerable.
But to grow up is to accept vulnerability...
To be alive is to be vulnerable.

— Madeleine L'Engle

This chapter recalls the necessary concepts used in the subsequent chapters of this thesis.

3.1 PRELIMINARY DEFINITIONS

3.1.1 *Mathematical notations*

Through this thesis, we denote by

- $\mathbb{R}, \mathbb{Q}, \mathbb{Z}, \mathbb{N}$ the sets of reals, rationals, integers and non-negative integers, respectively;
- $\mathbb{R}_{\geq 0}, \mathbb{Q}_{\geq 0}$ the sets of non-negative reals and non-negative rationals, respectively.

Given $\alpha \in \mathbb{R}$, let $\lfloor \alpha \rfloor$ and $\mathrm{fr}(\alpha)$ denote respectively the integral part and the fractional part of α.

Given a finite set $\mathcal{S}$, the cardinality of $\mathcal{S}$ is denoted by $|\mathcal{S}|$.

3.1.2 *Finite-state automaton*

Automata are a kind of well-known model to study discrete transition systems and their behaviors. Deterministic finite-state automata (DFAs) are automata with finitely many states.

Definition 3.1 (Deterministic finite-state automaton)**.** A DFA is a tuple $\mathcal{FA}_d = (\Sigma, S, s_0, F, \delta)$, where
- Σ is a finite alphabet;
- S is a finite set of states;
- $s_0 \in S$ is the initial state;
- $F \subseteq S$ is the set of accepting states;
- $\delta : S \times \Sigma \to S$ is the transition relation, a partial function on $S \times \Sigma$.

3.1.3 *Labeled transition system*

LTSs were defined as simple models for systems, based on directed graphs with actions. They allow to describe how a system evolves with an action.

Definition 3.2 (Labeled transition system [Kel76])**.** An LTS is a quadruple $S = (Q_S, \Sigma_S, \rightarrow_S, q_{0,S})$ where

- Q_S is a finite or infinite set of states;
- Σ_S a finite or infinite set of actions;
- $\rightarrow_S \subseteq Q_S \times \Sigma_S \times Q_S$ a set of transitions or steps;
- $q_{0,S} \in Q_S$ is the initial state.

We also recall the notion of simulation [BK08] between two transition systems. Intuitively, this notion is a relation between two systems sharing the same behavior, in the sense that they are undistinguishable from the other by an observer.

Definition 3.3 (Simulation, bisimulation)**.** Let $S_1 = (Q_1, \Sigma_1, \rightarrow_1, q_{0,1})$, $S_2 = (Q_2, \Sigma_2, \rightarrow_2, q_{0,2})$ be two LTSs. Let $q_1 \in Q_1$ be a state of S_1, $q_2 \in Q_2$ of S_2.
A binary relation $\mathcal{R}$ is a *simulation* if

$$(q_1, q_2) \in \mathcal{R}$$

$$\text{implies}$$

$$\forall q_1' \in Q_1, \forall a \in \Sigma, q_1 \overset{a}{\rightarrow}_1 q_1' \implies \exists q_2' \in Q_2,\ q_2 \overset{a}{\rightarrow}_2 q_2' \text{ and } (q_1', q_2') \in \mathcal{R}.$$

If $\mathcal{R}$ is symmetric, we say that $\mathcal{R}$ is a bisimulation.

3.1.4 *Timed transition system*

Timed transition systems (TTSs) [HMP91] are introduced as an extension of transition systems, where the transitions have timing constraints.

Definition 3.4 (Timed transition system [HMP91])**.** A TTS is a tuple $\mathfrak{T} = (S, \Sigma, \rightarrow, s_0)$ where

- S is a set of states;
- Σ is a set of actions;
- s_0 is the initial state;
- $\rightarrow \subseteq S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ is a labeled transition relation with two kinds of transitions:
  - event transitions: (s, e, s') with $e \in \Sigma$ (denoted $s \overset{e}{\mapsto} s'$);
  - time transitions: (s, d, s') with $d \in \mathbb{R}_{\geq 0}$ (denoted $s \overset{d}{\mapsto} s'$).

3.2 CLOCKS, PARAMETERS AND CONSTRAINTS

3.2.1 *Clocks*

Clocks are real-valued variables that all evolve over time at the same rate. Through this thesis, we assume a set $\mathbb{X} = \{x_1, \dots, x_{|\mathbb{X}|}\}$ of *clocks*.

A *clock valuation* is a function $\mu : \mathbb{X} \rightarrow \mathbb{R}_{\geq 0}$, assigning a non-negative value to each clock.

We write $\vec{0}$ for the clock valuation assigning 0 to all clocks. Given a constant $d \in \mathbb{R}_{\geq 0}$, $\mu + d$ denotes the valuation s.t. $(\mu + d)(x) = \mu(x) + d$, for all $x \in \mathbb{X}$.

3.2.2 *Parameters*

A *(timing) parameter* is an unknown constant of a model. Throughout this thesis, we assume a set $\mathbb{P} = \{p_1, \dots, p_{|\mathbb{P}|}\}$ of *parameters*.

A *parameter valuation* v is a function $v : \mathbb{P} \rightarrow \mathbb{Q}_{\geq 0}$.

Remark 1. *We choose $\mathbb{Q}_{\geq 0}$ by consistency with most of the PTA literature, but also because, for classical PTAs, choosing $\mathbb{R}_{\geq 0}$ leads to undecidability [Mil00].*

■

3.2.3 *Constraint*

We assume $\bowtie \in \{<, \leq, =, \geq, >\}$.

A constraint $\mathbf{C}$ is a conjunction of inequalities over $\mathbb{X} \cup \mathbb{P}$ of the form $x \bowtie \sum_{1 \leq i \leq |\mathbb{P}|} \alpha_i p_i + d$, with $p_i \in \mathbb{P}$, and $\alpha_i, d \in \mathbb{Z}$. Given a constraint $\mathbf{C}$, we write $\mu \models v(\mathbf{C})$ if the expression obtained by replacing each clock x with $\mu(x)$ and each parameter p with $v(p)$ in $\mathbf{C}$ evaluates to true.

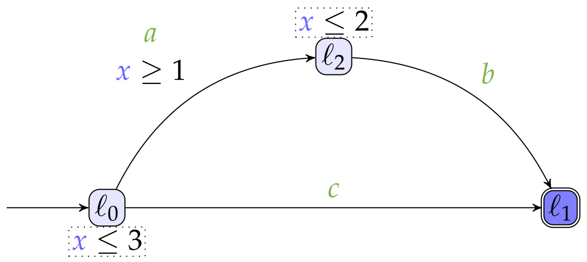


Figure 3.1: A TA example

3.3 Timed automaton

A TA is a finite automaton extended with a finite set of real-valued clocks [AD94].

Definition 3.5 (Timed automaton [AD94])**.** A TA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, \ell_f, \mathbb{X}, I, E)$, 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 final location,
5. $\mathbb{X}$ is a finite set of clocks,
6. I is the invariant, assigning to every $\ell \in L$ a constraint $I(\ell)$ over $\mathbb{X}$ (called *invariant*),
7. 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 constraint over $\mathbb{X}$ (called *guard*).

TAs are generally represented using directed graphs, where (*i*) locations are depicted by nodes, while their invariant are written near them and (*ii*) edges are depicted by vertices, labeled with the clock guard, the action and the set of clocks to be reset. Through this thesis, the initial location is designated with an incoming arrow, the final one by a double circle and the invariants are written in dashed boxes.

Example 3.1. *In Figure 3.1, we give an example of a TA with three locations ℓ_0, ℓ_1 and ℓ_2, three edges, with actions $\{a, b, c\}$, and one clock x. ℓ_0 has an invariant $x \leq 3$ and the edge from ℓ_0 to ℓ_2 has a guard $x \geq 1$.*
ℓ_0 is the initial location, while ℓ_1 is the (only) final *location.*

3.3.1 *Concrete semantics of timed automata*

We define the concrete semantics of a TA using a TTS.

Definition 3.6 (Semantics of a TA)**.** Given a TA $\mathcal{A} = \left(\Sigma, L, \ell_0, \ell_f, \mathbb{X}, I, E\right)$, the semantics of $\mathcal{A}$ is given by the TTS $\mathfrak{T}_{\mathcal{A}} = (\mathfrak{S}, \Sigma, \rightarrow, \mathfrak{s}_0)$, with

1. $\mathfrak{S} = \left\{(\ell, \mu) \in L \times \mathbb{R}_{\geq 0}^{|\mathbb{X}|} \mid \mu \models I(\ell)\right\}$,
2. $\mathfrak{s}_0 = (\ell_0, \vec{0})$,
3. $\rightarrow$ consists of the discrete and (continuous) delay transition relations:
   (a) discrete transitions: $(\ell, \mu) \overset{e}{\mapsto} (\ell', \mu')$, if $(\ell, \mu), (\ell', \mu') \in \mathfrak{S}$, and there exists $e = (\ell, g, a, R, \ell') \in E$, such that $\mu' = [\mu]_R$, and $\mu \models g$.
   (b) delay transitions: $(\ell, \mu) \overset{d}{\mapsto} (\ell, \mu + d)$, with $d \in \mathbb{R}_{\geq 0}$, if $\forall d' \in [0, d], (\ell, \mu + d') \in \mathfrak{S}$.

Moreover we write $(\ell, \mu) \xrightarrow{(d,e)} (\ell', \mu')$ for a combination of a delay and a discrete transition if there exists μ'' s.t. $(\ell, \mu) \overset{d}{\mapsto} (\ell, \mu'') \overset{e}{\mapsto} (\ell', \mu')$.

Given a TA $\mathcal{A}$ with concrete semantics $(\mathfrak{S}, \Sigma, \rightarrow, \mathfrak{s}_0)$, we refer to the states of $\mathfrak{S}$ as the *concrete states* of $\mathcal{A}$. A *run* of a TA $\mathcal{A}$ is an alternating sequence of concrete states of $\mathcal{A}$ and pairs of edges and delays starting from the initial state $\mathfrak{s}_0$ of the form $\mathfrak{s}_0, (d_0, e_0), \mathfrak{s}_1, \cdots, \mathfrak{s}_n$ with $e_i \in E$, $d_i \in \mathbb{R}_{\geq 0}$ and $\mathfrak{s}_i \xrightarrow{(d_i, e_i)} \mathfrak{s}_{i+1}$ for $i = 0, 1, \ldots, n-1$.

Definition 3.7 (Duration of a run)**.** Given a finite run $\rho : (\ell_0, \mu_0), (d_0, e_0), (\ell_1, \mu_1), \cdots, (d_{i-1}, e_{i-1}), (\ell_n, \mu_n)$, the *duration* of ρ is $dur(\rho) = \sum_{0 \leq i \leq n-1} d_i$. We also say that ℓ_n is reachable in time $dur(\rho)$.

Example 3.2. *Consider the TA $\mathcal{A}$ in Figure 3.1.*
Consider the following run ρ of $\mathcal{A}$: $(\ell_0, x = 0), (1.4, a), (\ell_2, x = 1.4), (0.4, b), (\ell_1, x = 1.8)$
Note that we write "$x = 1.4$" instead of "μ such that $\mu(x) = 1.4$".
We have $dur(\rho) = 1.4 + 0.4 = 1.8$.

3.3.2 *Timed automata regions*

In the following, we define an equivalence relation between the clocks of a TA. *Region equivalence* was first introduced in [AD94] and permits a kind of discretization of a TA: despite an infinite number of clock valuations, there is only a finite number of regions, while two clock valuations on the same regions have the same behavior.

Given a TA $\mathcal{A}$, for a clock x, we denote by $\mathcal{M}_x$ the largest constant to which x is compared within the guards and invariants of $\mathcal{A}$. That is, $\mathcal{M}_x = \max_i (\{ d_i \mid x \bowtie d_i \text{ appears in a guard or invariant of } \mathcal{A}\})$.

Example 3.3. *Consider again the TA defined in Figure 3.1. We have* $\mathcal{M}_x = \max\{1, 2, 3\} = 3$.

Region equivalence

Definition 3.8 (Region equivalence [AD94])**.** We say that two clock valuations μ and μ' are equivalent, denoted $\mu \approx \mu'$, if the following conditions hold for any clocks x_i, x_j:
either

1. (a) $\lfloor \mu(x_i) \rfloor = \lfloor \mu'(x_i) \rfloor$,
   (b) $\mathrm{fr}(\mu(x_i)) \leq \mathrm{fr}(\mu(x_j))$ *iff* $\mathrm{fr}(\mu'(x_i)) \leq \mathrm{fr}(\mu'(x_j))$, and
   (c) $\mathrm{fr}(\mu(x_i)) = 0$ *iff* $\mathrm{fr}(\mu'(x_i)) = 0$
   or
2. $\mu(x_i) > \mathcal{M}_{x_i}$ and $\mu'(x_i) > \mathcal{M}_{x_i}$

The equivalence relation $\approx$ is extended to the states of the semantics of $\mathcal{A}$: if $\mathfrak{s} = (\ell, \mu), \mathfrak{s}' = (\ell', \mu')$ are two states of $\mathfrak{T}_{\mathcal{A}}$, we write $\mathfrak{s} \approx \mathfrak{s}'$ *iff* $\ell = \ell'$ and $\mu \approx \mu'$.

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

Region graph

Definition 3.9 (Region graph [BDR08])**.** Let $\mathcal{A} = (\Sigma, L, \ell_0, \ell_f, \mathbb{X}, I, E)$ be a TA. The *region graph* $\mathcal{RG}_{\mathcal{A}} = (\mathcal{R}_{\mathcal{A}}, \mathcal{E}_{\mathcal{A}})$ is a finite graph with:

- $\mathcal{R}_{\mathcal{A}}$ as the set of vertices;
- given two regions $r = [\mathfrak{s}], r' = [\mathfrak{s}'] \in \mathcal{R}_{\mathcal{A}}$, we have $(r, r') \in \mathcal{E}_{\mathcal{A}}$ if one of the following holds:
  - $\mathfrak{s} \overset{e}{\mapsto} \mathfrak{s}' \in \mathfrak{T}_{\mathcal{A}}$ 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 $\mathfrak{s} + d \in r'$ and $\forall d' < d, \mathfrak{s} + d' \in r \cup r'$ (*delay transition*);
  - $r = r'$ is unbounded: $\mathfrak{s} = (\ell, \mu)$ with $\mu(x_i) > \mathcal{M}_{x_i}$ for all x_i (*equivalent unbounded regions*).

Region automaton

We now define a version of the region automaton based on [BDR08] where the only letter that can be read ("tick") means that one time unit has passed. Note that this automaton is not timed. As such, it is as usual described by a DFA.

We assume that the original TA $\mathcal{A}$ possesses a special location denoted ℓ_{priv} (as in Part II) and a special clock x_{tick} that is always reset every one time unit (through appropriate invariants and resets). This clock does not affect the behavior of the TA, but every time it is reset, we know that one unit of time passed. We also assume, as it will be done in Part II, that the TA is deadlocked once ℓ_f is reached (i. e., no transition can be taken and no time can elapse).

Definition 3.10 (Region automaton [BDR08]). The *region automaton* of a TA $\mathcal{A}$ is a DFA $\mathcal{RA}_{\mathcal{A}} = (\Sigma, S, s_0, F, \delta) = \{\{\mathsf{tick}\}, \mathcal{R}_{\mathcal{A}}, [\mathfrak{s}_0], F, T\}$ where

1. tick is the only action;
2. $\mathcal{R}_{\mathcal{A}}$ is the set of states (a state of $\mathcal{RA}_{\mathcal{A}}$ is a region of $\mathcal{A}$);
3. $[\mathfrak{s}_0]$ is the initial state (the region associated to the initial location of $\mathcal{A}$);
4. the set of final states F is the set of regions associated to the location ℓ_{priv} where x_{tick} is not equal to 1 (i. e., the set of regions $r = [(\ell_{priv}, \mu)]$ where $\mu(x_{\mathsf{tick}}) < 1$);
5. $(r, a, r') \in T$ *iff* $(r, r') \in \mathcal{E}_{\mathcal{A}}$ and $a = \mathsf{tick}$ if x_{tick} was reset in the discrete instantaneous transition corresponding to (r, r'), and $a = \varepsilon$ otherwise.

This definition is inspired on the construction presented in the proof of [BDR08, Proposition 5.3]. The main difference point is that [BDR08] presents a construction where the resulting automaton is determinized.

An important property of this automaton is that the word tick^k with $k \in \mathbb{N}$ is accepted by $\mathcal{RA}_{\mathcal{A}}$ *iff* there exists a run reaching the final location of $\mathcal{A}$ within $[k, k+1)$ time units.

3.4 Parametric timed automaton

A PTA is a TA extended with a finite set of parameters [AHV93].

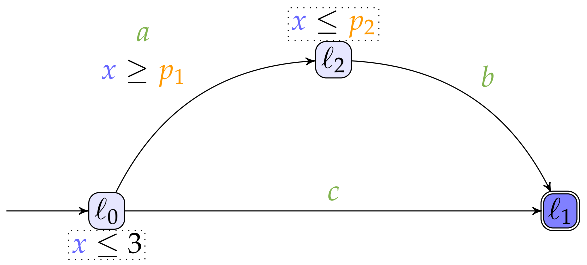


Figure 3.2: A PTA example

Definition 3.11 (Parametric timed automaton [AHV93]). A PTA $\mathcal{P}$ is a tuple $\mathcal{P} = (\Sigma, L, \ell_0, \ell_f, \mathbb{X}, \mathbb{P}, I, E)$, 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 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 constraint $I(\ell)$ over $\mathbb{X} \cup \mathbb{P}$ (called *invariant*);
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 constraint over $\mathbb{X} \cup \mathbb{P}$ (called *guard*).

Example 3.4. *In Figure 3.2, we give an example of a PTA with three locations ℓ_0, ℓ_1 and ℓ_2, three edges, with actions $\{a, b, c\}$, one clock x and two parameters $\{p_1, p_2\}$. ℓ_0 has an invariant $x \leq 3$ and the edge from ℓ_0 to ℓ_2 has a guard $x \geq p_1$.*
ℓ_0 is the initial location, while ℓ_1 is the (only) final *location.*

Definition 3.12 (Valuation of a PTA). Given a parameter valuation v, we denote by $v(\mathcal{P})$ the non-parametric structure where all occurrences of a parameter p_i have been replaced by $v(p_i)$.

Remark 2. *We have a direct correspondence between the valuation of a PTA and the definition of a TA given in Definition 3.5.*

TAs were originally defined with integer constants in [AD94] (as done in Definition 3.5). By assuming a rescaling of the constants (i. e., by multiplying all constants in a TA by the least common multiple of their denominators), we obtain an equivalent (integer-valued) TA, as defined in Definition 3.5.

■

Example 3.5. *Consider the PTA in Figure 3.2 and let v be such that $v(p_1) = 1$ and $v(p_2) = 2$. $v(\mathcal{P})$ is a TA, represented in Figure 3.1.*

3.4.1 *Synchronized product of parametric timed automata*

In the following, we recall the notion of *synchronous product* of a set of PTAs. This product (using strong broadcast, i. e., synchronization on a given set of actions), or *parallel composition*, of several PTAs gives a PTA.

Definition 3.13 (Synchronized product of PTAs)**.** Let $N \in \mathbb{N}$. Given a set of PTAs $\mathcal{P}_i = \big(\Sigma_i, L_i, (\ell_0)_i, (\ell_f)_i, \mathbb{X}_i, \mathbb{P}_i, I_i, E_i\big)$, $1 \leq i \leq N$, and a set of actions Σ_s, the *synchronized product* of $(\mathcal{P}_i)_{1 \leq i \leq N}$, denoted by $\mathcal{P}_1 \parallel_{\Sigma_s} \mathcal{P}_2 \parallel_{\Sigma_s} \cdots \parallel_{\Sigma_s} \mathcal{P}_N$, is the tuple $\big(\Sigma, L, \ell_0, \ell_f, \mathbb{X}, \mathbb{P}, I, E\big)$, where:

1. $\Sigma = \bigcup_{i=1}^{N} \Sigma_i$,
2. $L = \prod_{i=1}^{N} L_i$,
3. $\ell_0 = ((\ell_0)_1, \dots, (\ell_0)_N)$,
4. $\ell_f = ((\ell_f)_1, \dots, (\ell_f)_N)$,
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. $I((\ell_1, \dots, \ell_N)) = \bigwedge_{i=1}^{N} I_i(\ell_i)$ for all $(\ell_1, \dots, \ell_N) \in L$,
8. and E is defined as follows.

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

   - if $a \in \Sigma_s$, then
     (a) 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,
     (b) 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
     (a) 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,
     (b) for all $j \neq i$, $\ell'_j = \ell_j$.

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

3.4.2 *Symbolic semantics of parametric timed automaton*

We now define the symbolic semantics of PTAs [HRSV02; ACEF09; JLR15].

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

$$\mu' \models v(\mathbf{C}^{\nearrow}) \text{ if } \exists \mu : \mathbb{X} \to \mathbb{R}_{\geq 0}, \exists d \in \mathbb{R}_{\geq 0} \text{ s.t. } \mu \models v(\mathbf{C}) \wedge \mu' = \mu + d.$$

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

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

We denote by $\mathbf{C}{\downarrow_{\mathbb{P}}}$ the projection of $\mathbf{C}$ onto $\mathbb{P}$, i. e., obtained by eliminating the variables not in $\mathbb{P}$ (e. g., using Fourier-Motzkin [Sch99]).

The application of one of these operations (time elapsing, reset, projection) to a constraint yields a constraint; existential quantification can be handled, e. g., by adding variables and subsequently eliminating them using, e. g., Fourier-Motzkin.

Definition 3.14 (Symbolic state)**.** A symbolic state is a pair $\mathbf{s} = (\ell, \mathbf{C})$ where $\ell \in L$ is a location, and $\mathbf{C}$ its associated constraint over $\mathbb{X} \cup \mathbb{P}$ called *parametric zone*.

Definition 3.15 (Symbolic semantics)**.** Given a PTA $\mathcal{P} = (\Sigma, L, \ell_0, \ell_f, \mathbb{X}, \mathbb{P}, I, E)$, the symbolic semantics of $\mathcal{P}$ is the LTS called parametric zone graph (PZG) $\mathbf{PZG}(\mathcal{P}) = (E, \mathbf{S}, \mathbf{s}_0, \Rightarrow)$, with

- $\mathbf{S} = \{(\ell, \mathbf{C}) \mid \mathbf{C} \subseteq I(\ell)\} \in L \times \mathbb{R}_{\geq 0}^{|\mathbb{X}|}$
- $\mathbf{s}_0 = (\ell_0, (\bigwedge_{1 \leq i \leq |\mathbb{X}|} x_i = 0)^{\nearrow} \wedge I(\ell_0))$, and
- $((\ell, \mathbf{C}), e, (\ell', \mathbf{C}')) \in \Rightarrow$ if
  - $e = (\ell, g, a, R, \ell') \in E$ and
  - $\mathbf{C}' = ([(\mathbf{C} \wedge g)]_R \wedge I(\ell'))^{\nearrow} \wedge I(\ell')$ with $\mathbf{C}'$ satisfiable.

Example 3.6. *Consider again the PTA $\mathcal{P}$ of Figure 3.2.* $\mathbf{PZG}(\mathcal{P})$ *is represented in Figure 3.3.*

That is, in the PZG, nodes are symbolic states, and arcs are labeled by *edges* of the original PTA. Given a symbolic state $\mathbf{s}$ reachable

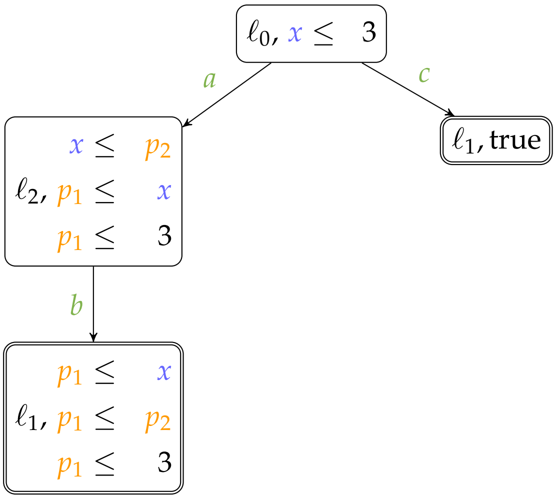


Figure 3.3: PZG of PTA of Figure 3.2

in $\mathbf{PZG}(\mathcal{P})$, we define the successors with edges, denoted $\mathsf{SuccE}(\mathbf{s})$, by $\{(e, \mathbf{s}') \mid (\mathbf{s}, e, \mathbf{s}') \in \Rightarrow\}$.

We also write $\mathbf{s} \Rightarrow \mathbf{s}'$ to denote that for some e, $(\mathbf{s}, e, \mathbf{s}') \in \Rightarrow$. Given $\mathbf{t} = (\mathbf{s}, e, \mathbf{s}') \in \Rightarrow$, $\mathbf{t}.\mathit{source}$ denotes $\mathbf{s}$ while $\mathbf{t}.\mathit{target}$ denotes $\mathbf{s}'$. Given $\mathbf{s} = (\ell, \mathbf{C})$, $\mathbf{s}.\mathit{constr}$ denotes $\mathbf{C}$ while $\mathbf{s}.\mathit{loc}$ denotes ℓ. Note that we usually use bold font to denote anything symbolic, i. e., (sets of) symbolic states, and constraints.

A well-known result [HRSV02] is that, given a PTA $\mathcal{P}$ and a reachable symbolic state $(\ell, \mathbf{C})$, if a parameter valuation v belongs to the projection onto the parameters of $\mathbf{C}$ (i. e., $v \in \mathbf{C}{\downarrow_{\mathbb{P}}}$), then ℓ is reachable in the TA $v(\mathcal{P})$. This justifies Definition 3.12.

The *(symbolic) state space* of a PTA is its PZG. This structure is in general infinite, due to the intrinsic undecidability of most decision problems for PTAs. However, for semi-algorithms for parameter synthesis (without a guarantee of termination), it is of utmost importance to *reduce* the size of this state space, so as to perform synthesis more efficiently. We will study and present a contribution on the reduction of this state space in Part I.

3.4.3 *Reachability synthesis*

We use reachability synthesis to solve the problems defined in Section 7.1. In the following, we define the reachability problem and recall the procedure to solve reachability synthesis.

The reachability problem

We say that a set of locations L_{target} is reachable in a TA if there exists a location $\ell_{target} \in L_{target}$ which is reachable.

We define the reachability synthesis problem. Given a PTA $\mathcal{P}$ and a set of locations L_{target}, this problems ask to synthesize the (maximal) set of parameter valuations v such that L_{target} is reachable in $v(\mathcal{P})$.

The reachability synthesis problem:
INPUT: A PTA $\mathcal{P}$, a set of target locations L_{target}
PROBLEM: Synthesize the (maximal) set of parameter valuations v such that L_{target} is reachable in $v(\mathcal{P})$.

The synthesis procedure

This procedure, called EFsynth and denoted $\mathsf{EFsynth}(\mathcal{P}, L_{target})$ 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 PZG of $\mathcal{P}$.

Example 3.7. *Consider again the PTA $\mathcal{P}$ in Figure 3.2.* $\mathsf{EFsynth}(\mathcal{P}, \{\ell_2\}) = p_1 \leq 3 \land p_1 \leq p_2$. *Intuitively, it corresponds to the disjunction of all parameter constraints in the PZG in Figure 3.3 associated to symbolic states with location ℓ_2.*

We finally recall the correctness of EFsynth.

Lemma 3.1 (Correctness of EFsynth [JLR15])**.** *Let $\mathcal{P}$ be a PTA, and let L_{target} be a subset of the locations of $\mathcal{P}$. Assume* $\mathsf{EFsynth}(\mathcal{P}, L_{target})$ *terminates with result K. Then $v \models K$ iff L_{target} is reachable in $v(\mathcal{P})$.*

3.5 LOWER/UPPER PARAMETRIC TIMED AUTOMATON

L/U-PTAs is the most well-known subclass of PTAs with some decidability results: for example, reachability emptiness ("the emptiness of the valuations set for which a given location is reachable"), which is undecidable for PTAs, becomes decidable for L/U-PTAs [HRSV02]. Various other results were studied (e. g., [BL09; JLR15; ALR22]).

Definition 3.16 (Lower/upper parametric timed automaton [HRSV02])**.** 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 $x \bowtie \sum_{1 \leq i \leq |\mathbb{P}|} \alpha_i p_i + d$, 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 3.8. *The PTA in Figure 3.2 is an L/U-PTA with $\{p_1\}$ as lower-bound parameter, and $\{p_2\}$ as upper-bound parameter.*

3.5.1 *Monotonicity of lower/upper parametric timed automata*

We have the following monotonicity property of L/U-PTAs:

Lemma 3.2 (L/U-PTA monotonicity [HRSV02])**.** *Let $\mathcal{P}_{\mathcal{LU}}$ 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 $v(\mathcal{P})$ is a run of $v'(\mathcal{P})$.*

3.6 IMITATOR MODEL CHECKER

IMITATOR [And21a] is a software tool for parametric verification of (extensions of) PTAs. It takes as input networks of PTAs extended with several handful features such as synchronization though strong broadcast, rational-valued global variables, stopwatches, multi-rate clocks, and some other useful features. It implements several algorithms, including parametric reachability or safety analysis ("EF-synthesis") [AHV93; JLR15]. IMITATOR is able to verify a large number of case studies from the literature and industry.

It is fully written in OCaml, and makes use of Parma Polyhedra Library (PPL) [BHZ08]. It is open-source and free, available under the GNU General Public License.

Part I

ZONE MERGING IN PARAMETRIC TIMED AUTOMATA

Eu não sei, de onde vem, essa força que me leva pra você
Eu só sei que faz bem, mas confesso que no fundo eu duvidei
Tive medo, e em segredo, guardei o sentimento e me sufoquei
Mas agora, é a hora, eu vou gritar pra todo mundo de uma vez

Eu tô apaixonado
Eu tô contando tudo e não tô nem ligando pro que vão dizer
Amar não é pecado
E se eu tiver errado, que se dane o mundo, eu só quero você

Eu tô apaixonado
Eu tô contando tudo e não tô nem ligando pro que vão dizer
Amar não é pecado
E se eu tiver errado, que se dane o mundo, eu só quero você

— Luan Santana, Amar não é pecado

4

A BENCHMARK LIBRARY FOR EXTENDED PARAMETRIC TIMED AUTOMATA

Fais de ta vie un rêve,
et d'un rêve, une réalité.

— Antoine De Saint-Exupéry

Before proposing new heuristics to improve the efficiency of the verification of PTA models, we need to gather a structured set of benchmarks as a fair comparison basis.

To his end, in this chapter, we present an extension of the IMITATOR benchmark library, that accumulated over the years a number of case studies from academic and industrial contexts. We extend here the library with several dozens of new benchmarks; these benchmarks highlight several new features.

Motivation

In the past few years, a growing number of new synthesis algorithms were proposed for PTAs, e. g., using bounded model-checking [KP12], compositional verification [ABBCR16; AL17b], distributed verification [ACN15], for liveness properties [BBBČ16; NPP18; AAPP21], for dedicated problems [CPR08]—notably for testing timed systems [FK13; And16; LSBL17; AAGR19; LGSBL19; AAPP21]. However, these works consider different benchmarks sets, making it difficult to evaluate which technique is the most efficient for each application domain. A benchmark suite for (extended) PTAs can be used for different purposes:

1. when developing new algorithms for (extensions of) PTAs and testing their efficiency by comparing them with existing techniques;

2. when evaluating benchmarks for (extensions of) TAs (note, as introduced in Definition 3.12, that valuating our benchmarks with a parameter valuation yields a TA or a multi-rate automaton (MRA) [Alu+95]); and

3. when looking for benchmarks fitting in the larger class of HAs.

Contributions of the chapter

In [And19a; And18], the author introduced a first library of 34 benchmarks, 80 models and 122 properties for PTAs.

However, this former version of the library suffers from several issues. First, its syntax is only compatible with the syntax of version 2.12 of IMITATOR [AFKS12], while IMITATOR shifted to version 3.0 [And21a] in 2021, with a different calling paradigm.[1] Second, the former version contains exclusively safety/reachability properties (plus some "robustness" computations, using the IM algorithm, also called trace preservation synthesis (TPS)). Third, only syntactic information is provided (benchmarks, metrics on the benchmarks), and no semantic information (expected result, approximate computation time, and approximate number of states to explore).

In this chapter, we extend the former library with a list of new features, including syntactic extensions (notably multi-rate clocks [Alu+95]); we also focus on *unsolvable* case studies, i. e., simple examples for which no known algorithm allows computation of the result, with the ultimate goal to encourage the community to address these cases. In addition, we add *liveness* properties, i. e., cycle synthesis. Also, we add *semantic* criteria, with an approximate computation time for the properties, an expected result (whenever available) and an approximate number of explored symbolic states (computed using IMITATOR). The rationale is to help users by giving them an idea of what to expect for each case study. Also, our consolidated classification aims at helping tool developers to select within our library which benchmarks suit them (e. g., "PTAs without stopwatches, with many locations and a large state space").

To summarize, we propose a new version of our library enhancing the former one as follows:

1. adding 22 new benchmarks (and 39 models, constituting or not these new benchmarks)
   - adding benchmarks for *liveness* properties;
   - adding a set of toy *unsolvable* benchmarks, to emphasize the limits of state-of-the-art parametric verification techniques, and to encourage the community to develop new dedicated algorithms in the future;

[1] While many keywords remain the same in the model, the property syntax has been completely rewritten, and the model checker now takes as input a model file *and* a property file. In addition, new properties are now possible, and the syntax has been extended with some useful features such as multi-rate clocks or "if-then-else" control structures.

Table 4.1: Selected new features

Library	Size			Metrics		Format		Categories	Properties			Analysis
Version	Bench.	Models	Prop.	Static	Semantic	.imi	JANI	Unsolvable	EF	IM	liveness	Results
1.0 [And18]	34	80	122	√	×	2.12	×	×	√	√	×	×
2.0 [AMP21c]	56	119	216	√	√	3.0	√	√	√	√	√	√

2. refactoring all existing benchmarks, so that they now implement the syntax of the 3.0 version of IMITATOR;
3. providing a better classification of benchmarks;
4. highlighting extensions of PTAs, such as multi-rate clocks [Alu+95] and stopwatches [CL00];
5. offering an automated translation of our benchmarks to the JANI [Bud+17; JANI17] model interchange format, offering a unified format for quantitative automata-based formalisms. This way, the library can be used by any tool using JANI as an input format, and supporting (extensions of) TAs. Even though other tools implementing the JANI formalism do not handle parameters, they can run on *instances* of our benchmarks, i. e., by valuating the PTAs with concrete valuations of the parameters.

We summarize the most significant dimensions of our extension in Table 4.1. EF (using the TCTL syntax) denotes reachability/safety, and IM denotes robustness analysis. We denote with a green cell the presence of a feature, and with a red cell its absence.

Organization of the chapter

We briefly introduce the IMITATOR extension of PTAs syntax in Section 4.1. We present our library in Section 4.2, and conclude in Section 4.3.

4.1 EXTENDING THE PARAMETRIC TIMED AUTOMATON SYNTAX

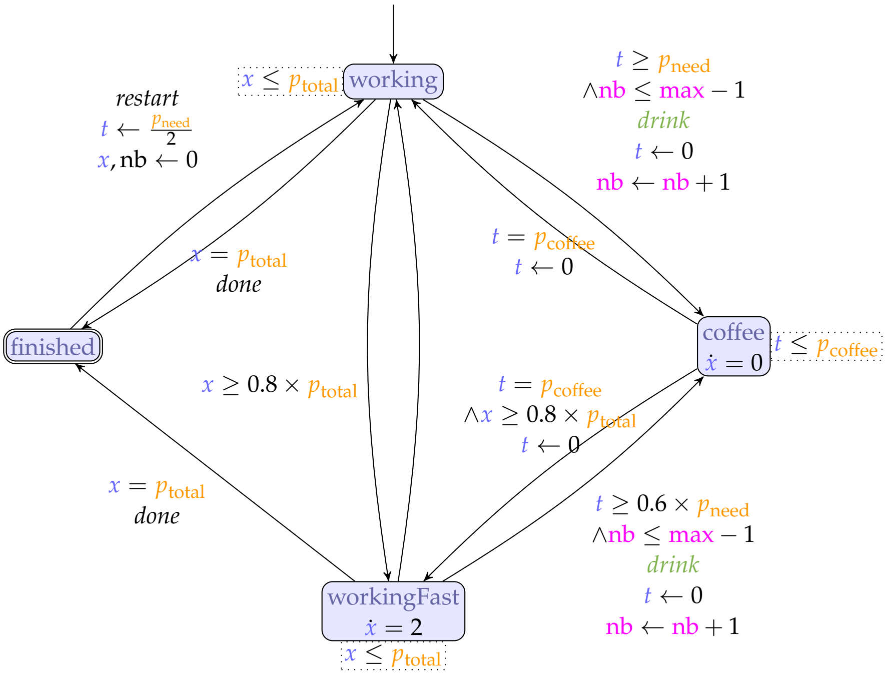
Our library follows the IMITATOR syntax. Therefore, some benchmarks (clearly marked as such) go beyond the traditional PTA syntax, and are referred to IMITATOR parametric timed automata (IPTAs) [And21a]. These extensions include:

URGENT LOCATIONS They are locations where time cannot elapse.

GLOBAL RATIONAL-VALUED VARIABLES Global variables (called "discrete") can be defined, and are part of the discrete part of a state, together with locations (and different from clocks and parameters that are part of the *continuous* part). Global variables in IMITATOR are exact rationals, following exact arithmetics (as opposed to, e. g., floating-point arithmetic that can accumulate errors and lead to faulty assertions). Exact rationals are encoded in IMITATOR using the GNU MP library. Non-linear arithmetic expressions over sole discrete variables are allowed too.

ARBITRARY FLOWS Some benchmarks require arbitrary (constant) flows for clocks; this way, clocks do not necessary evolve at the same time, and can encode different concepts from only time, e. g., temperature, amount of completion, continuous cost. Their value can increase or decrease at any predefined rate in each location, and can become negative. In that sense, these clocks are closer to *continuous variables* (as in HAs) rather than TAs' clocks; nevertheless, they still have a constant flow, while HAs can have more general flows. This makes some of our benchmarks fit into a parametric extension of MRAs. This notably includes stopwatches, where clocks can have a 1 or 0-rate [CL00]. In Figure 4.1, x has rate 2 in workingFast, and is stopped in coffee (rate 0, or stopwatch).

ADDITIONAL SYNTAX IMPROVEMENTS Beyond the aforementioned increase of the syntactic expressive power, the syntax was enhanced, for example with accepting locations (that can be used in properties), global constants, "if. . . then. . . else" conditions in updates.



$x \leq p_{\text{total}}$
working
restart
$t \leftarrow \frac{p_{\text{need}}}{2}$
$x, \text{nb} \leftarrow 0$
$t \geq p_{\text{need}}$
$\wedge \text{nb} \leq \text{max} - 1$
drink
$t \leftarrow 0$
$\text{nb} \leftarrow \text{nb} + 1$
$x = p_{\text{total}}$
done
$t = p_{\text{coffee}}$
$t \leftarrow 0$
finished
coffee
$\dot{x} = 0$
$t \leq p_{\text{coffee}}$
$x \geq 0.8 \times p_{\text{total}}$
$t = p_{\text{coffee}}$
$\wedge x \geq 0.8 \times p_{\text{total}}$
$t \leftarrow 0$
$x = p_{\text{total}}$
done
$t \geq 0.6 \times p_{\text{need}}$
$\wedge \text{nb} \leq \text{max} - 1$
drink
$t \leftarrow 0$
$\text{nb} \leftarrow \text{nb} + 1$
workingFast
$\dot{x} = 2$
$x \leq p_{\text{total}}$

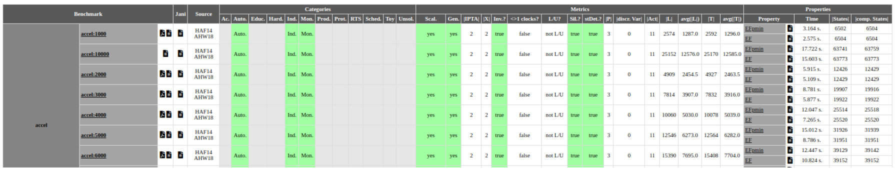
Figure 4.1: A IPTA example: Writing papers and drinking coffee

Example 4.1. *This case study* researcher *is part of the version 2 of our library, in categories "academic", "toy" and "teaching" (see our classification in Section 4.2.3).*
The model features two clocks t (measuring the time when needing a coffee) and x (measuring the amount of work done on a given paper), both initially 0. Their rate is always 1, unless otherwise specified (e. g., in workingFast*). Initially, the researcher is working (location* working*) on a paper, requiring an amount of work p_{total}. When the paper is completed (guard $x = p_{\text{total}}$), the IPTA moves to location* finished*. From there, at any time, the researcher can start working on a new paper (transition back to location* working*, updating x and t). Alternatively, after at least a certain time (guard $t \geq p_{\text{need}}$), the researcher may need a coffee; this action can only be taken until a maximum number of coffees have been drunk for this paper ($\mathrm{nb} \leq \mathrm{max} - 1$), where* nb *is a discrete global variable recording the number of coffees drunk while working on the current paper. When drinking a coffee (location* coffee*), the work is obviously not progressing ($\dot{x} = 0$). Drinking a coffee takes exactly p_{coffee} time units (guard $t = p_{\text{coffee}}$ back to location* working*). Observe that, from the second paper onward (transition labeled with restart), the researcher is already half-way of her/his need for a coffee (parametric update $t \leftarrow 0.5 \times p_{\text{need}}$ [ALR19]).*
Also, whenever 80 % of the work is done (guard $x \geq 0.8 \times p_{\text{total}}$), the researcher may work twice as fast (location workingFast*, with a rate 2 for clock x). In that case, (s)he needs a coffee faster too ($0.6 \times p_{\text{need}}$).*
All three durations p_{coffee}, p_{need} and p_{total} are timing parameters. We fix their parameter domains as follows: $p_{\text{coffee}}, p_{\text{total}} \in [0, \infty)$ and $p_{\text{need}} \in [1, \infty)$. The maximum number of coffees max $\in [0, \infty)$ *is also a parameter; observe that it is (only) compared to the discrete variable* nb*, which is allowed by the liberal syntax of IMITATOR.*

4.2 THE BENCHMARK LIBRARY

4.2.1 *Organization*

We decided the following organization for our new version of the library. The library is made of a set of *benchmarks*. Each benchmark may have different *models*: for example, Gear comes with ten models, of different sizes (the number of locations notably varies), named Gear-1000 to Gear-10000. Similarly, some Fischer benchmarks come with several models, each of them corresponding to a different number of processes. Finally, each model comes with one or more *properties*. For example, for Gear-2000, one can run either reachability synthesis,



Benchmark			Jani	Source	Categories												Metrics																Properties				
					Ac.	Auto.	Educ.	Hard.	Ind.	Mon.	Prod.	Prot.	RTS	Sched.	Toy	Unsol.	Scal.	Gen.	|IPTA|	|X|	Inv.?	<>1 clocks?	L/U?	Sil.?	stDet.?	|P|	|discr. Var|	|Act|	|L|	avg(|L|)	|T|	avg(|T|)	Property		Time	|States|	|comp. States|
accel	accel:1000			HAF14 AHW18		Auto.			Ind.	Mon.							yes	yes	2	2	true	false	not L/U	true	true	3	0	11	2574	1287.0	2592	1296.0	EFpmin		3.164 s.	6502	6504
																																	EF		2.575 s.	6504	6504
	accel:10000			HAF14 AHW18		Auto.			Ind.	Mon.							yes	yes	2	2	true	false	not L/U	true	true	3	0	11	25152	12576.0	25170	12585.0	EFpmin		17.722 s.	63741	63759
																																	EF		15.603 s.	63773	63773
	accel:2000			HAF14 AHW18		Auto.			Ind.	Mon.							yes	yes	2	2	true	false	not L/U	true	true	3	0	11	4909	2454.5	4927	2463.5	EFpmin		5.915 s.	12426	12429
																																	EF		5.109 s.	12429	12429
	accel:3000			HAF14 AHW18		Auto.			Ind.	Mon.							yes	yes	2	2	true	false	not L/U	true	true	3	0	11	7814	3907.0	7832	3916.0	EFpmin		8.781 s.	19907	19916
																																	EF		5.877 s.	19922	19922
	accel:4000			HAF14 AHW18		Auto.			Ind.	Mon.							yes	yes	2	2	true	false	not L/U	true	true	3	0	11	10060	5030.0	10078	5039.0	EFpmin		12.047 s.	25514	25518
																																	EF		7.265 s.	25520	25520
	accel:5000			HAF14 AHW18		Auto.			Ind.	Mon.							yes	yes	2	2	true	false	not L/U	true	true	3	0	11	12546	6273.0	12564	6282.0	EFpmin		15.012 s.	31926	31939
																																	EF		8.786 s.	31951	31951
	accel:6000			HAF14 AHW18		Auto.			Ind.	Mon.							yes	yes	2	2	true	false	not L/U	true	true	3	0	11	15390	7695.0	15408	7704.0	EFpmin		12.447 s.	39129	39142
																																	EF		10.824 s.	39152	39152

Figure 4.2: The IMITATOR benchmark library Web page: https://www.imitator.fr/library2/

or minimal reachability synthesis ("for which parameter valuations is the final location reachable in minimal time?").

The benchmark library, in its 2.0 version, covers 56 benchmarks, which group 119 models and 216 properties

From the previous version [And19a], 39 models have been added: beyond all Unsolvable models, and a few more additions, we notably added a second model of the Bounded Retransmission Protocol ("BRPAAPP21"), proposed in [AAPP21].

Benchmarks come from industrial collaborations (e. g., with Thales, ST-Microelectronics, ArianeGroup, Astrium), from academic papers from different communities (e. g., real-time systems, monitoring, testing) describing case studies, and from our experience in the field (notably the Unsolvable benchmarks). For benchmarks extracted from published works, a complete bibliographic reference is given.

4.2.2 *Distribution*

The benchmark library is presented on a Web page available at [AMP21c] and permanently available at [AMP21b]. Several columns (metrics, syntax used, categories, properties) allow users to select easily which benchmarks fit their need (see Figure 4.2).

Our benchmarks are distributed in the well-documented IMITATOR 3.0.0 input format [And21b], which is a *de facto* standard for PTAs. IMITATOR can provide automated translations to the non-parametric timed model checker Uppaal [LPY97], as well as the hybrid systems model checker HyTech [HHWT95] (not maintained anymore). However, some differences (presence of timing parameters or complex guards in IMITATOR, difference in the semantics of the synchronization model) may not preserve the semantic equivalence of the models.

In addition, we offer all benchmarks in the JANI format [Bud+17]. I implemented to this end (within IMITATOR) an automatic translation of IPTAs to their JANI specification. Thus, all of our benchmarks can be fed to other verification tools supporting JANI as input.

All our benchmarks are released under the CC by 4.0 license.

Table 4.2: Proportion of each category over the models

Category	Number of models	Proportion
All	119	100 %
Academic	54	45 %
Automotive	20	17 %
Education	9	8 %
Hardware	6	5 %
Industrial	33	28 %
Monitoring	25	21 %
ProdCons	5	4 %
Protocol	34	29 %
RTS	46	39 %
Scheduling	3	3 %
Toy	34	29 %
Unsolvable	18	15 %

4.2.3 *Benchmarks classification*

For each benchmark, we provide multiple criteria, notably the following ones.

SCALABILITY whether the models can be scaled according to some metrics, e. g., the FischerPS08 benchmark can be scaled according to the number of processes competing for the critical section;

GENERATION METHOD whether the models are automatically generated or not (e. g., by a script, notably for scheduling real-time systems using PTAs, or to generate random words in benchmarks from the testing or monitoring communities);

CATEGORIZATION Benchmarks are tagged with one or more categories: (*i*) Academic, (*ii*) Automotive, (*iii*) Education, (*iv*) Hardware, (*v*) Industrial, (*vi*) Monitoring, (*vii*) Producer-consumer, (*viii*) Protocol, (*ix*) Real-time system, (*x*) Scheduling, (*xi*) Toy, (*xii*) Unsolvable. The proportion of each of these tags are given in Table 4.2 (note that the sum exceeds 100 % since benchmarks can belong to multiple categories).

Moreover, we use the following static metrics to categorize our benchmarks:

- the numbers of IPTA components (subject to parallel composition), of clocks, parameters, discrete variables and actions;
- whether the benchmark has invariants, whether some clocks have a rate not equal to 1 (multi-rate/stopwatch) and silent actions (“ϵ-transitions”);
- whether the benchmark is an L/U-PTA and strongly deterministic;

Table 4.3: Statistics on the benchmarks

Metric	Average	Median
Number of IPTAs	3	3
Number of clocks	4	3
Number of parameters	4	3
Number of discrete variables	4	2
Number of actions	12	11
Total number of locations	2004	22
Locations per IPTA	979	5
Total number of transitions	2280	54
Transitions per IPTA	1067	13

Metric	Percentage
Has invariants?	92 %
Has discrete variables?	24 %
Has multi-rate clocks	17 %
L/U subclass	19 %
Has silent actions?	67 %
Strongly deterministic?	78 %

- the numbers of locations and transitions, and the total number of transitions.

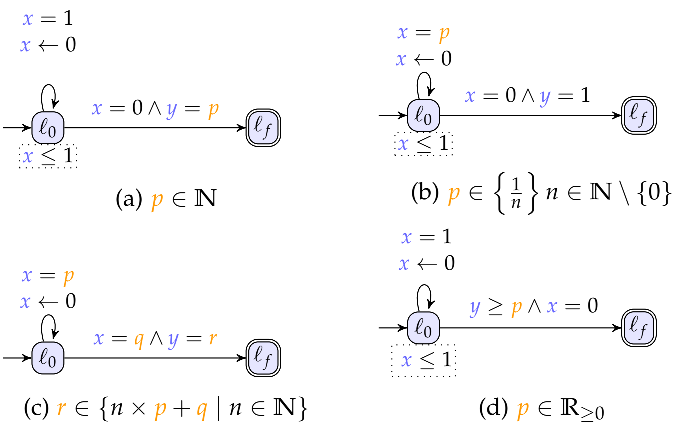
In Table 4.3, we present some statistics on our benchmarks. Because of the presence of 3 benchmarks and 25 models (all in the "monitoring" category) with a very large number of locations (up to several dozens of thousands), only giving the average of some metrics is irrelevant. To this end, we also provide the *median* values. Moreover, the average and the median of the number of discrete variables are computed only on the benchmarks which contain at least one such variable; they represent 24% of our models.

4.2.4 *Properties*

Properties follow the IMITATOR syntax. In the 1.0 version, they mainly consisted of reachability and safety properties; in addition, the properties were not explicitly provided, since IMITATOR 2.x did not specify properties (they were provided using options in the executed command).

In the new version of our library, we added several *liveness* (cycle synthesis) properties, i. e., for which one aims at synthesizing parameter valuations featuring at least one infinite (accepting) run [NPP18; AAPP21]; in addition, we added properties such as deadlock-freeness synthesis ("exhibit parameter valuations for which the model is deadlock-free") [And16], optimal-parameter or minimal-time reachability [ABPP19], and some "pattern"-based properties [And13] that eventually reduce to reachability checking [ABBL03].

More in details, we provide and study eight classes of properties:



(a) $p \in \mathbb{N}$

(b) $p \in \left\{\frac{1}{n}\right\} n \in \mathbb{N} \setminus \{0\}$

(c) $r \in \{n \times p + q \mid n \in \mathbb{N}\}$

(d) $p \in \mathbb{R}_{\geq 0}$

Figure 4.3: Examples of unsolvable benchmarks

1. Safety (AGnot) to obtain the set of valuations for which the target location is unreachable.

2. Cycle (resp. CycleThrough) to synthesize a parameter constraint such that, for any parameter valuation in that constraint, the system contains at least one infinite run [NPP18; AAPP21] (resp. passing infinitely often through the specified location).

3. DeadlockFree which synthesizes a parameter constraint such that, for any parameter valuation in that constraint, the system is deadlock-free [And16].

4. Reachability (EF) to compute the set of parameter valuations for which some location is reachable.

5. EFpmin (resp. EFpmax) to synthesize the minimum (resp. maximum) valuation for a given parameter for which a given location is reachable [ABPP19].

6. EFtmin to synthesize the parameter valuation for which a given location is reachable in minimal time [ABPP19].

7. Inverse method (IM) to compute a parameter constraint such that, for any parameter valuation in that constraint, the set of traces is the same as for the reference valuation [ALM20].

8. Pattern [And13], i. e., a set of predefined properties reducing to reachability or safety synthesis.

4.2.5 *Unsolvable Benchmarks*

A novelty of our library is to provide a set of toy unsolvable benchmarks. They have been chosen for being beyond the limits of the state-of-the-art techniques. Four of them are illustrated in Figure 4.3. For example, in Figure 4.3a, the reachability of ℓ_f is achievable only

Table 4.4: Statistics on executions (over 157 properties)

Metric	Average	Median
Total computation time (in seconds)	245.8	2.819
Number of states	20817.8	580
Number of computed states	34571.7	1089

if $p \in \mathbb{N}$; but no verification tool—to the best of our knowledge—terminates this computation. Moreover, the final location of the PTA presented in Figure 4.3d is reachable for all $p \geq 0$, which is a convex constraint, but this solution remains not computable.

4.2.6 *Expected Performances*

Another novelty of the 2.0 version is to provide users with all the expected results, as generated by IMITATOR. For all properties, we provide either a computed result, or (for the "unsolvable" benchmarks), a human-solved theoretical result.

We also give an approximate computation time, and the number of (symbolic) states explored. These metrics are not strict, as they may depend on the target model checker and the target hardware/OS, but this provides the user an idea of the complexity of our models.

If IMITATOR is able to compute the result of a property application, we provide its output file and extract some relevant features to have a general idea of its performances: the total computation time, the number of states and the number of computed states. The executions were made on an Intel Xeon Gold 5220 CPU @ 2.20 GHz with 96 GiB running Linux Ubuntu 20 ("gros" cluster of the Grid'5000 testbed [Gri]). If IMITATOR is not able to compute it—recall that the concerned benchmarks are tagged with "Unsolvable"—, we provide a similar result file, whose extension is expres. In this case, the metrics as presented as "Not executed (Unsolvable)" ("NE (Uns.)").

In Table 4.4, we present the statistics over 157 IMITATOR executions. Note that the *unsolvable* executions (which are computed with a timeout) are not included in this table.

4.3 CONCLUSION AND PERSPECTIVES

In this chapter, we presented a new version of the IMITATOR benchmark library. In its V2.0 version, the library contains 56 benchmarks with 119 different models and 216 properties. Each of these benchmarks come with many syntactic and semantic metrics, allowing user to choose the better examples for their use. We also provide their translation into the JANI specification.

Perspectives

To allow a version of our library in both the IMITATOR format and the JANI format, we implemented within IMITATOR the translation of an IMITATOR model to its JANI specification. In order for IMITATOR to also be fed with other existing models specified in JANI, we would like to implement the reverse translation (from JANI to IMITATOR), using JANI tools or within IMITATOR. This development would allow us to extend our library to include models offered by other JANI-supporting model checkers.

Ultimately, we hope our library can serve as a basis for a *parametric* timed model checking competition, a concept yet missing in the model checking community.

Opening the library to volunteer contributions is also on our agenda.

Finally, we hope that this library can be used as a reference for the comparison of verification algorithms. This will notably be the purpose of the the next chapter, in which we use our new version of the library in order to evaluate new heuristics for the efficient verification of PTAs.

5

EFFICIENT CONVEX ZONE MERGING IN PARAMETRIC TIMED AUTOMATA

Live as if you were to die tomorrow.
Learn as if you were to live forever.

— Mahatma Gandhi

In this chapter, we present a method allowing to reduce the PZGs of PTAs, based on merges of convex constraints. Effectively, our method provides a gain of 62 % of the average computation time over the benchmark library presented in the previous chapter compared to not using the option.

Motivation

PTAs are an inherently expressive but hard formalism, in the sense that most decision problems are undecidable (see e. g., [And19b] for a survey), while verification and parameter synthesis are subject to the infamous state-space explosion in practice. Reducing the state space, built on-the-fly when performing parameter synthesis, is a significant way to reduce the sometimes large computation times.

The symbolic semantics of TAs is often represented as *zones*, i. e., linear constraints over the clocks with a special form. In [Dav05], a convex zone merging technique is presented for Uppaal, that preserves reachability properties. This merging technique was extended to PTAs in [AFS13], and applied to the symbolic semantics of PTAs in the form of parametric zones, i. e., linear constraints over the clocks and the parameters, obeying to a special form [HRSV02]. In [AFS13], the analysis is only performed in the framework of the IM; no other properties are considered.

Contributions of the chapter

We propose here different merging techniques for PTAs, with the goal to reduce the state-space size and/or the analysis time. In Section 5.4, we implement our techniques in IMITATOR, and we perform extensive experiments on the benchmark library presented in Chapter 4. It turns out that these various heuristics have very different outcomes in terms of size of the state space and analysis speed. We then identify the best heuristics in practice, allowing to significantly decrease the number of states and the computation time, while preserving the correctness of the parameter synthesis for the whole class of reachability properties. The two main differences with [AFS13] are:

1. the definition of merging for reachability synthesis (and not only for IM); and
2. the systematic investigation of new heuristics, leading to a largely increased efficiency w.r.t. the original merging of [AFS13].

Organization of the chapter

We recall the notion of merging in Section 5.1 and present our merging algorithm in Section 5.2. Several merging heuristics are proposed in Section 5.3 and evaluated in Section 5.4. We conclude in Section 5.5.

5.1 THE NOTION OF MERGING

We recall the notion of merging from [AFS13]. Two states are *mergeable* if:

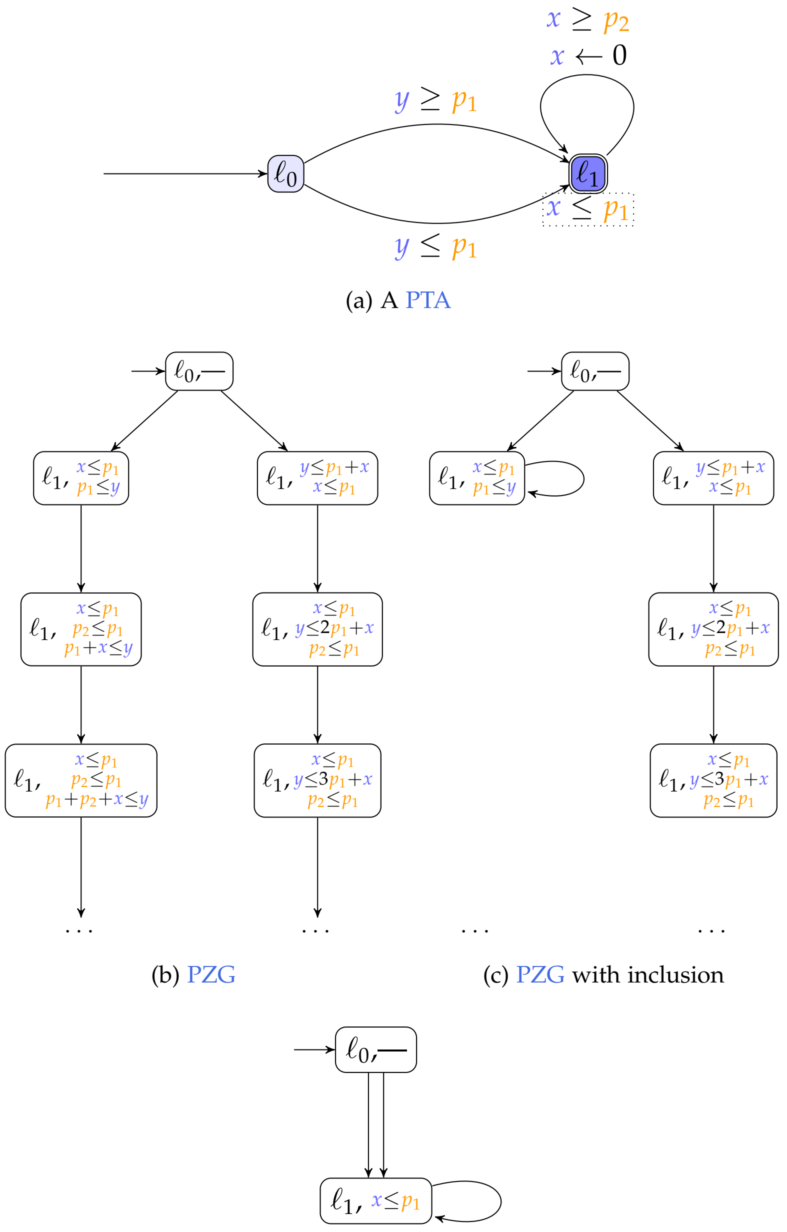
1. they share the same location; and
2. the union of their constraints is convex.

Definition 5.1 (Merging [AFS13]). Two symbolic states $\mathbf{s}_1 = (\ell_1, \mathbf{C}_1)$, $\mathbf{s}_2 = (\ell_2, \mathbf{C}_2)$ are *mergeable*, denoted by the predicate $\mathit{is\_mergeable}(\mathbf{s}_1, \mathbf{s}_2)$, if $\ell_1 = \ell_2$ and $\mathbf{C}_1 \cup \mathbf{C}_2$ is convex.
In that case, we define their *merging* as $(\ell_1, \mathbf{C}_1 \cup \mathbf{C}_2)$.

Remark 3. *Merging is a generalization of inclusion abstraction (also known as subsumption) [LODLDP13; AAPP21]. Note that if $\mathbf{s}_2$ includes $\mathbf{s}_1$, i. e., $\mathbf{C}_1 \subseteq \mathbf{C}_2$, then $\mathbf{C}_1 \cup \mathbf{C}_2 = \mathbf{C}_2$ is convex, so the states can be merged, and the result will be $\mathbf{s}_2$.*

■

Example 5.1. *We display examples of 2-dimensional zones in Figure 5.3c on page 63. Note that these box-shaped parametric zones are fictitious and displayed for the purpose of illustration; similar zones, sometimes using "diagonal" edges, can be obtained from actual PTAs. Zone $\mathbf{C}_1$ can be merged with $\mathbf{C}_4$; $\mathbf{C}_2$ can also be merged with $\mathbf{C}_4$. The result of these two merging operations is shown in Figure 5.3g. These two new zones can also be merged together, leading to the zone in Figure 5.3h.*



(a) A PTA

(b) PZG

(c) PZG with inclusion

(d) PZG with merge

Figure 5.1: Example of a PTA with infinite PZG that becomes finite by merging

> **Example 5.2.** *Let us now consider the PTA in Figure 5.1a, with two clocks (x and y) and two parameters (p_1 and p_2).*
> *Both clocks and parameters are initially bound to be non-negative (clocks initially different from 0 can be simulated using an appropriate gadget, omitted here).*
> *Two transitions lead from the initial location ℓ_0 to ℓ_1, with a guard differing in the condition on clock y. The self-loop on ℓ_1 can be taken depending on the value of clock x, and resets it.*

The PZG of this PTA is shown in Figure 5.1b. It features two separate infinite executions which depend on the first chosen transition. In the right-hand branch, the first state with location ℓ_1 has constraint $y \leq p_1 + x$ (which can be read $y - x \leq p_1$) since, although we have $y \leq p_1$ when taking the transition from ℓ_0, time can then elapse in ℓ_1—but only up to p_1 time units, due to invariant $x \leq p_1$. Then coefficients on p_1 (i. e., $2p_1$ then $3p_1$, etc.) start to appear from the second state with location ℓ_1 due to the self-loop on ℓ_1 that resets x.
Inclusion *reduces one of these two symbolic executions, which exhibits decreasing zones, as in Figure 5.1c. However, even using inclusion, the PZG remains infinite.*
Finally, Figure 5.1d displays the graph obtained with the merging *approach: the two states obtained after taking a single transition can indeed be merged. In this example, the PZG with merging becomes finite, which illustrates the importance of merging.*

5.1.1 *Preservation of Properties*

In the following, we prove that our merging method preserve reachability properties.

Proposition 5.1. *Given a PTA $\mathcal{P}$, let $\mathbf{PZG}(\mathcal{P})$ and $\mathbf{PZG}(\mathcal{P})'$ be the PZG before and after merging. Then $\mathbf{PZG}(\mathcal{P})'$ simulates $\mathbf{PZG}(\mathcal{P})$.*

Proof. (sketch) Consider the relation $\mathbf{s} \sqsubseteq \mathbf{s}'$ *iff* $\mathbf{s}.loc = \mathbf{s}'.loc$ and $\mathbf{s}.constr \subseteq \mathbf{s}'.constr$. [NPP18] shows that this forms a simulation relation, i. e., if $\mathbf{s} \sqsubseteq \mathbf{s}'$ and $\mathbf{s} \Rightarrow \mathbf{s}_0$, then for some $\mathbf{s}'_0$, we have $\mathbf{s}_0 \sqsubseteq \mathbf{s}'_0$ and $\mathbf{s}' \Rightarrow \mathbf{s}'_0$.

Note that while merging, we repeatedly replace a state $(\ell, \mathbf{C})$ by a state $(\ell, \mathbf{C} \cup \mathbf{C}')$, in which case $(\ell, \mathbf{C}) \sqsubseteq (\ell, \mathbf{C} \cup \mathbf{C}')$. So indeed, merged states can simulate the behavior of all original states that were merged. □

Corollary 5.1. *Given a PTA $\mathcal{P}$, let $\mathbf{PZG}(\mathcal{P})$ and $\mathbf{PZG}(\mathcal{P})'$ be the PZG before and after merging. Let φ be a property in ∀CTL* [BK08] with atomic propositions defined in terms of state locations only. Then $\mathbf{PZG}(\mathcal{P})' \models \varphi$ implies $\mathbf{PZG}(\mathcal{P}) \models \varphi$.*

Proof. (sketch) All universal properties (in ∀CTL*) are preserved by simulation [BK08, Theorem 7.76]. In this case, the simulation $\sqsubseteq$ also implies that related states have the same locations, so they satisfy the same atomic properties. □

The next proposition shows that we do not add arbitrary new behavior. Indeed, although merging can add behavior, it cannot add unreachable locations and, more precisely, the set of locations reachable for each parameter valuation remains unchanged. This guarantees that merging preserves reachability synthesis. A version of this result was shown in a different context in [AFS13, Theorem 1].

Proposition 5.2 (Preservation of reachability properties)**.** *Given a PTA $\mathcal{P}$, let $\mathbf{PZG}(\mathcal{P})'$ be the PZG after merging. Let ℓ be a location, let v be a parameter valuation.*
ℓ is reachable in $v(\mathcal{P})$ iff $\exists(\ell, \mathbf{C}') \in \mathbf{PZG}(\mathcal{P})'$ such that $v \in \mathbf{C}'{\downarrow_{\mathbb{P}}}$.

Proof.

- $\Rightarrow$ Let us show that, if ℓ is reachable in $v(\mathcal{P})$, then there exists $(\ell, \mathbf{C}') \in \mathbf{PZG}(\mathcal{P})'$ such that $v \in \mathbf{C}'{\downarrow_{\mathbb{P}}}$.

  Let $\mathbf{PZG}(\mathcal{P})$ be the PZG without merging. By the equivalence between the concrete and symbolic semantics [HRSV02], there exists $(\ell, \mathbf{C}) \in \mathbf{PZG}(\mathcal{P})$ such that $v \in \mathbf{C}{\downarrow_{\mathbb{P}}}$. As merging only adds behavior, and creates larger constraints, then there exists $(\ell, \mathbf{C}') \in \mathbf{PZG}(\mathcal{P})'$ such that $\mathbf{C} \subseteq \mathbf{C}'$; therefore, $v \in \mathbf{C}{\downarrow_{\mathbb{P}}}$ implies $v \in \mathbf{C}'{\downarrow_{\mathbb{P}}}$.

- $\Leftarrow$ Let us now show that, if there exists $(\ell, \mathbf{C}') \in \mathbf{PZG}(\mathcal{P})'$ such that $v \in \mathbf{C}'{\downarrow_{\mathbb{P}}}$, then ℓ is reachable in $v(\mathcal{P})$.

  We prove this by induction on the number of merge- and successor- operations in the algorithm. Note that symbolic states in $\mathbf{PZG}(\mathcal{P})'$ can arise in two ways:

  - As a merge of previous states $(\ell, \mathbf{C}_1)$ and $(\ell, \mathbf{C}_2)$ into $(\ell, \mathbf{C}_1 \cup \mathbf{C}_2)$. Let v and μ be given, with $\mu \models v(\mathbf{C}_1 \cup \mathbf{C}_2)$. Then either $\mu \models v(\mathbf{C}_1)$ or $\mu \models v(\mathbf{C}_2)$. So either $v \in \mathbf{C}_1{\downarrow_{\mathbb{P}}}$ or $v \in \mathbf{C}_2{\downarrow_{\mathbb{P}}}$, and by induction hypothesis, (ℓ, μ) is reachable in $v(\mathcal{P})$.

  - As a symbolic successor of a state, i. e., $((\ell', \mathbf{C}'), e, (\ell, \mathbf{C})) \in \Rightarrow$, for some edge e.

    From the property of the PTA semantics [HRSV02] that parameter valuations can only be restricted over a symbolic run, we have that $\mathbf{C}'{\downarrow_{\mathbb{P}}} \supseteq \mathbf{C}{\downarrow_{\mathbb{P}}}$ and therefore $v \in \mathbf{C}{\downarrow_{\mathbb{P}}}$ implies $v \in \mathbf{C}'{\downarrow_{\mathbb{P}}}$. By induction hypothesis, ℓ' is therefore reachable in $v(\mathcal{P})$. Following a reasoning similar to the case above, since merging can only add behaviors, then there exists a transition between ℓ' to ℓ in $v(\mathcal{P})$; and therefore ℓ is reachable in $v(\mathcal{P})$.

□

Note that path properties in $\mathbf{PZG}(\mathcal{P})$ are not always preserved in $\mathbf{PZG}(\mathcal{P})'$, as the following example shows. Also, liveness properties in $\mathbf{PZG}(\mathcal{P})$ (such as "every path visits location ℓ infinitely often") are not necessarily preserved in $\mathbf{PZG}(\mathcal{P})'$.

Example 5.3. *Figure 5.2b shows the PZG of the PTA in Figure 5.2a. The maximal paths are $\ell_0, \ell_1, \ell_3, \ell_4$ and $\ell_0, \ell_1, \ell_3, \ell_2$ (for $p \leq 1$) and ℓ_0, ℓ_2, ℓ_3 (for $p > 1$). All maximal paths satisfy the LTL [Pnu77] property $\square(\ell_2 \rightarrow \square \neg \ell_4)$ ("no ℓ_4 after an ℓ_2"). Also, there is no loop (infinite run) containing ℓ_2.*
However, the result after merging in Figure 5.2c introduces the spurious path $\ell_0, \ell_2, \ell_3, \ell_4$, violating the first property. It also introduces a spurious loop $\ell_0, (\ell_2, \ell_3)^{\omega}$, around ℓ_2.

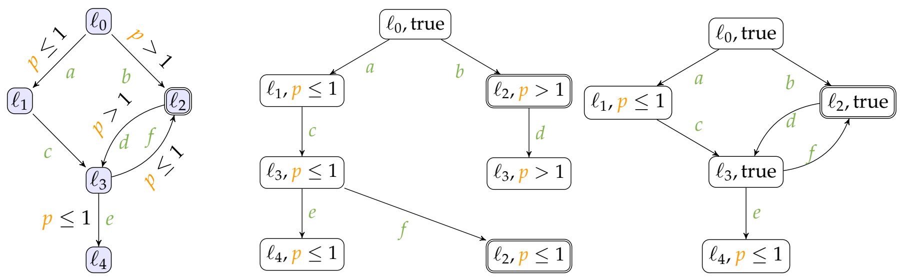
Remark 4. *This example uses parameters, but no clocks. [LODLDP13, Fig. 4] shows an example with only clocks (i. e., a TA) where a spurious loop is introduced by zone inclusion (subsumption), which is, as specified earlier, just a special case of zone merging.*

■

5.2 MERGING ALGORITHM

In the following, we introduce the algorithms used for the construction of the PZG. We figure in each algorithm the heuristics; they are discussed in Section 5.3.

Algorithm 5.1 constructs the state space for a given PTA $\mathcal{P}$ by BFS from the initial state $\mathbf{s}_0$. It computes the set of reachable states **Visited** by repeatedly adding the next layer of successor states $\mathbf{Q}_{new}$ (Line 8), maintaining the transitions (Line 9). Note that each iteration (Line 11) calls a merging function *mergeSets* (given in Algorithm 5.2), which may reduce both **Visited** and **Queue**. This call to the merging function is the crux of our approach.



(a) A PTA (b) PZG before merging (c) PZG after merging

Figure 5.2: No preservation of paths properties after merging

Algorithm 5.1: BFS by layer layerBFS($\mathcal{P}$)

```
/* Building $\mathbf{PZG}(\mathcal{P}) = (E, \mathbf{S}, \mathbf{s}_0, \Rightarrow)$ */
$\mathbf{Visited} \leftarrow \{\mathbf{s}_0\}$
$\mathbf{Queue} \leftarrow \{\mathbf{s}_0\}$
$\Rightarrow \leftarrow \emptyset$
while $\mathbf{Queue} \neq \emptyset$ do
    $\mathbf{Q}_{new} \leftarrow \emptyset$
    foreach $\mathbf{s} \in \mathbf{Queue}$ do
        foreach $(e, \mathbf{s}') \in \mathsf{SuccE}(\mathbf{s})$ do
            $\mathbf{Q}_{new} \leftarrow \mathbf{Q}_{new} \cup (\{\mathbf{s}'\} \setminus \mathbf{Visited})$
            $\Rightarrow \leftarrow \Rightarrow \cup \{(\mathbf{s}, e, \mathbf{s}')\}$
    $\mathbf{Visited} \leftarrow \mathbf{Visited} \cup \mathbf{Q}_{new}$
    $\mathbf{Visited}, \mathbf{Queue} \leftarrow mergeSets(\mathbf{PZG}(\mathcal{P}), \mathbf{Visited}, \mathbf{Q}_{new})$
```

Algorithm 5.2: Heuristics to merge states within Q and/or V

```
Visited   Queue   Ordered
Function $mergeSets(\mathbf{PZG}(\mathcal{P}), V, Q)$
    foreach $\mathbf{s} \in Q$ do
        $mergeOneState(\mathbf{s}, \mathbf{PZG}(\mathcal{P}), Q)$
        $mergeOneState(\mathbf{s}, \mathbf{PZG}(\mathcal{P}), V)$
    return $(V, Q)$
```

Algorithm 5.1 can be extended, depending on the analysis or parameter synthesis problem. For instance, an invariant property can be checked for each reachable symbolic state and terminate as soon as the property is violated. For reachability synthesis (EFsynth), one may accumulate all solutions as a set of constraints that lead to a state satisfying a property. As an optimization, one can prune the state space on reaching states that lie already within the accumulated constraints, since these cannot lead to new solutions. We do not add such extensions to the algorithm, since this chapter focuses on the merging of states during state space generation.

Then, Algorithm 5.2 "simply" calls recursively the *mergeOneState* function (given in Algorithm 5.3) on each state of **Queue**, using additional arguments **PZG**($\mathcal{P}$) and **Visited** and/or **Queue**. The heuristics to select arguments for calls to *mergeOneState* will be discussed later. Note that *mergeOneState modifies* its arguments, notably **PZG**($\mathcal{P}$) (in the implementation, we use a call by reference).

Algorithm 5.3: Merging a state with an update of the statespace $\mathbf{PZG}(\mathcal{P}) = (E, \mathbf{S}, \mathbf{s}_0, \Rightarrow)$ on-the-fly.

The variant with restart after a merge is indicated as in Restart

```
1  Function mergeOneState($\mathbf{s}$, $\mathbf{PZG}(\mathcal{P})$, SiblingCandidates)
2      $isMerged \leftarrow$ false
3      Candidates $\leftarrow getSiblings(\mathbf{s}$, SiblingCandidates$)$
4      foreach $\mathbf{y} \in$ Candidates do
           /* Mergeability test                                  */
5          if $is\_mergeable(\mathbf{s}, \mathbf{y})$ then
6              $\mathbf{s}.constr \leftarrow \mathbf{s}.constr \cup \mathbf{y}.constr$
7              $isMerged \leftarrow$ true
               /* Update transition targets and source            */
8              foreach $\mathbf{t} \in \Rightarrow$ do
9                  if $\mathbf{t}.target = \mathbf{y}$ then $\mathbf{t}.target \leftarrow \mathbf{s}$ ;
10                 if $\mathbf{t}.source = \mathbf{y}$ then $\mathbf{t}.source \leftarrow \mathbf{s}$ ;
               /* Delete $\mathbf{y}$                                 */
11             $\mathbf{S} \leftarrow \mathbf{S} \setminus \{\mathbf{y}\}$
               /* Handle initial state                            */
12             if $\mathbf{s}_0 = \mathbf{y}$ then $\mathbf{s}_0 \leftarrow \mathbf{s}$ ;
13     if $isMerged$ then mergeOneState($\mathbf{s}$, $\mathbf{PZG}(\mathcal{P})$, SiblingCandidates) ;
```

Algorithm 5.3 attempts at merging a state $\mathbf{s}$ while looking for candidate states in **SiblingCandidates**. We first look for the *siblings* of $\mathbf{s}$ (states with same location) within **SiblingCandidates** (Line 3). We use a function $getSiblings((\ell, \mathbf{C}), \mathbf{S})$ that returns the siblings, as in $\{(\ell', \mathbf{C}') \in \mathbf{S} \mid \ell = \ell'\}$. If the union of the constraints is convex (Line 5), then state $\mathbf{S}$ becomes the result of merging $\mathbf{s}$ with the candidate $\mathbf{y}$ (Line 6). The candidate $\mathbf{y}$ is deleted (Line 11), as well as all transitions leading to or coming from it (Lines 8 to 10). We finally modify the initial state $\mathbf{s}_0$ of $\mathbf{PZG}(\mathcal{P})$, in case it was merged (Line 12).

5.3 HEURISTICS FOR MERGING

We now introduce and discuss several heuristics for merging states, leading to various options in the merging algorithm. There is no provably best option that is guaranteed to be superior over all other possible options. We will perform extensive experiments in Section 5.4 to find out what works well on a number of benchmarks. The two main driving forces to select between these options are:

1. a maximal reduction of the state space; and
2. a minimization of the computation time.

Although usually smaller state spaces tend to require less computation time, this is not always the case. Sometimes one might need extra effort to check if states can be merged, in order to perform even more reduction. Subsequent computations might profit from the smaller state space, but if one is checking properties on-the-fly, the extra effort might not be justifiable. The discussion on the options will be guided by some questions.

The subsequent Example 5.4 will show that different choices can indeed lead to state spaces of different size. Note that, even when we fix the answers to the questions, the result is still non-deterministic, since the result of merging depends on the *order* in which we would consider the siblings.

Question 1: What to merge with what?

Assume that we are computing the next level of reachable states in a BFS process (Algorithm 5.1). Assuming that the states in **Visited** have been properly merged, we clearly still need to merge the new states in $\mathbf{s} \in$ **Queue**. What to merge them with? Do we only compare $\mathbf{s}$ with other states in the **Queue**? Or also with **Visited**? If we merge with **Visited** states, the final state space could become smaller. On the other hand, since time was spent to compute those states already, is it worth looking at them? The different strategies considered merge a new state with its siblings:

- only in the queue (Queue), or
- in all visited states (Visited) (including the queue), or
- first in the queue and, after that, in the visited list (Ordered).

These different possibilities are pictured by different colors in Algorithm 5.2.

Question 2: Restart after a merge?

The next question is what to do if we find that $\mathbf{s}$ could be merged with some $\mathbf{s}'$ into the (larger) $\mathbf{s}_m$? We have already searched through some set Q' of states before we found $\mathbf{s}'$. Those states in Q' could not be merged with $\mathbf{s}$. However, it could be possible that a state $\mathbf{s}'' \in Q'$ can be merged with $\mathbf{s}_m$. So should we *restart* the search (and lose some time to find more reduction), or should we just resume the search, and only find merge candidates for $\mathbf{s}_m$ in the remaining states that we have not yet considered? So, if a state can be merged with one of its siblings, should we restart or not restart the search through all candidate siblings?

Question 3: When to update the statespace?

Assume that we find a successful merge of a state $\mathbf{s} \in \mathbf{Queue}$ with some other state $\mathbf{s}' \in \mathbf{Visited}$, leading to a larger state $\mathbf{s}_m$. How do we now modify the already computed part of the state space? We replace $\mathbf{s}'$ by $\mathbf{s}_m$, redirecting all transitions going to $\mathbf{s}'$ to $\mathbf{s}_m$. This could make the successors of $\mathbf{s}'$ unreachable, so we could also redirect transitions from $\mathbf{s}'$ to transitions from $\mathbf{s}_m$. Alternatively, we could just remove the successors of $\mathbf{s}'$. This is valid, since we will still compute all successors of $\mathbf{s}_m$ in the next level. Similar considerations apply to all states reachable from successors of $\mathbf{s}'$.

These approaches can have unforeseen effects. First of all, if we remove successor states, they cannot act anymore as merge candidates, thus potentially blocking future merges (see also Question 5). Second, removing transitions may change the "shortest path" to reachable states, leading to wrong answers for depth-bounded and shortest-path searches. Third, not removing states leads to a larger state space than necessary. Finally, doing a full reachability analysis is linear in the size of the state space generated so far (but does not involve any polyhedra computations).

So one question is how often we should update the computed part of the state space? The options we considered are to do "garbage collection" after:

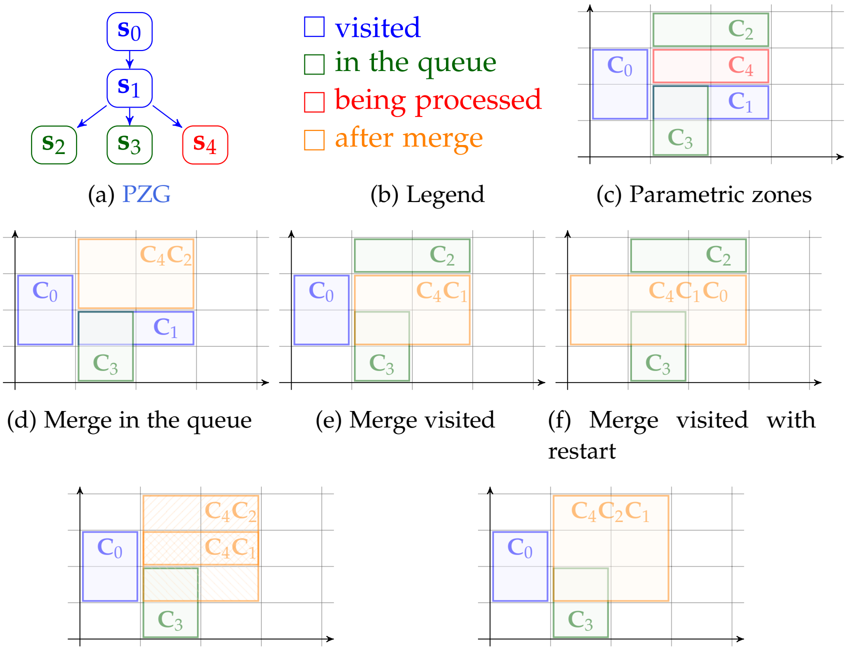
- each merge with each sibling (uMerge), or
- having processed the whole candidate list of a state (uCandidates)

Question 4: How to update the state space?

The "garbage collection" can be implemented in two ways. If we can merge a state, we update the state space:

- reconstruct: with a copy of the reachable part of the statespace; or
- on-the-fly: deleting the merged state and updating its transitions *in situ*.

Deleting states on-the-fly is cheaper than running a separate algorithm to mark and copy the reachable part of the state space. However, note that when updating transitions on-the-fly, some unnecessary successor states may stay in the state space. On the one hand, these unnecessary states and transitions lead to a waste of memory. On the other hand, they might still be useful as merge candidates for future merges.



(a) PZG

(b) Legend

(c) Parametric zones

(d) Merge in the queue

(e) Merge visited

(f) Merge visited with restart

(g) Queue then visited

(h) Update after merge of 5.3g

Figure 5.3: Illustration of the merging heuristics

Example 5.4. *Recall that Figure 5.3 presents a fictitious example summarizing the effect of the options discussed in this section. The PZG (with five states) is shown in Figure 5.3a, the corresponding projections of the zones on the parameters in Figure 5.3c and the legend for the different colors in Figure 5.3b. All states have the same location, hence may be candidates for merging. Two states ($\mathbf{s}_0$ and $\mathbf{s}_1$) are in the visited set, while two are in the queue ($\mathbf{s}_2$ and $\mathbf{s}_3$), and the last one, $\mathbf{s}_4$, is currently being handled.*

Let us first consider that the merge is only done with states in the queue. Then $\mathbf{s}_4$ is merged with $\mathbf{s}_2$ and no merge with $\mathbf{s}_3$ can occur. This leads to the zones depicted in Figure 5.3d.

Let us now consider that the merge is done with all visited states. Then the following execution becomes possible: State $\mathbf{s}_4$ could be first merged with $\mathbf{s}_1$, leading to the zones in Figure 5.3e. Now, if the restart option is used, this newly computed zone could be merged with $\mathbf{s}_0$, leading to the zones in Figure 5.3f. Note that we cannot merge the result with $\mathbf{s}_2$ anymore.

Finally, let us consider the case where we merge with the queue first and then with visited. State $\mathbf{s}_4$ is then merged with $\mathbf{s}_2$, as in Figure 5.3d. Then no merge with $\mathbf{s}_3$ nor with $\mathbf{s}_0$ can be performed, but a merge with $\mathbf{s}_1$ is possible, leading to Figure 5.3g. If furthermore the state space is updated immediately after a merge, the new state (instead of $\mathbf{s}_4$) is merged with $\mathbf{s}_1$, leading to Figure 5.3h.

Table 5.1: Size of our dataset

	# benchmarks	# models	# properties
Whole reachability dataset	49	84	124
Where at least one execution ends within 120 s	35	68	102
Where at least one merge is performed	24	35	42

5.4 EXPERIMENTS

We evaluate here the effect of the merging heuristics on reachability synthesis, i. e., the synthesis of the parameter valuations for which a given reachability property holds. The synthesis algorithm explores the PZG to find all valid parameter valuations.

5.4.1 *Implementation*

I implemented all our heuristics in IMITATOR. In this tool, the parametric zones in the symbolic states are encoded using polyhedra. All operations on polyhedra, and notably the mergeability test, are performed using PPL.

We also reimplemented in the latest version of IMITATOR the original merging technique of IMITATOR 2.12 of [AFS13] (which was an upgrade of the merging technique in IMITATOR 2.6.1), so as to obtain a fair comparison. To avoid differences due to orthogonal improvements and thus enable a fair experiment, the 2.12 merge functionality has been re-introduced in the used version of IMITATOR.

5.4.2 *Dataset and experimental environment*

We use the full set of models with reachability properties from the IMITATOR benchmark library, described in Chapter 4.

Our dataset comprises 124 pairs made of a model and a reachability property (i. e., 124 possible executions of IMITATOR). We set a timeout of 120 s; only 102 executions terminate within this time bound for at least one of the merging heuristics. For 42 of these executions, at least one of the heuristics performs at least one successful merge. Full statistics on our dataset are given in Table 5.1.

Experiments were run on an Intel Xeon Gold 5220 (Cascade Lake-SP, 2.20GHz, 1 CPU/node, 18 cores/CPU) with 96 GiB running Linux Ubuntu 20 (“gros” cluster of the Grid’5000 testbed [Gri]), using IMITATOR 3.3-beta-2 “Cheese Caramel au beurre salé”. The sources, binaries, models, raw results and full experiments tables are available at 10.5281/zenodo.6806915.

5.4.3 *Description of the Experiments*

We compare each combination of the heuristics proposed in Section 5.3. We reference each merge heuristic as a combination of three or four letters:

1. R or O: the state space is updated by reconstruction (R) or on-the-fly (O), according to Question 4;
2. V,Q or O: the selected candidates are Visited (V), Queue (Q) or Ordered (O), according to Question 1;
3. M or C: state space is updated for each merge (M) or after all candidates (C), according to Question 3;
4. r: the restart option is enabled (nothing otherwise), according to Question 2.

We refer to the previous implementation of merging (reintroduced in IMITATOR 3.3-beta-2) with M2.12. Nomerge denotes the experiments performed when the merging option is disabled.

These algorithms are compared according to:

1. the total computation time needed for a property; and
2. the size of the generated state space.

5.4.4 *Results*

Our results are obtained over the 102 executions of the dataset for which at least one algorithm ends before reaching the 120 s timeout. We do not use any penalty on executions that do not end: their computation time is set to the timeout (120 s) in the subsequent analyses. Recall that, among these 102 executions, only 42 have states that can be merged (by at least one of the studied heuristics), while 60 do not perform any merge for any heuristic.

The metrics tagged by "(merge)" in Tables 5.2 and 5.3 are computed over the 42 executions where some states can be merged, while the "(no merge)" only consider the 60 executions where no merge can be made.

We present in Table 5.2 some of the experimental results obtained for the different merge heuristics that allow the best reduction of computation time or in the state-space size. The results for all the heuristics are presented in Table 5.3 on page 70. In order to allow a good visualization of the results, the best result in each cell is given in **bold**, while the level of green denotes the "quality" of the value in each cell (white is worst, and 100 % green is best).

The different lines tabulate the following information:

Table 5.2: Partial results comparing merge heuristics on time and state-space size over 102 models

		Nomerge	M2.12	RVMr	OQM
Time	# wins	24	20	22	**42**
	Avg (s)	10.0	5.47	4.56	**3.77**
	Avg (merge) (s)	18.8	7.83	5.57	**3.63**
	Avg (no merge) (s)	3.83	**3.82**	3.85	3.88
	Median (s)	1.39	1.2	1.14	**1.12**
	Norm. avg	1.0	0.91	0.91	**0.87**
	Norm. avg (merge)	1.0	0.75	0.74	**0.64**
	Norm. avg (no merge)	**1.0**	1.02	1.03	1.03
States	# wins	0	19	**37**	16
	Avg	11,443.08	11,096.54	**11,064.37**	11,120.79
	Avg (merge)	1,512.02	670.43	**592.31**	729.33
	Median	2389.5	703.5	**604.5**	905.0
	Norm. avg	1.0	0.86	**0.84**	0.88

1. the number of wins over the computation time, i. e., the number of executions for which the current heuristics gives the smallest execution time;

2. the average time (in seconds) over all executions;

3. the average time (in seconds), excluding executions where no states can be merged for any heuristics;

4. the average time (in seconds) for only the executions where no states can be merged for any heuristics;

5. the median time (in seconds) over all executions;

6. the normalized time average, compared to the Nomerge results, i. e., the ratio between the heuristic execution time and the Nomerge one;

7. the normalized time average, excluding executions where no states can be merged for any heuristics;

8. the normalized time average, for only the executions where no states can be merged for any heuristics;

We also present the results over the size of the state space (i. e., the number of states after the merging phase):

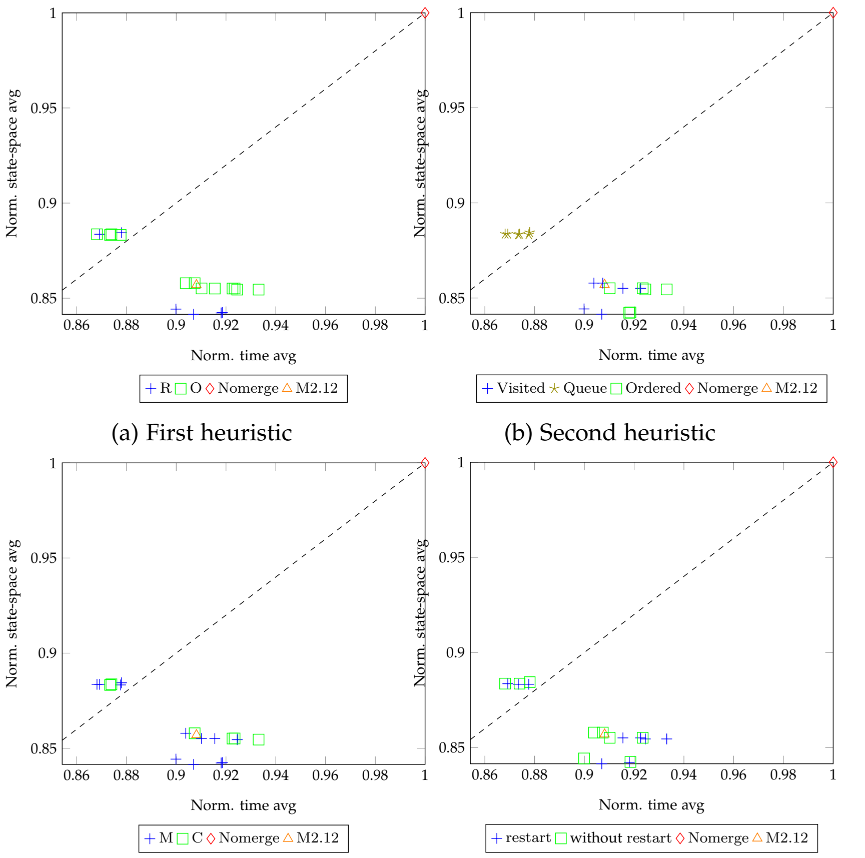
1. the number of wins over the size of the state space (i. e., the total number of symbolic states after merging);

2. the average size of the state space over all models;

3. the average size of the state space over all models, excluding the executions where no states can be merged;

4. the median size of the state space over all models;

5. the normalized size average, compared to the Nomerge results.



(a) First heuristic

(b) Second heuristic

(c) Third heuristic

(d) Fourth heuristic

Figure 5.4: Plot of merging experiment results following the different heuristic choices

The reason to give both an average time (resp. number of states) and a normalized time (resp. number of states) is because both metrics complement each other: the weight of the large models has a higher influence in the average (which can be seen as unfair, as a few models have a large influence), while all models have equal influence in the normalized average (which can also be seen as unfair, as very small models have the same influence as very large models).

In Figure 5.4, we plot the normalized averages on time and state-space size in scatter plots, according to the heuristic choices. For readability concerns and to ensure a printable scale, the result entries were these averages are too large are not displays. It concerns heuristics were the normalized time average is greater than 8 (heuristics RVCr, RVC, RQCr, RQC, ROCr, ROC).

In Tables 5.2 and 5.3, we notice that the best (i. e., smallest) times are obtained when the merging is performed on the queue and when the update is done after a performed merge, even though doing it with a reconstruction of the state space after each step loses time compared to the Otf heuristic. Moreover, restarting when a merge is performed does not seem to bring any gain in time. Thus, with respect to time, when the winner is OQM (i. e., merging when the candidates are taken from the Queue, when the update is done on-the-fly after each merge without any restart), which minimizes both the time when a merge is possible, but also when considering models where no merge can be performed. Moreover, this heuristic gives the smaller times for the executions where no merge can be done.

Concerning the state space size, the winner is RVMr (i. e., merging when the candidates are taken from the Visited states, updating the state space by a reconstruction after each merge, and with a restart if a merge can be performed). This performs more checks to identify states for merging (comparing with all the visited, not only those in the queue), thus reducing the state space even more.

Note that the methods Nomerge and M2.12 are almost always the losers (i. e., slowest and largest state space), except for the heuristic where the update of the state space is performed after the list of candidates.

Concerning our new heuristics, we note that OQM decreases the average computation time to 69 % when compared to the previous merging heuristic (M2.12), and even to only 46 % (i. e., a division by a factor > 2) compared with M2.12 on the subset of models for which at least one merge can be done. Compared to disabling merging (Nomerge), our new heuristic OQM decreases to 38 % on the whole benchmark set, and even to 19 % (i. e., a division by a factor > 5) on the subset of models for which at least one merge can be done. This leads us to consider the new combination of merging only in the queue and with an on-the-fly update after each merge and without restart (heuristic OQM) as the default merging heuristic in IMITATOR.

For use cases that require a minimal state space, the new combination RVMr is the recommended option. Note that this version is still faster on average (83 %) than the previous heuristic (M2.12), and more than twice as fast (46 %) as not merging at all (Nomerge).

5.5 CONCLUSION AND PERSPECTIVES

In this chapter, we investigated the importance of the merging operations in reachability synthesis using PTAs. We investigated different combinations of options.

The chosen heuristic (OQM, when the candidates are taken from the Queue, when the update is done on-the-fly after each merge without any restart) brings a decrease to 38 % of the average computation time for our entire benchmark library compared to the absence of merging. In other words, despite the cost of the mergeability test, the overall gain is large and shows the importance of the merge operation for parameter synthesis.

Compared to the previous merging heuristic from [AFS13], the gain of our new heuristic is a decrease to 69 % of the average computation time—meaning that our new heuristic decreases the computation time by 31 % compared to the former heuristic from [AFS13]. We also provide an heuristic for use cases where a minimal state space is important, for instance for a follow-up analysis. Even though this is not the fastest heuristic, it is still faster than not merging at all, and faster than the old merging heuristic [AFS13].

Our experiments show the high importance of carefully choosing the merging heuristics. Finally, our heuristics preserve the correctness of parameter synthesis for reachability properties.

Perspectives

We noted that pruning merged states away ("garbage collection") can prevent future merges. Another option would be to keep such states in a collection of "potential mergers". These extra states could be useful as "glue" to merge a number of other states, that otherwise could not be merged, into one superstate—but at the cost of more memory. Another option could be to merge more than two states in one go. These options remain to be investigated.

Other options of our questioning could be investigated. For example, one could try to update the statespace after having processed all states in a complete level, creating a new version of this heuristic.

It is well-known that less heuristics can be used for *liveness* properties than for reachability properties. Investigating whether *some* merging can still be used for liveness synthesis (i. e., the synthesis of parameter valuations for which some location is infinitely often reachable) is an interesting future work.

Investigating the recent PPLite [BZ18; BZ20] instead of PPL for polyhedra computation is on our agenda.

Another, more theoretical question is to define and compute the "best possible merge". Currently, the result of merging is not canonical, since it depends on the exploration order and the order of searching for siblings. We have not found a candidate definition that minimizes the state space and provides natural, canonical merge representatives.

Finally, these better performances in parametric timed model checking can be applied to security contexts, which we do in the next part of the manuscript.

Table 5.3: Complete results comparing merge heuristics on time and state-space size over 102 models

	With Restarting	Nomerge	M2.12	RVMr	RVCr	RQMr	RQCr	ROMr	ROCr	OVMr	OVCr	OQMr	OQCr	OOMr	OOCr
Time	# wins	**20**	17	15	2	10	3	5	3	9	4	8	8	4	8
	Avg (s)	10.0	5.47	4.56	46.7	3.84	43.53	4.69	49.05	5.58	5.7	**3.79**	3.81	5.54	5.63
	Avg (merge) (s)	18.8	7.83	5.57	17.8	3.83	10.1	5.86	18.69	8.0	8.32	**3.66**	3.7	7.89	8.14
	Avg (no merge) (s)	3.83	**3.82**	3.85	66.93	3.85	66.93	3.88	70.29	3.89	3.87	3.88	3.88	3.89	3.87
	Median (s)	1.39	1.2	1.14	4.85	1.14	2.98	1.19	6.49	1.15	1.15	1.12	**1.11**	1.15	1.17
	Nrm. avg	1.0	0.91	0.91	9.06	**0.87**	8.91	0.92	10.34	0.92	0.92	0.88	**0.87**	0.92	0.93
	Nrm. avg (merge)	1.0	0.75	0.74	1.74	0.66	1.4	0.76	1.86	0.75	0.76	0.66	**0.65**	0.77	0.78
	Nrm. avg (no-mrg)	**1.0**	1.02	1.03	14.27	1.02	14.25	1.03	16.38	1.03	1.03	1.04	1.03	1.03	1.04
States	# wins	0	19	**32**	29	15	15	29	30	20	20	16	16	20	20
	Avg	11445.61	11096.54	**11064.37**	11106.09	11120.34	11120.55	11066.85	11105.73	11089.5	11089.5	11118.73	11118.73	11087.77	11087.77
	Avg (merge)	1518.17	670.43	**592.31**	693.62	728.24	728.74	598.33	692.74	653.33	653.33	724.31	724.31	649.14	649.14
	Median	2389.5	703.5	**604.5**	607.0	905.0	905.0	**604.5**	607.0	701.0	701.0	905.0	905.0	701.0	701.0
	Nrm. avg	1.0	0.86	**0.84**	0.85	0.88	0.88	**0.84**	0.85	0.86	0.86	0.88	0.88	0.85	0.85

	No Restarting	Nomerge	M2.12	RVM	RVC	RQM	RQC	ROM	ROC	OVM	OVC	OQM	OQC	OOM	OOC
Time	# wins	**20**	17	11	4	8	3	4	2	6	6	19	6	8	8
	Avg (s)	10.0	5.47	4.56	46.48	3.85	43.31	4.65	48.92	5.09	5.19	**3.77**	3.78	5.16	5.25
	Avg (merge) (s)	18.8	7.83	5.58	17.19	3.86	9.57	5.74	18.41	6.8	7.07	**3.63**	3.64	7.0	7.24
	Avg (no merge) (s)	3.83	**3.82**	3.85	66.97	3.85	66.94	3.89	70.28	3.89	3.88	3.88	3.88	3.88	3.87
	Median (s)	1.39	1.2	1.14	4.34	1.14	2.82	1.17	5.4	1.13	**1.12**	**1.12**	**1.12**	1.13	1.15
	Nrm. avg	1.0	0.91	0.9	9.04	0.88	8.91	0.92	10.33	0.9	0.91	**0.87**	**0.87**	0.91	0.92
	Nrm. avg (merge)	1.0	0.75	0.74	1.69	0.65	1.38	0.76	1.83	0.72	0.74	**0.64**	0.65	0.74	0.76
	Nrm. avg (no-mrg)	**1.0**	1.02	1.02	14.28	1.04	14.27	1.03	16.38	1.03	1.03	1.03	1.03	1.03	1.04
States	# wins	0	19	23	26	15	15	29	**32**	19	19	15	15	19	19
	Avg	11445.61	11096.54	11073.57	11105.33	11123.2	11122.24	**11068.0**	11105.41	11096.89	11096.89	11120.79	11120.79	11090.01	11090.01
	Avg (merge)	1518.17	670.43	614.64	691.79	735.17	732.83	**601.12**	691.98	671.29	671.29	729.33	729.33	654.57	654.57
	Median	2389.5	703.5	636.5	583.0	905.0	905.0	604.5	**577.0**	706.5	706.5	905.0	905.0	701.0	701.0
	Nrm. avg	1.0	0.86	**0.84**	0.85	0.88	0.88	**0.84**	0.85	0.86	0.86	0.88	0.88	0.86	0.86

Part II

Execution-time opacity

Grândola, Vila Morena
Terra da fraternidade
O povo é quem mais ordena
Dentro de ti, ó cidade

Dentro de ti, ó cidade
O povo é quem mais ordena
Terra da fraternidade
Grândola, Vila Morena

Em cada esquina, um amigo
Em cada rosto, igualdade
Grândola, Vila Morena
Terra da fraternidade

Terra da fraternidade
Grândola, Vila Morena
Em cada rosto, igualdade
O povo é quem mais ordena

À sombra duma azinheira
Que já não sabia a idade
Jurei ter por companheira
Grândola, a tua vontade

Grândola, a tua vontade
Jurei ter por companheira
À sombra duma azinheira
Que já não sabia a idade

— Zeca Afonso, Grândola, Vila Morena

6 PRELIMINARIES

All truths are easy to understand once they are discovered;
the point is to discover them.

— Galileo Galilei

We now move to the second part of this manuscript, which is concerned with execution-time opacity (ET-opacity). In the next chapters, we present the three contributions concerning this notion: ET-opacity problems are studied in Chapter 7, they are extended with expiring secrecy in Chapter 8, and we focus on untimed control for ET-opacity in Chapter 9.

In this chapter, we introduce general definitions for Chapters 7 to 9. Apart from the definition of RA arithmetic recalled in Section 6.2.1, all the content presented in this chapter is an original contribution of this thesis.

Organization of the chapter

In Section 6.1, we present preliminary definitions and we develop in Section 6.2 the computation of two particular sets: the set of execution times for runs visiting a particular (private) location, and the one for the runs not visiting it (denoted $\mathit{DVisit}^{priv}(\mathcal{A})$ and $\mathit{DVisit}^{\neg priv}(\mathcal{A})$ in the following).

6.1 PRELIMINARY DEFINITIONS

6.1.1 *Extending the timed automaton and parametric timed automaton definitions*

For all the content of the Part II, we consider a special location, called "private location" and denoted ℓ_{priv}, in each TA and PTA.

Therefore, we extend the Definitions 3.5 and 3.11 to have $\mathcal{A} = (\Sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, I, E)$ and $\mathcal{P} = (\Sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, \mathbb{P}, I, E)$.

6.1.2 *Special sets of execution times: $\mathit{DVisit}^{priv}(\mathcal{A})$ and $\mathit{DVisit}^{\neg priv}(\mathcal{A})$*

Let us introduce two key concepts to define our notion of opacity.

Given a TA $\mathcal{A}$ and a run ρ, we say that ℓ_{priv} is *reached on the way to ℓ_f 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 $\ell_m = \ell_{priv}$, $\ell_n = \ell_f$ and $\forall 0 \leq i \leq n-1, \ell_i \neq \ell_f$. We denote by $Visit^{priv}(\mathcal{A})$ the set of those runs, and refer to them as *private* runs. We denote by $DVisit^{priv}(\mathcal{A})$ the set of all the durations of these runs.

Conversely, we say that ℓ_{priv} is *avoided on the way to ℓ_f 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 $\ell_n = \ell_f$ and $\forall 0 \leq i < n, \ell_i \notin \{\ell_{priv}, \ell_f\}$. We denote the set of those runs by $Visit^{\neg priv}(\mathcal{A})$, referring to them as *public* runs, and by $DVisit^{\neg priv}(\mathcal{A})$ the set of all the durations of these public runs.

Therefore, $DVisit^{priv}(\mathcal{A})$ (resp. $DVisit^{\neg priv}(\mathcal{A})$) is the set of all the durations of the runs for which ℓ_{priv} is reachable (resp. avoided) on the way to ℓ_f.

These concepts can be seen as the set of execution times from the initial location ℓ_0 to the final location ℓ_f while visiting (resp. not visiting) a private location ℓ_{priv}. Observe that, from the definition of the duration of a run, this "execution time" does not include the time spent in ℓ_f.

> **Example 6.1.** *Consider again the TA in Figure 3.1 on page 28, now considering that ℓ_2 is the private location. We have $DVisit^{priv}(\mathcal{A}) = [1, 2]$ and $DVisit^{\neg priv}(\mathcal{A}) = [0, 3]$.*

6.2 COMPUTING $DVisit^{priv}(\mathcal{A})$ AND $DVisit^{\neg priv}(\mathcal{A})$

We must be able to express the set of execution times, i. e., the durations of all runs from the initial location to the final location. While the problem of expressing the set of execution times seems very natural for TAs, it seems to be barely addressed in the literature, with the exception of [BDR08; Ros19].

6.2.1 *The RA arithmetic*

We use the RA arithmetic, which is the set of first-order formulae, interpreted over the real numbers, of $\langle \mathbb{R}, +, <, \mathcal{N}, 0, 1 \rangle$ where $\mathcal{N}$ is a unary predicate such that $\mathcal{N}(z)$ is true *iff* z is a natural number. This arithmetic has a decidable theory with a complexity of 3EXPTIME [Wei99].

6.2.2 *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 $\rho = (\mathfrak{s}_i)_i$ in $\mathfrak{T}_{\mathcal{A}}$ such that $dur(\rho) = d$ and the associated path $\pi(\rho) = (r_k)_{0\leq k\leq K}$ in the region graph $\mathcal{RG}_\mathcal{A}$ satisfies $r_0 = r, r_K = r'$. It is shown in [BDR08, 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 (i) $d = m$ (ii) $\exists z, \mathcal{N}(z) \wedge d = m + cz$ (iii) $m < d < m + 1$, and (iv) $\exists z, \mathcal{N}(z) \wedge m + cz < d < m + cz + 1$, where $c, m \in \mathbb{N}$.

Let us give the main idea of the proof presented in [BDR08] (even though this explanation is not necessary to follow our reasoning for the computation of execution times of TAs). The idea of the proof of [BDR08] is to consider the region automaton of $\mathcal{A}$. For this construction, we consider the particular 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 $\mathfrak{T}_\mathcal{A}$ corresponds to a run ρ^0 of $\mathfrak{T}_{\mathcal{A}^0}$ (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 $\mathcal{RG}_{\mathcal{A}^0}$, where the only action a denotes the reset of x_0 (all other transitions are labeled with the silent action): this is the region automaton of $\mathcal{A}$ as defined in Definition 3.10. The conclusion follows because $t \in \lambda_{r,r'}$ if $\lfloor t \rfloor$ is the length of a path in the deterministic automaton obtained from $\mathcal{RA}_\mathcal{A}$ by subset construction.

Lemma 6.1 (Reachability-duration computation). *The sets $DVisit^{priv}(\mathcal{A})$ and $DVisit^{\neg priv}(\mathcal{A})$ are computable and definable in RA arithmetic.*

Proof. Let $\mathcal{A}$ be a TA. We aim at reducing the computation of the sets $DVisit^{priv}(\mathcal{A})$ and $DVisit^{\neg priv}(\mathcal{A})$ to the computation of sets $\lambda_{r,r'}$ for some regions r and r'.

First, let us compute $DVisit^{priv}(\mathcal{A})$. 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 $b = \text{false}$ and ℓ_f with $b = \text{true}$ can be seen as two different locations. Then, we set $b \leftarrow \text{true}$ on any transition whose target location is ℓ_{priv}; therefore, $b = \text{true}$ denotes that ℓ_{priv} has been visited. We denote by ℓ'_{f_true} the final state of $\mathcal{A}'$ where $b = \text{true}$. $DVisit^{priv}(\mathcal{A})$ is exactly the set of execution times in $\mathcal{A}'$ between ℓ_0 and ℓ'_{f_true}. For all the regions r'_i associated to ℓ'_{f_true}, we can compute (using [BDR08]) λ_{r,r'_i}, where r is the region associated to ℓ_0 in $\mathcal{A}'$. Therefore, $DVisit^{priv}(\mathcal{A})$ 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 $DVisit^{\neg priv}(\mathcal{A})$. We define another TA $\mathcal{A}''$ obtained from $\mathcal{A}$ by deleting all the transitions leading to ℓ_{priv}. Therefore, the set of execution times reaching ℓ_f in $\mathcal{A}''$ is exactly the set of execution times reaching ℓ_f in $\mathcal{A}$ associated to runs not visiting ℓ_{priv}.

$DVisit^{\neg priv}(\mathcal{A})$ 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, $DVisit^{priv}(\mathcal{A})$ can be computed as the union of all the λ_{r,r'_i}. □

The computation of these sets of execution times $DVisit^{\neg priv}(\mathcal{A})$ and $DVisit^{priv}(\mathcal{A})$ will be used in the three subsequent chapters dealing with ET-opacity problems in TAs. Particularly, we use these results to perform the computation of the execution times for which a system matches the definition of ET-opaque.

7 GUARANTEEING EXECUTION-TIME OPACITY USING PARAMETRIC TIMED MODEL CHECKING

That which we call a rose
by any other name would smell as sweet.

— Shakespeare

In this chapter, we investigate problems considering a notion of opacity (namely ET-opacity, for execution-time opacity): we consider that the attacker has access (only) to the system execution time. We address the following 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 visited the private location. We also consider a full notion of ET-opacity, asking whether the system is ET-opaque for all execution times. We show that these problems are decidable for TAs but become undecidable when one adds parameters (yielding PTAs). Finally, we devise an algorithm for synthesizing PTA parameter valuations guaranteeing that the resulting TA is ∃-ET-opaque.

Motivation

Among harmful information leaks, the *timing information leakage* is the ability for an attacker to deduce internal information depending on timing information. Franck Cassez proposed in [Cas09] a first definition of timed opacity for TAs: 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. As surveyed in Section 1.3, two main lines of works study some restrictions of this problem, to obtain decidability results. It was done by restricting the models to RTAs in [WZ18; WZA18] and to time-bounded opacity (where the execution time of the system is bounded) in [AETYM21].

In this chapter, 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. We are interested in system opacity w.r.t. the execution time: a TA is ET-opaque if an attacker cannot deduce private information only knowing the execution time of a run of the system.

Our system formalism is much more expressive (and therefore able to encode richer applications) than in [WZ18; WZA18] because we consider the full class of TAs instead of the restricted RTAs. We also consider parametric extensions, not discussed in [Cas09; WZ18; WZA18; AETYM21].

Contributions of the chapter for TAs

We consider the following version of ET-opacity: given a TA with a private location denoting the execution of some secret behavior, the TA is ET-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 visiting the private location, and another run *not* visiting the private location. That is, for this particular execution time, the system is ET-opaque if one cannot deduce whether the system visited the private location. Such a notion of ET-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 ET-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 three problems based on this notion of ET-opacity:

1. an emptiness problem: the decision of the (non) existence of an execution time for which the system is ET-opaque (referred to as $\exists$-ET-opaque);

2. a computation problem: the computation of the set of possible execution times for which the system is ET-opaque; and

3. a decision problem: whether the TA is ET-opaque for all execution times (referred to as fully ET-opaque).

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 of the chapter for PTAs

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 therefore becomes PTAs.

On the theoretical side, we answer an existential parametric version of the aforementioned problems, that is, the existence of (at least) one parameter valuation for which the associated TA is ∃-ET-opaque (resp. fully ET-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 and a private location, synthesize the timing parameters and the execution times for which one cannot deduce whether the system visited 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 of the chapter

To sum up, this chapter proposes the following contributions:

1. a notion of ET-opacity, and a notion of full ET-opacity for TAs;
2. a procedure to solve the ET-opacity t-computation problem for TAs, and a procedure to answer the full ET-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 ET-opaque;
5. a set of experiments on a set of benchmarks, including PTAs translations from Java programs.

The contributions of this chapter were originally presented in [ALMS22]. Since this publication, the notions introduced here were renamed to clarify some concepts. In Appendix B, we present the correspondence with the previous namings.

Organization of the chapter

Section 7.1 introduces the problem and Section 7.2 addresses ET-opacity for TAs. We then address the parametric version of ET-opacity, with theory studied in Sections 7.3 and 7.4, algorithmic in Section 7.5 and experiments in Section 7.6. Section 7.7 concludes the chapter.

7.1 Execution-time opacity problems

7.1.1 *Definitions*

We now introduce the concept of "ET-opacity for a set of durations (or execution times) D": a system is ET-opaque for execution times D whenever, for any duration in D, it is not possible to deduce whether the system visited 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 7.1 (Execution-time opacity (ET-opacity) for D)**.** Given a TA $\mathcal{A}$ with a private location ℓ_{priv}, and a set of execution times D, we say that $\mathcal{A}$ is *execution-time opaque (ET-opaque) for execution times D* if $D \subseteq DVisit^{priv}(\mathcal{A}) \cap DVisit^{\neg priv}(\mathcal{A})$.

In the following, we will be interested in the existence of such an execution time. We say that a TA is $\exists$-ET-opaque if it is ET-opaque for a non-empty set of execution times.

Definition 7.2 ($\exists$-ET-opacity)**.** Given a TA $\mathcal{A}$, we say that $\mathcal{A}$ is *$\exists$-ET-opaque* if $DVisit^{priv}(\mathcal{A}) \cap DVisit^{\neg priv}(\mathcal{A}) \neq \emptyset$.

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 ET-opaque for all execution times. In other words, if a system is *fully* *ET-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 7.3 (Full ET-opacity)**.** Given a TA $\mathcal{A}$, we say that $\mathcal{A}$ is *fully ET-opaque* if $DVisit^{priv}(\mathcal{A}) = DVisit^{\neg priv}(\mathcal{A})$.

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

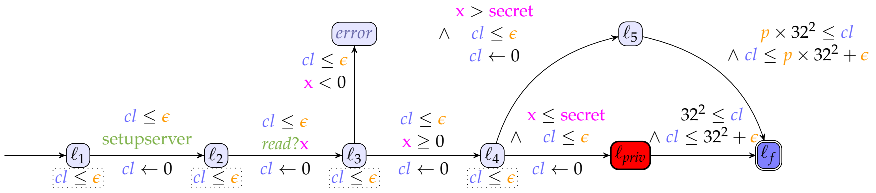


Figure 7.1: A Java program encoded in a PTA

Remark 5. *This definition is symmetric: a system is not fully ET-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 fully ET-opaque w.r.t. Definition 7.3. This case will be defined as weak ET-opacity, introduced and studied with an expiring secrecy in Chapter 8.*

■

Example 7.1. *Consider the PTA $\mathcal{P}$ in Figure 7.1 where cl is a clock, while ϵ, p are parameters. We use a slightly 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 *and IMITATOR.*

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[a], 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 C.*

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 C, 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, $DVisit^{priv}(v_1(\mathcal{P})) = [1024, 1029]$ while $DVisit^{\neg priv}(v_1(\mathcal{P})) = [2048, 2053]$. Therefore, $v_1(\mathcal{P})$ is ET-opaque for execution times $D = [1024, 1029] \cap [2048, 2053] = \emptyset$.

Let v_2 such that $v_2(\epsilon) = 2$ and $v_2(p) = 1.002$. $DVisit^{priv}(v_2(\mathcal{P})) = [1024, 1034]$ while $DVisit^{\neg priv}(v_2(\mathcal{P})) = [1026.048, 1036.048]$.

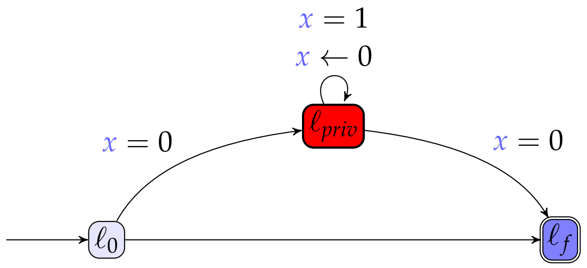


Figure 7.2: TA for which the set of execution times ensuring execution-time opacity is $\mathbb{N}$

Therefore, $v_2(\mathcal{P})$ is ET-opaque for execution times $D = [1026.048, 1034]$.
Obviously, neither $v_1(\mathcal{P})$ nor $v_2(\mathcal{P})$ are fully ET-opaque.

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

7.1.2 *Decision and computation problems*

Computation problem for execution-time opacity

We can now define the ET-opacity t-computation problem, which consists in computing the possible execution times ensuring ET-opacity. In other words, the attacker model is as follows: the attacker knows the system model in the form of a TA, and can only observe the execution time between the start of the program and the time it reaches the final location.

The ET-opacity t-computation problem:
INPUT: A TA $\mathcal{A}$
PROBLEM: Compute the execution times D such that $\mathcal{A}$ is ET-opaque for these execution times D.

Let us illustrate that this computation problem is certainly not easy. For the TA $\mathcal{A}$ in Figure 7.2, the execution times D for which $\mathcal{A}$ is ET-opaque is exactly $\mathbb{N}$; that is, only integer times ensure ET-opacity (as the system can only leave ℓ_{priv} and hence enter ℓ_f at an integer time), while non-integer rational times violate ET-opacity.

Decision problem for ∃-ET-opacity

We can also define the ∃-ET-opacity decision problem, which consists in answering whether a TA is ∃-ET-opaque.

The $\exists$-ET-opacity decision problem:
Input: A TA $\mathcal{A}$
Problem: Is $\mathcal{A}$ $\exists$-ET-opaque?

Decision problem for full ET-opacity

We can also define the full ET-opacity decision problem, which consists in answering whether a TA is fully ET-opaque.

The full ET-opacity decision problem:
Input: A TA $\mathcal{A}$
Problem: Is $\mathcal{A}$ fully ET-opaque?

7.2 Execution-time opacity problems for timed automata

In this section, we address the non-parametric problems defined in Section 7.1.2, i. e., the ET-opacity t-computation problem (Section 7.2.1), $\exists$-ET-opacity decision problem (Section 7.2.2) and full ET-opacity decision problem (Section 7.2.3). We show that both problems can be solved using a construction of the *DVisit* sets based on the RA arithmetic (which was recalled in Section 6.2).

7.2.1 *Answering the ET-opacity t-computation problem*

Proposition 7.1 (Solvability of ET-opacity t-computation problem)**.** *The ET-opacity t-computation problem is solvable for TAs.*

Proof. Let $\mathcal{A}$ be a TA. From Lemma 6.1, we can compute and define in RA arithmetic the sets $DVisit^{\neg priv}(\mathcal{A})$ and $DVisit^{priv}(\mathcal{A})$.

By the decidability of RA arithmetic, the intersection of these sets is computable. Then, the set $D = DVisit^{priv}(\mathcal{A}) \cap DVisit^{\neg priv}(\mathcal{A})$ is effectively computable. □

This positive result can be put in perspective with the negative result of [Cas09] that proves that it is undecidable whether a TA (and even the more restricted subclass of ERAs 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 [Cas09].

7.2.2 *Checking for ∃-ET-opacity*

Proposition 7.2 (Decidability of ∃-ET-opacity decision problem)**.** *The ∃-ET-opacity decision problem is decidable for TAs.*

Proof. Let $\mathcal{A}$ be a TA. From Lemma 6.1, we can compute and define in RA arithmetic the sets $DVisit^{\neg priv}(\mathcal{A})$ and $DVisit^{priv}(\mathcal{A})$.

From the decidability of RA arithmetic, the emptiness of the intersection between these sets is decidable. Therefore, answering $DVisit^{priv}(\mathcal{A}) \cap DVisit^{\neg priv}(\mathcal{A}) \overset{?}{=} \emptyset$ is decidable. □

7.2.3 *Checking for full ET-opacity*

Proposition 7.3 (Decidability of full ET-opacity decision problem)**.** *The full ET-opacity decision problem is decidable for TAs.*

Proof. Let $\mathcal{A}$ be a TA. From Lemma 6.1, we can compute and define in RA arithmetic the sets $DVisit^{\neg priv}(\mathcal{A})$ and $DVisit^{priv}(\mathcal{A})$.

From the decidability of RA arithmetic, the equality between these sets is decidable. Therefore, answering $DVisit^{priv}(\mathcal{A}) \overset{?}{=} DVisit^{\neg priv}(\mathcal{A})$ is decidable. □

From [Wei99] and [BDR08, 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 5EXPTIME, which is an upper-bound for the problem complexity. Note that, as in [BDR08], we did not compute a lower bound for the complexity of Propositions 7.1 to 7.3. However, Theorem 8.1 will show that this problem is solvable in NEXPTIME (see Remark 11). The exact complexity remains to be done as future work.

Remark 6. *Remark 11 in the next chapter will also extend this work with a notion of* weakness*, not only asking for the equality between public and private execution times, but allowing a model to be weakly ET-opaque if $DVisit^{priv}(\mathcal{A}) \subseteq DVisit^{\neg priv}(\mathcal{A})$. This will be formally defined and studied in Chapter 8.*

■

Example 7.2. *Consider again the PTA $\mathcal{P}$ in Figure 3.2 and consider v' such that $v'(p_1) = v'(p_2) = 2$. Recall from Example 6.1 that $DVisit^{priv}(v(\mathcal{P})) = [1,3]$ and $DVisit^{\neg priv}(v(\mathcal{P})) = [2,3]$. Thus,*

$DVisit^{priv}(v(\mathcal{P})) \neq DVisit^{\neg priv}(v(\mathcal{P}))$ and therefore $\mathcal{A}$ is not *fully ET-opaque.*
Now, consider v' such that $v'(p_1) = v'(p_2) = 1.5$. This time, $DVisit^{priv}(v'(\mathcal{P})) = DVisit^{\neg priv}(v'(\mathcal{P})) = [1.5, 3]$ and therefore $v'(\mathcal{P})$ is fully ET-opaque.

7.3 THE THEORY OF PARAMETRIC ∃-ET-OPACITY

We now address in this section the parametric problems of ∃-ET-opacity.

7.3.1 *Problem definitions*

Emptiness problem for ∃-ET-opacity

Let us consider the following decision problem, i. e., the problem of checking the *emptiness* of the set of parameter valuations guaranteeing ∃-ET-opacity. The decision problem associated to full ET-opacity will be considered in Section 7.4.

The ∃-ET-opacity p-emptiness problem:
INPUT: A PTA $\mathcal{P}$
PROBLEM: Decide the emptiness of the set of parameter valuations v such that $v(\mathcal{P})$ is ∃-ET-opaque.

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

Synthesis problem for ∃-ET-opacity

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

The ∃-ET-opacity p-synthesis problem:
INPUT: A PTA $\mathcal{P}$
PROBLEM: Synthesize the parameter valuations v such that $v(\mathcal{P})$ is ∃-ET-opaque.

7.3.2 *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

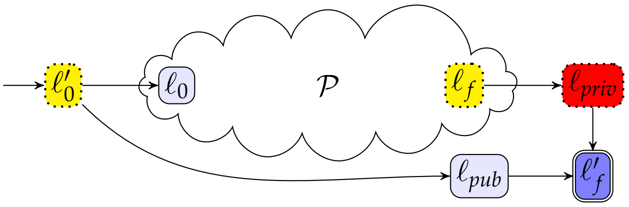


Figure 7.3: Reduction from reachability emptiness

are obviously undecidable for larger numbers, but the decidability is open for smaller numbers. This will be briefly discussed in Section 7.7.

With the rule of thumb that all non-trivial decision problems are undecidable for general PTAs [And19b], the following result is not surprising, and follows from the undecidability of reachability emptiness for PTAs.

Theorem 7.1 (Undecidability of ∃-ET-opacity p-emptiness problem)**.** *The ∃-ET-opacity p-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., [AHV93; Mil00; Doy07; JLR15; BBLS15]). Consider an arbitrary PTA $\mathcal{P}$ with initial location ℓ_0 and final 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 [Min67], so one can see $\mathcal{P}$ as an encoding of a 2-counter machine).

Now, add the following locations and transitions (all unguarded) as in Figure 7.3: a new urgent[1] 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 $\mathcal{P}'$ 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 visiting ℓ_{pub} and not visiting ℓ_{priv}. That is, for all v, $DVisit^{\neg priv}(v(\mathcal{P}')) = [0, \infty)$.

Assume there exists some parameter valuation v such that ℓ_f is reachable from ℓ_0 in $v(\mathcal{P})$ 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 $v(\mathcal{P}')$ for the exact same execution times D. Therefore, $v(\mathcal{P}')$ is ET-opaque for execution times D.

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

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

Remark 7. *Our proof reduces from the reachability-emptiness problem, for which several undecidability proofs were proposed (notably [AHV93; Mil00; Doy07; JLR15; BBLS15]), with various flavors (numbers of parameters, integer- or dense-time, integer- or rational-valued parameters, etc.). See [And19b] for a survey. Notably, this means (from [BBLS15]) that Theorem 7.1 holds for PTAs with at least 3 clocks and a single parameter.*

■

Since the emptiness problem is undecidable, the synthesis problem is immediately intractable as well.

Corollary 7.1. *The ∃-ET-opacity p-synthesis problem is unsolvable for general PTAs.*

7.3.3 *A decidable subclass*

We now show that the ∃-ET-opacity p-emptiness problem is decidable for L/U-PTAs. Despite early positive results for L/U-PTAs [HRSV02; BL09], more recent results (notably [JLR15; AL17a; ALR18; ALM20]) mostly proved undecidable properties of L/U-PTAs, and therefore this positive result is welcome.

Theorem 7.2 (Decidability of ∃-ET-opacity p-emptiness problem)**.** *The ∃-ET-opacity p-emptiness problem is decidable for L/U-PTAs.*

Proof. We reduce to the ET-opacity t-computation problem of a given TA, which is decidable (Proposition 7.1).

Let $\mathcal{P}_{\mathcal{LU}}$ be an L/U-PTA. Let $\mathcal{P}_{0,\infty}$ 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. $\mathcal{P}_{0,\infty}$ is therefore a TA.

Let us show that the set of valuations v such that $v(\mathcal{P}_{\mathcal{LU}})$ is ∃-ET-opaque is non empty *iff* the solution to the ET-opacity t-computation problem for $\mathcal{P}_{0,\infty}$ is non-empty.

- $\Rightarrow$ Assume there exists a valuation v such that $v(\mathcal{P}_{\mathcal{LU}})$ is $\exists$-ET-opaque. Therefore, the solution to the ET-opacity t-computation problem for $\mathcal{P}_{0,\infty}$ 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 avoided on the way to ℓ_f.

  Therefore, from Lemma 3.2, the runs of $v(\mathcal{P}_{\mathcal{LU}})$ of duration d such that ℓ_{priv} is reachable (resp. avoided) on the way to ℓ_f are also runs of $\mathcal{P}_{0,\infty}$. Therefore, there exists a non-empty set of durations such that $\mathcal{P}_{0,\infty}$ is ET-opaque, i. e., solution to the ET-opacity t-computation problem for $\mathcal{P}_{0,\infty}$ is non-empty.

- $\Leftarrow$ Assume the solution to the ET-opacity t-computation problem for $\mathcal{P}_{0,\infty}$ 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 $\mathcal{P}_{0,\infty}$, and there exists a run of duration d such that ℓ_{priv} is avoided on the way to ℓ_f in $\mathcal{P}_{0,\infty}$.

  The result could follow immediately—if only assigning 0 and ∞ to parameters was a proper parameter valuation. From [HRSV02; BL09], 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, given v assigns 0 to lower-bound and C to upper-bound parameters, this run exists in $v(\mathcal{P}_{\mathcal{LU}})$. 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 $d+1$ 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 $v(\mathcal{P}_{\mathcal{LU}})$, and there exists a run of duration d such that ℓ_{priv} is avoided on the way to ℓ_f in $v(\mathcal{P}_{\mathcal{LU}})$. Therefore, the set of valuations v such that $v(\mathcal{P}_{\mathcal{LU}})$ is $\exists$-ET-opaque is non empty—which concludes the proof.

□

Remark 8. *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 [HRSV02], e. g., a toy railroad crossing model and a model of Fischer mutual exclusion protocol given in [AHV93] (see [And19b] for a survey). Even though the PTA in Figure 7.1 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).*

■

7.3.4 *Intractability of synthesis for lower/upper parametric timed automata*

Even though the ∃-ET-opacity p-emptiness problem is decidable for L/U-PTAs (Theorem 7.2), the *synthesis* of the parameter valuations remains intractable in general, as shown in the following Proposition 7.4. 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 decide "the emptiness of the valuation set of the computed solution intersected with an equality test between variables" cannot be used to represent the solution. For example, let us question whether we could represent the solution of the ∃-ET-opacity p-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. PPL can typically compute the answer to this question. Therefore, from the following Proposition 7.4, finite unions of polyhedra cannot be used to represent the solution of the ∃-ET-opacity p-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 [JLR15, Theorem 2]).

Proposition 7.4 (Intractability of ∃-ET-opacity p-synthesis problem)**.** *If it can be computed, the solution to the ∃-ET-opacity p-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 [JLR15, Theorem 2], and we reduce from the undecidability of the ∃-ET-opacity p-emptiness problem for general PTAs (Theorem 7.1). Assume the solution of the ∃-ET-opacity p-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 $\mathcal{P}$ with notably two locations ℓ_{priv} and ℓ_f. From $\mathcal{P}$, we define an L/U-PTA $\mathcal{P}_{\mathcal{LU}}$ 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 ∃-ET-opacity p-synthesis problem

$$\Gamma = \{v \mid v(\mathcal{P}_{\mathcal{LU}}) \text{ is } \exists\text{-ET-opaque}\}$$

for $\mathcal{P}_{\mathcal{LU}}$ 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 $\{v \in \Gamma \mid \bigwedge_i v({p_i}^u) = v({p_i}^l)\}$ (which is decidable by assumption), we can decide the $\exists$-ET-opacity p-emptiness problem for the PTA $\mathcal{P}$ (which is undecidable from Theorem 7.1). This leads to a contradiction, and therefore the solution of the $\exists$-ET-opacity p-synthesis 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. □

7.4 THE THEORY OF PARAMETRIC FULL ET-OPACITY

We address here the following decision problem, which asks about the emptiness of the parameter valuation set guaranteeing full ET-opacity. We also define the full ET-opacity p-synthesis problem, synthesizing the timing parameters guaranteeing full ET-opacity.

7.4.1 *Problem definitions*

The full ET-opacity p-emptiness problem:
INPUT: A PTA $\mathcal{P}$
PROBLEM: Decide the emptiness of the set of parameter valuations v such that $v(\mathcal{P})$ is fully ET-opaque.

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

We also define the *full ET-opacity p-synthesis problem*, aiming at synthesizing (ideally the entire set of) parameter valuations v for which $v(\mathcal{P})$ is fully ET-opaque.

The full ET-opacity p-synthesis problem:
INPUT: A PTA $\mathcal{P}$
PROBLEM: Synthesize the parameter valuations v such that $v(\mathcal{P})$ is fully ET-opaque.

7.4.2 *Undecidability in general*

Considering that Theorem 7.1 shows the undecidability of the $\exists$-ET-opacity p-emptiness problem, the undecidability of the full ET-opacity p-emptiness problem is not surprising, but does not follow immediately.

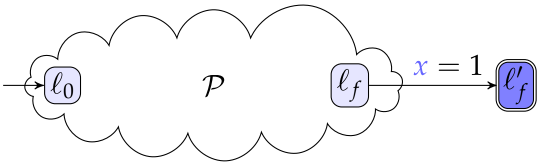


Figure 7.4: Undecidability of reachability-emptiness over constant time

To prove this result (that will be stated formally in Theorem 7.3), we first need the following lemma stating that the reachability 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 $\mathcal{P}$, a target location ℓ_{target} and a time bound T, it is undecidable whether the set of parameter valuations for which there exists a run reaching ℓ_{target} 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 [ALM20, Theorem 3.12], the authors showed that the reachability-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_{target}, it is undecidable whether the set of parameter valuations for which at least one run reaches ℓ_{target} 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 $\mathcal{P}$ with a location ℓ_{target}. We define a PTA $\mathcal{P}'$ as in Figure 7.4 by adding a new location ℓ'_f, and a transition from ℓ_{target} to ℓ'_f guarded by $x = 1$, where x is a new clock (initially 0), not used in $\mathcal{P}$ and therefore never reset in the automaton.

Let us show that there is no valuation such that ℓ_{target} is reachable in at most 1 time unit *iff* there is no valuation such that ℓ'_f is reachable exactly in 1 time unit.

- $\Leftarrow$ Assume reachability emptiness holds in constant time for $\mathcal{P}'$, i. e., there exists no parameter valuation for which ℓ'_f is reachable in $T = 1$ time units. Then, from the construction of $\mathcal{P}'$, no parameter valuation exists for which ℓ_{target} is reachable in $T \leq 1$ time units.
- $\Rightarrow$ Conversely, assume reachability emptiness holds over bounded time for $\mathcal{P}$, i. e., there exists no parameter valuation for which ℓ_{target} is reachable in $T \leq 1$ time units. Then, from the con-

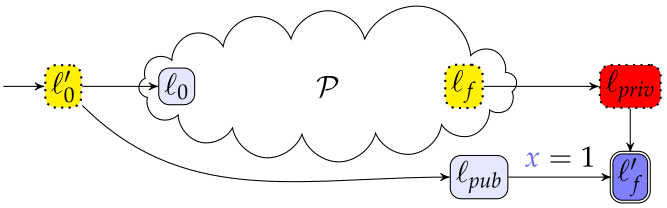


Figure 7.5: Reduction from reachability-emptiness for the proof of Theorem 7.3

struction of $\mathcal{P}'$, 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 7.3.

Theorem 7.3 (Undecidability of the full ET-opacity p-emptiness problem)**.** *The full ET-opacity p-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 $\mathcal{P}$ with (at least) 4 clocks and 2 parameters, with initial location ℓ_0 and a final location ℓ_f. We add the following locations and transitions in $\mathcal{P}$ to obtain a PTA $\mathcal{P}'$, as in Figure 7.5: (*i*) a new urgent initial location ℓ'_0, with outgoing transition to ℓ_0 and to a new location ℓ_{pub}, (*ii*) a new urgent location ℓ_{priv} with an incoming transition from ℓ_f, (*iii*) a new urgent and final location ℓ'_f with incoming transitions from ℓ_{priv} and ℓ_{pub}, and (*iv*) 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 visiting ℓ_{pub} and not visiting ℓ_{priv}. That is, for all v, $\mathit{DVisit}^{\neg priv}(v(\mathcal{P}')) = \{1\}$.

We show that there exists a valuation v such that $v(\mathcal{P}')$ is fully ET-opaque (with ℓ_{priv} as private location, and ℓ'_f as final location) iff there exists a valuation v such that ℓ_f is reachable in $v(\mathcal{P})$ for execution time equal to 1.

$\Leftarrow$ Assume there exists some valuation v such that ℓ_f is reachable from ℓ_0 in $\mathcal{P}$ (only) for execution time equal to 1. Then, due to our construction, ℓ_{priv} is reachable on the way to ℓ'_f in $v(\mathcal{P}')$ for the exact same execution time 1. Therefore, $v(\mathcal{P}')$ is fully ET-opaque.

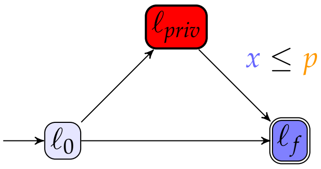


Figure 7.6: $\mathcal{P}_{0,\infty}$ is not sufficient for the full ET-opacity p-emptiness problem

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

Therefore, there exists a valuation v such that $v(\mathcal{P}')$ is fully ET-opaque iff there exists a valuation v such that ℓ_f is reachable in $v(\mathcal{P})$ 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 7.3, 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. □

Since the emptiness problem is undecidable, the synthesis problem is immediately intractable as well.

Corollary 7.2. *The full ET-opacity p-synthesis problem is unsolvable for PTAs with (at least) 4 clocks and 2 parameters.*

7.4.3 *Undecidability for lower/upper parametric timed automata*

Note that reasoning like in Section 7.3.3, i. e., reducing the full ET-opacity p-emptiness problem to a full ET-opacity decision problem of the non-parametric $\mathcal{P}_{0,\infty}$, is not relevant. Figure 7.6 shows an L/U-PTA $\mathcal{P}_{\mathcal{LU}}$ (and more precisely, an upper parametric timed automaton (U-PTA) [BL09], i. e., an L/U-PTA with an empty set of lower-bound parameters) which is not fully ET-opaque for any parameter valuation, but whose associated TA $\mathcal{P}_{0,\infty}$ is.

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 3.2, we can show in the following example that this is not the case for full ET-opacity.

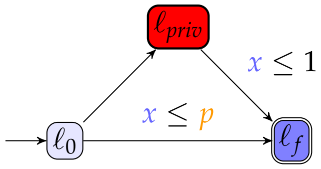


Figure 7.7: No monotonicity for full ET-opacity in L/U-PTAs

Example 7.3. *Consider the PTA in Figure 7.7. First assume v such that $v(p) = 0.5$. Then, $v(\mathcal{P})$ is not fully ET-opaque: indeed, ℓ_f can be reached in 1 time unit via ℓ_{priv}, but not without visiting ℓ_{priv}.*
Second, assume v' such that $v'(p) = 1$. Then, $v'(\mathcal{P})$ is fully ET-opaque: indeed, ℓ_f can be reached for any duration in $[0, 1]$ by runs both visiting and not visiting ℓ_{priv}.
Finally, let us enlarge p further, and assume v'' such that $v''(p) = 2$. Then, $v''(\mathcal{P})$ becomes again not fully ET-opaque: indeed, ℓ_f can be reached in 2 time units not visiting ℓ_{priv}, but cannot be reached in 2 time units by visiting ℓ_{priv}.
As a side note, remark that this PTA is actually a a U-PTA, that is, monotonicity for this problem does not even hold for U-PTAs.

In fact, we show that, while the ∃-ET-opacity p-emptiness problem is decidable for L/U-PTAs (Theorem 7.2), the full ET-opacity p-emptiness problem becomes undecidable for this same class (from 4 parameters). This confirms (after previous works in [JLR15; AL17a; ALR18]) that L/U-PTAs stand at the frontier between decidability and undecidability.

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

Proof. Let us recall from [ALM20, Theorem 3.12] that the reachability-emptiness problem is undecidable over bounded time for PTAs with (at least) 3 clocks and 2 parameters. Assume a PTA $\mathcal{P}$ with 3 clocks and 2 parameters, say p_1 and p_2, and a final location ℓ_f. Take 1 as a time bound. From [ALM20, 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 [JLR15; ABPP19], we "split" each of the two parameters used in $\mathcal{P}$ 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. $x \geq p_i$)

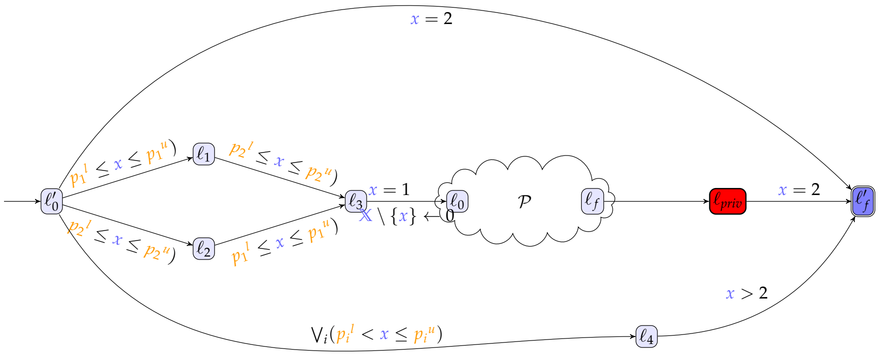

$x = 2$
$p_1{}^l \leq x \leq p_1{}^u)$
ℓ_1
$p_2{}^l \leq x \leq p_2{}^u)$
ℓ'_0
ℓ_3
$x = 1$
$\mathbb{X} \setminus \{x\} \leftarrow 0$
ℓ_0
$\mathcal{P}$
ℓ_f
ℓ_{priv}
$x = 2$
ℓ'_f
$p_2{}^l \leq x \leq p_2{}^u)$
ℓ_2
$p_1{}^l \leq x \leq p_1{}^u)$
$x > 2$
$\bigvee_i(p_i{}^l < x \leq p_i{}^u)$
ℓ_4

Figure 7.8: Undecidability of full ET-opacity p-emptiness problem for L/U-PTAs

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 $\mathcal{P}$ 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 [ALM20, Theorem 3.12].

Now, consider the extension of $\mathcal{P}$ given in Figure 7.8, and let $\mathcal{P}'$ be this extension. We assume that x is an extra clock not used in $\mathcal{P}$. The syntax "$\mathbb{X} \setminus \{x\} \leftarrow 0$" denotes that all clocks of the original PTA $\mathcal{P}$ 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 visiting location ℓ_{priv};
2. the original automaton $\mathcal{P}$ 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 [ALM20, Theorem 3.12], it is undecidable whether there exists a parameter valuation for which there exists a run reach-

ing 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 $\mathcal{P}$ (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 $\mathcal{P}'$ is not fully ET-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 visiting ℓ_{priv}. Since no run can reach ℓ'_f while visiting ℓ_{priv} for a duration $\neq 2$, then again $\mathcal{P}'$ is not fully ET-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 $\mathcal{P}$ (with duplicate parameters) is exactly the one of the original $\mathcal{P}$. 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 $\mathcal{P}$ 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 $\mathcal{P}'$ is fully ET-opaque for such a parameter valuation.

   Conversely, assume there exists no parameter valuation for which there exists a run of $\mathcal{P}$ of duration ≤ 1 reaching ℓ_f. In that case, $\mathcal{P}'$ is not fully ET-opaque for any parameter valuation.

As a consequence, there exists a parameter valuation v' for which $v'(\mathcal{P}')$ is fully ET-opaque *iff* there exists a parameter valuation v for which there exists a run in $v(\mathcal{P})$ of duration ≤ 1 reaching ℓ_f—which is undecidable from [ALM20, Theorem 3.12]. □

As the emptiness problems are undecidable, the synthesis problems are immediately intractable as well.

Corollary 7.3. *The full ET-opacity p-synthesis problem is unsolvable for L/U-PTAs with (at least) with (at least) 4 clocks and 4 parameters.*

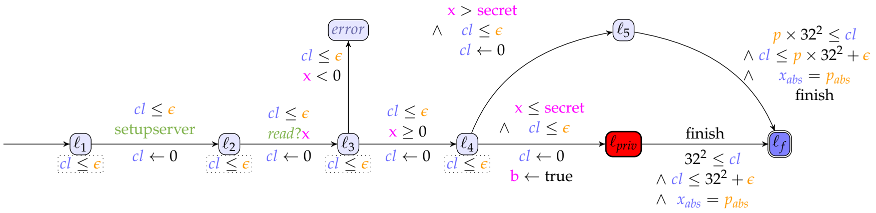


Figure 7.9: Transformed version of Figure 7.1

7.5 PARAMETER SYNTHESIS FOR EXECUTION-TIME OPACITY

Despite the negative theoretical result of Theorem 7.1, we now address the $\exists$-ET-opacity p-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 $\mathsf{EFsynth}$ on ℓ_f with contradictory values of b.

We detail each operation in the following.

In this section, we assume a PTA $\mathcal{P}$.

7.5.1 *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 7.1). Therefore, $b = \text{true}$ 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 $\mathsf{Enrich}(\mathcal{P})$ denote this procedure.

Example 7.4. *Figure 7.9 shows the transformed version of the PTA in Figure 7.1.*

7.5.2 *Self-composition*

We use here the principle of *self-composition* [Tri98; BDR11], i. e., composing the PTA with a copy of itself. More precisely, given a PTA $\mathcal{P}' = \mathsf{Enrich}(\mathcal{P})$, we first perform an identical copy of $\mathcal{P}'$ *with*

Algorithm 7.1: Formalization of the procedure $\mathsf{SynthOp}(\mathcal{P})$

```
input  : A PTA $\mathcal{P}$ with parameters set $\mathbb{P}$
output: Parameter constraint $K$ over $\mathbb{P} \cup \{p_{abs}\}$
$\mathcal{P}' \leftarrow \mathsf{Enrich}(\mathcal{P})$
$\mathcal{P}'' \leftarrow \mathcal{P}' \parallel_{\{\text{finish}\}} \mathsf{Copy}(\mathcal{P}')$
return $\mathsf{EFsynth}\Big(\mathcal{P}'', \big\{(\ell_f \wedge b = \text{true}, \ell_f^c \wedge b^c = \text{false})\big\}\Big)$
```

distinct variables: that is, a clock x of $\mathcal{P}'$ is distinct from a clock x in the copy of $\mathcal{P}'$—which can be trivially performed using variable renaming.[2] Let $\mathsf{Copy}(\mathcal{P}')$ denote this copy of $\mathcal{P}'$. We then compute $\mathcal{P}' \parallel_{\{\text{finish}\}} \mathsf{Copy}(\mathcal{P}')$. That is, $\mathcal{P}'$ and $\mathsf{Copy}(\mathcal{P}')$ 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.

7.5.3 *Synthesis*

Then, we apply reachability synthesis $\mathsf{EFsynth}$ (over all parameters, i. e., the "internal" timing parameters, but also the p_{abs} parameter) to the following goal location: the original $\mathcal{P}'$ is in ℓ_f with $b = \text{true}$ while its copy $\mathsf{Copy}(\mathcal{P}')$ is in ℓ_f^c with $b^c = \text{false}$, where variables with a c as exponent 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 $\mathsf{SynthOp}(\mathcal{P})$ denote the entire procedure. We formalize $\mathsf{SynthOp}$ in Algorithm 7.1, 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 7.5.1.

> **Example 7.5.** *Consider again the PTA $\mathcal{P}$ in Figure 7.1: its enriched version $\mathcal{P}'$ is given in Figure 7.9. Fix $v(\epsilon) = 1$, $v(p) = 2$. We then perform the synthesis applied to the self-composition of $\mathcal{P}'$ according to Algorithm 7.1. The result obtained with* IMITATOR *is: $p_{abs} = \emptyset$ (as expected from Example 7.1).*
> *Now fix $v(\epsilon) = 2$, $v(p) = 1.002$. We obtain: $p_{abs} \in [1026.048, 1034]$ (again, as expected from Example 7.1).*

[2] 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.

Now let us keep all parameters unconstrained. The result of Algorithm 7.1 is the following 3-dimensional constraint:

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

7.5.4 *Correctness*

We will state below that, whenever $\mathsf{SynthOp}(\mathcal{P})$ terminates, then its result is an exact (sound and complete) answer to the ∃-ET-opacity p-synthesis problem. Recall that this problem aims to, given a PTA $\mathcal{P}$, synthesize the set of parameters valuations v s.t. $v(\mathcal{P})$ is ∃-ET-opaque.

Let us first prove a technical lemma used later to prove the soundness of $\mathsf{SynthOp}$.

Lemma 7.2. *Assume* $\mathsf{SynthOp}(\mathcal{P})$ *terminates with result* K. *For all* $v \models K$, *there exists a run ending in* ℓ_f *at time* $v(p_{abs})$ *in* $v(\mathcal{P})$.

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 $v(\mathcal{P})$. □

We can now prove soundness and completeness.

Proposition 7.5 (Soundness of $\mathsf{SynthOp}$)**.** *Assume* $\mathsf{SynthOp}(\mathcal{P})$ *terminates with result* K. *For all* $v \models K$, *there exists a run of duration* $v(p_{abs})$ *such that* ℓ_{priv} *is visited on the way to* ℓ_f *in* $v(\mathcal{P})$ *and there exists a run of duration* $v(p_{abs})$ *such that* ℓ_{priv} *is avoided on the way to* ℓ_f *in* $v(\mathcal{P})$.

Proof. $\mathsf{SynthOp}(\mathcal{P})$ is the result of $\mathsf{EFsynth}$ called on the self-composition of $\mathsf{Enrich}(\mathcal{P})$. Recall that Enrich has enriched $\mathcal{P}$ 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 3.1, there exists a run of $\mathcal{P}'' = \mathcal{P}' \parallel_{\{\text{finish}\}} \mathsf{Copy}(\mathcal{P}')$ reaching $\ell_f \wedge b = \text{true}, \ell'_f \wedge b' = \text{false}$. From Lemma 7.2, 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 $v(\mathcal{P})$ and there exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is avoided on the way to ℓ_f in $v(\mathcal{P})$. □

Proposition 7.6 (Completeness of SynthOp)**.** *Assume* $\mathsf{SynthOp}(\mathcal{P})$ *terminates with result K. Assume v s.t. there exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is reachable on the way to ℓ_f in $v(\mathcal{P})$ and there exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is avoided on the way to ℓ_f in $v(\mathcal{P})$. Then $v \models K$.*

Proof. Assume $\mathsf{SynthOp}(\mathcal{P})$ 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 $v(\mathcal{P})$ and there exists a run ρ' of duration $v(p_{abs})$ such that ℓ_{priv} is avoided on the way to ℓ_f in $v(\mathcal{P})$.

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 $v(\mathcal{P})$ implies that there exists a run ρ of duration $v(p_{abs})$ such that $\ell_f \land b = \mathrm{true}$ (resp. $b = \mathrm{false}$) is reachable in $v(\mathsf{Enrich}(\mathcal{P}))$.

Since our self-composition allows any interleaving, runs ρ of $v(\mathcal{P}')$ and ρ' in $v(\mathsf{Copy}(\mathcal{P}'))$ 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 $\ell_f \land b = \mathrm{true}, \ell'_f \land b' = \mathrm{false}$ in $v(\mathcal{P}'')$.

Finally, from Lemma 3.1, $v \models K$. □

Theorem 7.5 (Correctness of SynthOp)**.** *Assume* $\mathsf{SynthOp}(\mathcal{P})$ *terminates with result K and let v be a parameter valuation. 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 $v(\mathcal{P})$ and there exists a run of duration $v(p_{abs})$ such that ℓ_{priv} is avoided on the way to ℓ_f in $v(\mathcal{P})$.*
2. *$v \models K$.*

Proof. From Propositions 7.5 and 7.6 □

7.6 EXPERIMENTS: ENSURING ∃-ET-OPACITY

This experimental section is taken from an earlier version [AS19]. We present here algorithms to solve ET-opacity t-computation problem (computing the execution times for which a given TA is ET-opaque) and ∃-ET-opacity p-synthesis problem (synthesizing parameter valuations ensuring that the valuated PTA is ∃-ET-opaque).

7.6.1 *Experimental environment*

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.[3]

7.6.2 *Translating programs into parametric timed automata*

We will consider case studies from the PTA community and from previous works focusing on privacy using (P)TA. 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 TA 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, [LYGY10]. 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.[4] 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.

7.6.3 *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 Figure 7.1, we turn them to parameters (i. e., unknown constants),

[3] Sources, models and results are available at doi.org/10.5281/zenodo.3251141 and imitator.fr/static/ATVA19/.

[4] https://github.com/Apogee-Research/STAC/

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) TAs 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 7.6. *For example, in Figure 7.10 (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 7.7. *For example, setting b to true on the upper transition from ℓ_1 to ℓ_2 in Figure 7.10, the answer to the ET-opacity t-computation problem is $D = (30, \infty)$, and the system is therefore not fully ET-opaque since ℓ_2 can be reached for any execution time in $[0, \infty)$.*

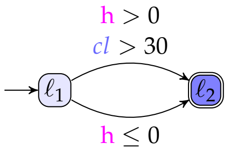


Figure 7.10: [VNN18, Fig. 5]

7.6.4 *Experiments*

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 [GMR07; BCLR15; VNN18], also addressing non-interference or opacity in TAs[5]. In addition, we used two common models from the PTA 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) [HRSV02]. 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 7.6.2. 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 PTAs is orthogonal to our work, and is the object of future works.

The ET-opacity t-computation problem

First, we *verified* whether a given TA model is fully ET-opaque, i. e., if for all execution times reaching the final location, both an execution visits the private location and an execution does not visit this private location. To this end, we also answer the ET-opacity t-computation problem, i. e., to synthesize all execution times for which the system is ET-opaque. While this problem can be verified on the region graph (Proposition 7.1), we use the same framework as in Section 7.5 (solving

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

Table 7.1: Experiments: ET-opacity t-computation problem

Model			Transf. PTA			Result		
Name	$\lvert\mathcal{P}\rvert$	$\lvert\mathbb{X}\rvert$	$\lvert\mathcal{P}\rvert$	$\lvert\mathbb{X}\rvert$	$\lvert\mathbb{P}\rvert$	States	Time (s)	Opaque?
[VNN18, Fig. 5]	1	1	2	3	3	13	0.02	(×)
[GMR07, Fig. 1b]	1	1	2	3	1	25	0.04	(×)
[GMR07, Fig. 2a]	1	1	2	3	1	41	0.05	(×)
[GMR07, Fig. 2b]	1	1	2	3	1	41	0.02	(×)
Web privacy problem [BCLR15]	1	2	2	4	1	105	0.07	(×)
Coffee	1	2	2	5	1	43	0.05	√
Fischer-HSRV02	3	2	6	5	1	2495	5.83	(×)
STAC:1:n			2	3	6	65	0.12	(×)
STAC:1:v			2	3	6	63	0.11	×
STAC:3:n			2	3	8	289	0.72	√
STAC:3:v			2	3	8	287	0.74	(×)
STAC:4:n			2	3	8	904	6.40	×
STAC:4:v			2	3	8	19183	265.52	×
STAC:5:n			2	3	6	144	0.24	√
STAC:11A:v			2	3	8	5037	47.77	(×)
STAC:11B:v			2	3	8	5486	59.35	(×)
STAC:12c:v			2	3	8	1177	18.44	×
STAC:12e:n			2	3	8	169	0.58	×
STAC:12e:v			2	3	8	244	1.10	(×)
STAC:14:n			2	3	8	1223	22.34	(×)

the ∃-ET-opacity p-synthesis problem), 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 7.1. We give from left to right the model name, the numbers of automata and of clocks in the original TA (this information is not relevant for Java programs as the original model is not a TA), the numbers of automata[6], of clocks and of parameters in the transformed PTA, the computation time in seconds (for the ET-opacity t-computation problem), and the result. In the result column, "√" (resp. "×") denotes that the model is fully ET-opaque (resp. is not fully ET-opaque, i. e., vulnerable), while "(×)" denotes that the model is not fully ET-opaque but ∃-ET-opaque, so could be fixed at a lower cost. That is, although $\mathit{DVisit}^{priv}(v(\mathcal{P})) \neq \mathit{DVisit}^{\neg priv}(v(\mathcal{P}))$, their intersection is non-empty and therefore, by tuning the execution time, it may be possible to make the system fully ET-opaque. This will be discussed in Section 7.6.5.

Even though we are interested here in ET-opacity t-computation problem (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 dif-

[6] As usual, it may be simpler to write PTA models as a network of PTAs. Recall from Definition 3.13 that a network of PTAs gives a PTA. In this case, $\lvert\mathcal{P}\rvert$ denotes the number of input PTA components.

ferent from p_{abs}. In addition, the number of parameters is increased by the parameters encoding the symbolic variables (such as x and secret in Figure 7.1).

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 ET-opacity t-computation problem for practical case studies, exhibiting which execution times ensure ET-opacity, and whether *all* execution times indeed guarantee ET-opacity (full ET-opacity decision problem).

In many cases, while the system is not fully ET-opaque, we are able to *infer* the execution times guaranteeing ET-opacity (cells marked "$(\times)$"). 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 $DVisit^{priv}(v(\mathcal{P}))$ and $DVisit^{\neg priv}(v(\mathcal{P}))$ will lead our algorithm to answer "not fully ET-opaque", while some methods may not be able to differentiate this mismatch from imprecision or 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 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.

The ∃-ET-opacity p-synthesis problem

Then, we address the ∃-ET-opacity p-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 ET-opacity t-computation problem; 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 ET-opacity. Note that it may not be that easy to tune a Java program to make it ET-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 7.2, where the columns are similar to Table 7.1. A difference is that the first $|\mathbb{P}|$ column denotes the number

Table 7.2: Experiments: full ET-opacity p-synthesis problem

Model				Transf. PTA			Result		
Name	$\lvert\mathcal{P}\rvert$	$\lvert\mathbb{X}\rvert$	$\lvert\mathbb{P}\rvert$	$\lvert\mathcal{P}\rvert$	$\lvert\mathbb{X}\rvert$	$\lvert\mathbb{P}\rvert$	States	Time (s)	Constraint
[VNN18, Fig. 5]	1	1	0	2	3	4	13	0.02	K
[GMR07, Fig. 1b]	1	1	0	2	3	3	25	0.03	K
[GMR07, Fig. 2]	1	1	0	2	3	3	41	0.05	K
Web privacy problem [BCLR15]	1	2	2	2	4	3	105	0.07	K
Coffee	1	2	3	2	5	4	85	0.10	$\top$
Fischer-HSRV02	3	2	2	6	5	3	2495	7.53	K
STAC:3:v			2	2	3	9	361	0.93	K

of parameters in the original model (without counting these added by our transformation). In addition, Table 7.2 does not contain a "opaque?" 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 ∃-ET-opacity ("⊥"), all of them ("⊤"), or some of them ("K").

DISCUSSION An interesting outcome is that the computation timed needed to solve this synthesis is comparable to the (non-parametric) ET-opacity t-computation problem, 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 ET-opaque. Also note that our method always terminates for these models, and therefore the result exhibited is complete. Interestingly, Coffee is ∃-ET-opaque for any valuation of the 3 internal parameters.

7.6.5 *"Repairing" a non-execution-time opaque parametric timed automaton*

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.

7.7 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 ∃-ET-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.

SUMMARY In Table 7.3, we present all the decidability results introduced in this chapter. We denote a problem with a green check if it is decidable (or solvable), a red cross if it is undecidable (or unsolvable).

Table 7.3: Summary of the results for execution-time opacity [ALMS22]

		∃-ET-opaque	**fully ET-opaque**
Decision	TA	√(Proposition 7.2)	√(Proposition 7.3)
p-emptiness	L/U-PTA	√(Theorem 7.2)	×(Theorem 7.4)
	PTA	×(Theorem 7.1)	×(Theorem 7.3)
p-synthesis	L/U-PTA	×(Proposition 7.4)	×(Corollary 7.3)
	PTA	×(Corollary 7.1)	×(Corollary 7.2)

Perspectives

Theory

We proved decidability of the ET-opacity t-computation problem (Proposition 7.1), of the ∃-ET-opacity decision problem (Proposition 7.2) and of the full ET-opacity decision problem (Proposition 7.3) 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 full ET-opacity p-emptiness problem (Theorem 7.1) remains open: that is, is the ∃-ET-opacity p-synthesis problem decidable for PTAs using a single clock? Our method in Section 7.5 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 [AHV93]. 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 [And19b], despite recent advances, notably over discrete time [BO14; GH21]. The 1-clock question also remains open for full ET-opacity p-emptiness problem (Theorem 7.3). The minimum number of parameters required for our proof of the undecidability of the full ET-opacity p-emptiness problem for PTAs (resp. L/U-PTAs) to work is 2 (resp. 4), as seen in Theorem 7.3 (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 ∃-ET-opacity p-emptiness problem (Theorem 7.2), the undecidability of the full ET-opacity p-emptiness problem (Theorem 7.4) and the intractability of ∃-ET-opacity p-synthesis problem (Proposition 7.4). It remains open whether these results still apply to the more restrictive class of U-PTAs.

Applications

We did not propose algorithm to ensure full ET-opacity, and particularly to solve the full ET-opacity p-synthesis problem (which is shown to be undecidable, Corollaries 7.2 and 7.3). Indeed, the construction used to solve ∃-ET-opaque problems cannot be easily extended to solve their full version. Exhibiting such algorithms (or semi-algorithms) remains as a future work.

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 TAs 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).

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.

Finally, we may be interested in close other problems, extending our notion to take into consideration other paradigms. For example, this definition of ET-opacity may be not sufficient to formalize side-channels attacks based on the status of the memory: a secret remains private, no matter when it happened. In the following chapter, we will consider an extension of our ET-opacity with an expiration date of the secret, i. e., an expiration bound after which knowing the secret is deemed useless.

8 EXPIRING EXECUTION-TIME OPACITY PROBLEMS IN PARAMETRIC TIMED AUTOMATA

When did everything start having
an expiration date?

— Wong Kar-wai

In this chapter, we introduce a new notion of ET-opacity with expiring secrecy. We prove some decidability results over different problems for TAs and PTAs.

Motivation

In Chapter 7, we considered an attacker who had access only to the execution times: this is ET-opacity. We considered an existential version of this definition (asking for the existence of an execution times ensuring ET-opacity) but also a full version (asking if a system is ET-opaque for all execution times).

In [AETYM21], the authors consider a time-bounded notion of the opacity of [Cas09], where the attacker has to disclose the secret before an upper bound, using a partial observability. This can be seen as a secrecy with an *expiration date*. The rationale is that retrieving a secret "too late" is useless; this is understandable, e. g., when the secret depend of the status of the memory; if the cache was overwritten since, then knowing the secret is probably useless in most situations. In addition, the analysis is carried over a time-bounded horizon; this means there are two time bounds in [AETYM21]: one for the secret expiration date, and one for the bounded-time execution of the system. In this chapter, we will incorporate this secret expiration date into our notion of ET-opacity: we will only consider the former one (the secret expiration date), and lift the assumption regarding the latter (the bounded-time execution of the system).

Contributions of the chapter

In this chapter, we consider an *expiring version of ET-opacity*, where the secret is subject to an expiration date; this can be seen as a combination of both concepts from Chapter 7 and [AETYM21]. That is, we consider that an attack is successful only when the attacker can decide that the secret location was entered less than Δ time units before the system completion. Conversely, if the attacker exhibits an execution time d for which it is certain that the secret location was visited, but this location was entered strictly more than Δ time units prior to the system completion, then this attack is useless, and can be seen as a failed attack. The system is therefore *fully* exp-ET-opaque if the set of execution times for which the private location was entered within Δ time units prior to system completion is exactly equal to the set of execution times for which the private location was either not visited or entered $> \Delta$ time units prior to system completion.

In addition, when the former (secret) set of execution times is *included* into the latter (non-secret) set of times, we say that the system is *weakly* exp-ET-opaque; this encodes situations when the attacker might be able to deduce that no secret location was visited, but is not able to confirm that the secret location *was* indeed visited.

On the one hand, our attacker model is *less powerful* than [AETYM21], because our attacker has only access to the execution time (and to the input model); in that sense, our attacker capability is identical to Chapter 7. On the other hand, we lift the time-bounded horizon analysis from [AETYM21], allowing to analyze systems without any assumption on their execution time; therefore, we only import from [AETYM21] the notion of *expiring secret*.

We first consider exp-ET-opacity for TAs. We show that it is possible to:

1. decide whether a TA is fully (resp. weakly) exp-ET-opaque w.r.t. a given time bound Δ (decision problem);
2. decide whether a TA is fully (resp. weakly) exp-ET-opaque w.r.t. at least one bound Δ (emptiness problem);
3. compute the set of time bounds (or expiration dates) for which a TA is weakly exp-ET-opaque (computation problem).

Second, we show that, in PTAs, the emptiness of the parameter valuation sets for which the system is fully (resp. weakly) exp-ET-opaque is undecidable, even for the L/U-PTA subclass of PTAs, so far known for its decidability results.

Organization of the chapter

We define exp-ET-opacity problems in Section 8.1. We address problems for TAs in Section 8.2, and parametric extensions in Section 8.3. We conclude in Section 8.4.

8.1 Expiring execution-time opacity problems

In this section, we formally introduce the problems we address in this chapter. In the following, let $\mathcal{A}$ be a TA.

Let $\mathbb{N}^\infty = \mathbb{N} \cup \{+\infty\}$ and $\mathbb{R}^\infty_{\geq 0} = \mathbb{R}_{\geq 0} \cup \{+\infty\}$.

8.1.1 *Expiring execution-time opacity*

Let us first recall some notations introduced in Section 6.1.2. We denote by $\mathit{Visit}^{\mathit{priv}}(\mathcal{A})$ the set of private runs (i. e., runs visiting the private location) and by $\mathit{DVisit}^{\mathit{priv}}(\mathcal{A})$ the set of all the durations of these runs. Conversely, we denote by $\mathit{Visit}^{\neg\mathit{priv}}(\mathcal{A})$ the set of *public* runs, and by $\mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A})$ the set of all the durations of these runs.

Given a TA $\mathcal{A}$ and a finite run ρ in $\mathfrak{T}_{\mathcal{A}}$, the *duration* between two states of $\rho : \mathfrak{s}_0, (d_0, e_0), \mathfrak{s}_1, \cdots, \mathfrak{s}_k$ is $\mathit{dur}_\rho(\mathfrak{s}_i, \mathfrak{s}_j) = \sum_{i \leq m \leq j-1} d_m$. We also define the *duration* between two locations ℓ_1 and ℓ_2 as the duration $\mathit{dur}_\rho(\ell_1, \ell_2) = \mathit{dur}_\rho(\mathfrak{s}_i, \mathfrak{s}_j)$ with $\rho : \mathfrak{s}_0, (d_0, e_0), \mathfrak{s}_1, \cdots, \mathfrak{s}_i, \cdots, \mathfrak{s}_j, \cdots, \mathfrak{s}_k$ where $\mathfrak{s}_j$ the first occurrence of a state with location ℓ_2 and $\mathfrak{s}_i$ is the last state of ρ with location ℓ_1 before $\mathfrak{s}_j$. We choose this definition to coincide with the definitions of opacity that we will define in the following Definition 8.1. Indeed, we want to make sure that revealing a secret (ℓ_1 in this definition) is not a failure if it is done after a given time. Thus, as soon as the system reaches its final state (ℓ_2), we will be interested in knowing how long the secret has been present, and thus the last time it was entered ($\mathfrak{s}_i$).

Given $\Delta \in \mathbb{R}^\infty_{\geq 0}$, we define $\mathit{Visit}^{\mathit{priv}}_{>\Delta}(\mathcal{A})$ (resp. $\mathit{Visit}^{\mathit{priv}}_{\leq\Delta}(\mathcal{A})$) as the set of runs $\rho \in \mathit{Visit}^{\mathit{priv}}(\mathcal{A})$ s.t. $\mathit{dur}_\rho(\ell_{\mathit{priv}}, \ell_f) > \Delta$ (resp. $\mathit{dur}_\rho(\ell_{\mathit{priv}}, \ell_f) \leq \Delta$). We refer to the runs of $\mathit{Visit}^{\mathit{priv}}_{\leq\Delta}(\mathcal{A})$ as *secret* runs.

We define below two notions of ET-opacity w.r.t. a time bound Δ. We will compare two sets:

1. the set of execution times for which the private location was entered at most Δ time units prior to system completion; and

2. the set of execution times for which either the private location was not visited at all, or it was last entered more than Δ time units prior to system completion (which, in our setting, is somehow similar to *not* visiting the private location, in the sense that entering it "too early" is considered of little interest).

If both sets match, the system is fully $(\leq \Delta)$-ET-opaque. If the former is included into the latter, then the system is weakly $(\leq \Delta)$-ET-opaque.

Definition 8.1 (Expiring execution-time opacity)**.** Given a TA $\mathcal{A}$ and a bound (i. e., an expiration date for the secret) $\Delta \in \mathbb{R}_{\geq 0}^{\infty}$ we say that $\mathcal{A}$ is *fully exp-ET-opaque* w.r.t. the expiration date Δ, denoted *fully* $(\leq \Delta)$*-ET-opaque*, if

$$DVisit^{priv}_{\leq\Delta}(\mathcal{A}) = DVisit^{priv}_{>\Delta}(\mathcal{A}) \cup DVisit^{\neg priv}(\mathcal{A}).$$

Moreover, we say that $\mathcal{A}$ is *weakly exp-ET-opaque* w.r.t. the expiration date Δ, denoted *weakly* $(\leq \Delta)$*-ET-opaque*, if

$$DVisit^{priv}_{\leq\Delta}(\mathcal{A}) \subseteq DVisit^{priv}_{>\Delta}(\mathcal{A}) \cup DVisit^{\neg priv}(\mathcal{A}).$$

Remark 9. *Our notion of* weak *exp-ET-opaque may still leak some information: on the one hand, if a run indeed enters the private location* $\leq \Delta$ *time units before system completion, there exists an equivalent run not visiting it (or entering it earlier), and therefore the system is exp-ET-opaque;* but *on the other hand, there may exist execution times for which the attacker can deduce that the private location was* not *entered* $\leq \Delta$ *before system completion. This remains acceptable in some cases, and this motivates us to define a weak version of exp-ET-opacity. Also note that the "initial-state opacity" for RTAs considered in [WZ18] can also be seen as* weak *in the sense that their language inclusion is also unidirectional.*

∎

Example 8.1. *Consider again the PTA in Figure 3.2; let v be such that $v(p_1) = 1$ and $v(p_2) = 2.5$. Fix $\Delta = 1$.*
We have:

- $DVisit^{\neg priv}(v(\mathcal{P})) = [0, 3]$
- $DVisit^{priv}_{>\Delta}(v(\mathcal{P})) = (2, 2.5]$
- $DVisit^{priv}_{\leq\Delta}(v(\mathcal{P})) = [1, 2.5]$

Therefore, we say that $v(\mathcal{P})$ is:

- *weakly* (≤ 1)*-ET-opaque, as* $[1, 2.5] \subseteq \big((2, 2.5] \cup [0, 3]\big)$
- *not fully* (≤ 1)*-ET-opaque, as* $[1, 2.5] \neq \big((2, 2.5] \cup [0, 3]\big)$

As introduced in Remark 9, despite the weak (≤ 1)*-ET-opacity of* $\mathcal{A}$*, the attacker can deduce some information about the visit of the private location for some execution times. For example, if a run has a duration of 3 time units, it cannot be a private run, and therefore the attacker can deduce that the private location was not visited at all.*

8.1.2 *Problems*

We define six different problems in the context of (non-parametric) TAs:

The full (resp. weak) exp-ET-opacity decision problem:
Input: A TA $\mathcal{A}$ and a bound $\Delta \in \mathbb{R}_{\geq 0}^{\infty}$
Problem: Decide whether $\mathcal{A}$ is fully (resp. weakly) $(\leq \Delta)$-ET-opaque.

The full (resp. weak) exp-ET-opacity Δ-emptiness problem:
Input: A TA $\mathcal{A}$
Problem: Decide the emptiness of the set of bounds Δ such that $\mathcal{A}$ is fully (resp. weakly) $(\leq \Delta)$-ET-opaque.

The full (resp. weak) exp-ET-opacity Δ-computation problem:
Input: A TA $\mathcal{A}$
Problem: Compute the maximal set $\mathcal{D}$ of bounds such that $\mathcal{A}$ is fully (resp. weakly) $(\leq \Delta)$-ET-opaque w.r.t. all $\Delta \in \mathcal{D}$.

Example 8.2. *Consider again the PTA in Figure 3.2; let v be such that $v(p_1) = 1$ and $v(p_2) = 2.5$ (as in Example 8.1).*

- *Given $\Delta = 1$, the weak exp-ET-opacity decision problem asks whether $v(\mathcal{P})$ is weakly $(\leq \Delta)$-ET-opaque—the answer is "yes" from Example 8.1.*
- *The weak exp-ET-opacity Δ-emptiness problem is therefore "no" because the set of bounds Δ such that $v(\mathcal{P})$ is weakly $(\leq \Delta)$-ET-opaque is not empty.*
- *Finally, the weak exp-ET-opacity Δ-computation problem asks to compute all the corresponding bounds: in this example, the solution is $\Delta \in \mathbb{R}_{\geq 0}^{\infty}$.*

Relations with the execution-time opacity problems

Note that, when considering $\Delta = +\infty$, $DVisit^{priv}_{>\Delta}(\mathcal{A}) = \emptyset$ and all the execution times of runs visiting ℓ_{priv} are in $DVisit^{priv}_{\leq\Delta}(\mathcal{A})$. Therefore, full $(\leq +\infty)$-ET-opacity matches full ET-opacity. We can therefore notice that answering the full exp-ET-opacity decision problem for $\Delta = +\infty$ is decidable (Proposition 7.3). However, the emptiness and computation problems cannot be reduced to full ET-opacity problems from Chapter 7.

Conversely, it is possible to answer the full ET-opacity decision problem by checking the full exp-ET-opacity decision problem with $\Delta = +\infty$. Moreover, ET-opacity t-computation problem reduces to full exp-ET-opacity Δ-computation problem: if $+\infty \in \mathcal{D}$, we get the answer.

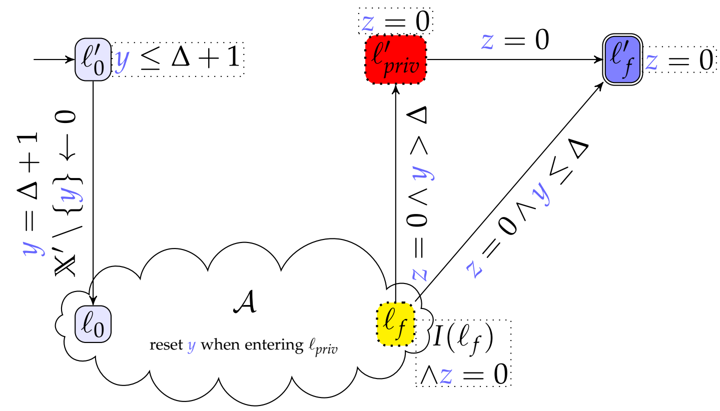


Figure 8.1: Construction used in Proposition 8.1

8.2 Expiring execution-time opacity in timed automata

In this section, we consider the six problems defined previously on TAs. In general, the link between the full and weak notions of the three aforementioned problems is not obvious. However, for a fixed value of Δ, we establish the following result.

Proposition 8.1 (Reduction from full exp-ET-opacity decision problem to weak exp-ET-opacity decision problem)**.** *The full exp-ET-opacity decision problem reduces to the weak exp-ET-opacity decision problem.*

Proof. Fix a TA $\mathcal{A}$ and a time bound $\Delta \in \mathbb{R}_{\geq 0}^{\infty}$.

In this reduction, we build a new TA $\mathcal{A}'$ where secret and non-secret runs are swapped. More precisely, we add a new clock y that measures how much time has elapsed since the latest entrance of the private location. It is thus reset whenever we enter the private location ℓ_{priv}. This clock is initialized to value $\Delta + 1$ (which can be ensured by waiting in a new initial location ℓ'_0 for $\Delta + 1$ time units before going to the original initial location ℓ_0 and resetting every clock but y). When reaching the final location ℓ_f, one can urgently (a new clock z can be used to force the system to move immediately) move to a new secret location ℓ'_{priv} if $y > \Delta$ and then to the new final location ℓ'_f; otherwise (if $y \leq \Delta$), the TA can go directly to the new final location ℓ'_f. Therefore, a run that would not be secret (as $y > \Delta$) is now secret and reciprocally. Then, by testing weak $(\leq \Delta)$-ET-opacity of both $\mathcal{A}$ and $\mathcal{A}'$, one can check full $(\leq \Delta)$-ET-opacity of $\mathcal{A}$.

We give a graphical representation of our construction in Figure 8.1. Formally, given a TA $\mathcal{A} = \left(\Sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, I, E\right)$ and $\Delta \in \mathbb{R}_{\geq 0}^{\infty}$, we build a second TA $\mathcal{A}' = \left(\Sigma \cup \{\sharp\}, L', \ell'_0, \ell'_{priv}, \ell'_f, \mathbb{X} \cup \{y, z\}, I', E'\right)$ where $\sharp$ denotes a special action absent from Σ and where:

- $L' = L \cup \left\{\ell'_0, \ell'_{priv}, \ell'_f\right\}$;
- $\forall \ell \in L \setminus \{\ell_f\} : I'(\ell) = I(\ell)$; $I'(\ell_f) = (I(\ell_f) \land z = 0)$; $I'(\ell'_0) = (y \leq \Delta + 1)$; $I'(\ell'_{priv}) = (z = 0)$; $I'(\ell'_f) = (z = 0)$.
- for each $(\ell, g, a, R, \ell') \in E$, we add $\left(\ell, g, a, R', \ell'\right)$ to E' where $R' = R \cup \{y, z\}$ if $\ell' = \ell_{priv}$ and $R' = R \cup \{z\}$ otherwise.

  We also add the following edges to E':

  - $\left(\ell'_0, (y = \Delta + 1), \sharp, \mathbb{X}, \ell_0\right)$;
  - $\left(\ell_f, (z = 0 \land y > \Delta), \sharp, \emptyset, \ell'_{priv}\right)$;
  - $\left(\ell_f, (z = 0 \land y \leq \Delta), \sharp, \emptyset, \ell'_f\right)$;
  - $\left(\ell'_{priv}, (z = 0), \sharp, \emptyset, \ell'_f\right)$.

There is a one-to-one correspondence between the secret (resp. non-secret) runs ending in ℓ_{priv} in $\mathcal{A}$ and the non-secret (resp. secret) runs ending in ℓ'_{priv} in $\mathcal{A}'$. Given ρ a run in $\mathcal{A}$ and ρ' the corresponding run in $\mathcal{A}'$, we have $dur(\rho') = dur(\rho) + \Delta + 1$ (where $\Delta + 1$ is the time waited in ℓ'_0).

Recall from Definition 8.1 the definition of weak $(\leq \Delta)$-ET-opacity for $\mathcal{A}'$: $DVisit^{priv}_{\leq\Delta}(\mathcal{A}') \subseteq DVisit^{priv}_{>\Delta}(\mathcal{A}') \cup DVisit^{\neg priv}(\mathcal{A}')$.

1. First consider the left-hand part "$DVisit^{priv}_{\leq\Delta}(\mathcal{A}')$": these execution times correspond to runs of $\mathcal{A}'$ for which ℓ'_{priv} was entered less than Δ (and actually 0) time units prior to reaching ℓ'_f. These runs passed the $y > \Delta$ guard between ℓ_f and ℓ'_{priv}.

   From our construction, these runs correspond to runs of the original $\mathcal{A}$ either not visiting ℓ_{priv} at all (since y was never reset since its initialization to $\Delta + 1$, and therefore $y \geq \Delta + 1 > \Delta$), or to runs which visited ℓ_{priv} more than Δ time units before reaching ℓ_f. Therefore, $DVisit^{priv}_{\leq\Delta}(\mathcal{A}') = \left\{d + 1 + \Delta \mid d \in DVisit^{priv}_{>\Delta}(\mathcal{A}) \cup DVisit^{\neg priv}(\mathcal{A})\right\}$

2. Second, consider the right-hand part "$DVisit^{priv}_{>\Delta}(\mathcal{A}') \cup DVisit^{\neg priv}(\mathcal{A}')$": the set $DVisit^{priv}_{>\Delta}(\mathcal{A}')$ is necessarily empty, as any run of $\mathcal{A}'$ visiting ℓ'_{priv} reaches ℓ'_f immediately in 0-time. The execution times from $DVisit^{\neg priv}(\mathcal{A}')$ correspond to runs of $\mathcal{A}'$ *not* visiting ℓ'_{priv}, therefore for which only the guard $y \leq \Delta$ holds. Hence, they correspond to runs of $\mathcal{A}$ which entered ℓ_{priv} less than Δ time units prior to reaching ℓ_f. Therefore, $DVisit^{priv}_{>\Delta}(\mathcal{A}') \cup DVisit^{\neg priv}(\mathcal{A}') = \left\{d + 1 + \Delta \mid d \in DVisit^{priv}_{\leq\Delta}(\mathcal{A})\right\}$

To conclude, checking that $\mathcal{A}'$ is weakly $(\leq \Delta)$-ET-opaque (i. e., $DVisit^{priv}_{\leq\Delta}(\mathcal{A}') \subseteq DVisit^{priv}_{>\Delta}(\mathcal{A}') \cup DVisit^{\neg priv}(\mathcal{A}')$) is equivalent to

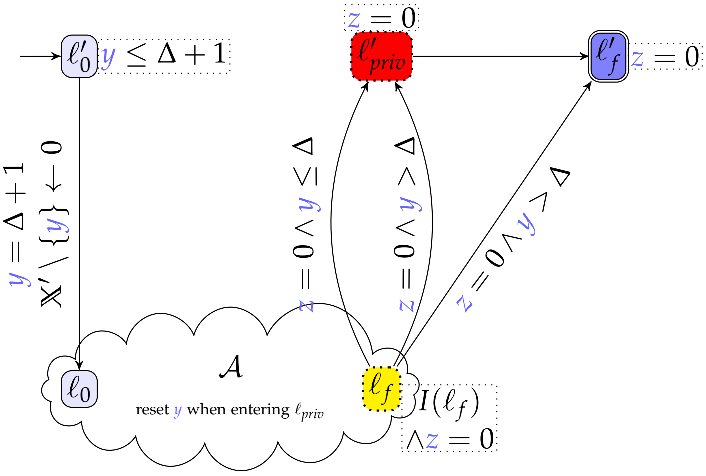


Figure 8.2: Construction used in Remark 10

$DVisit^{priv}_{>\Delta}(\mathcal{A}) \cup DVisit^{\neg priv}(\mathcal{A}) \subseteq DVisit^{priv}_{\leq\Delta}(\mathcal{A})$. Moreover, from Definition 8.1, checking that $\mathcal{A}$ is weakly ($\leq \Delta$)-ET-opaque denotes checking $DVisit^{priv}_{\leq\Delta}(\mathcal{A}) \subseteq DVisit^{priv}_{>\Delta}(\mathcal{A}) \cup DVisit^{\neg priv}(\mathcal{A})$.

Therefore, checking that both $\mathcal{A}'$ and $\mathcal{A}$ are weakly ($\leq \Delta$)-ET-opaque denotes $DVisit^{priv}_{\leq\Delta}(\mathcal{A}) = DVisit^{priv}_{>\Delta}(\mathcal{A}) \cup DVisit^{\neg priv}(\mathcal{A})$, which is the definition of full ($\leq \Delta$)-ET-opacity for $\mathcal{A}$.

To conclude, $\mathcal{A}$ is fully ($\leq \Delta$)-ET-opaque *iff* $\mathcal{A}$ and $\mathcal{A}'$ are weakly ($\leq \Delta$)-ET-opaque. □

Remark 10. *We can similarly establish the opposite reduction.*

Given a TA $\mathcal{A}$, we build a second TA $\mathcal{A}'$ differing from the automaton created in the proof of Proposition 8.1 only in the transitions exiting ℓ_f. We need to consider the following transitions (depicted in Figure 8.2):

- $(\ell_f, (z = 0 \land y \leq \Delta), \sharp, \emptyset, \ell'_{priv})$;
- $(\ell_f, (z = 0 \land y > \Delta), \sharp, \emptyset, \ell'_f)$;
- $(\ell_f, (z = 0 \land y > \Delta), \sharp, \emptyset, \ell'_{priv})\}$.

This construction ensures that the runs which were secret in $\mathcal{A}$ correspond to secret runs of $\mathcal{A}'$, while the runs that were non-secret in $\mathcal{A}$ correspond to a secret and a non-secret run of $\mathcal{A}'$.

Thus $DVisit^{priv}_{\leq\Delta}(\mathcal{A}') \supseteq DVisit^{priv}_{>\Delta}(\mathcal{A}') \cup DVisit^{\neg priv}(\mathcal{A}')$ with equality iff *$DVisit^{priv}_{\leq\Delta}(\mathcal{A}) \subseteq DVisit^{priv}_{>\Delta}(\mathcal{A}) \cup DVisit^{\neg priv}(\mathcal{A})$. Therefore the weak ($\leq \Delta$)-ET-opacity of $\mathcal{A}$ can be deduced from the full ($\leq \Delta$)-ET-opacity of $\mathcal{A}'$.*

■

We now temporarily restrict Δ to the integer set $\mathbb{N}^\infty$. (Theorem 8.4 will lift the coming results to $\mathbb{R}^\infty_{\geq 0}$.)

Theorem 8.1 (Decidability of full (resp. weak) exp-ET-opacity decision problem)**.** *The full (resp. weak) exp-ET-opacity decision problem is decidable in* NEXPTIME*.*

Proof. Given a TA $\mathcal{A}$, we first build two TAs from $\mathcal{A}$, named $\mathcal{A}_s$ and $\mathcal{A}_n$ and representing respectively the secret and non-secret behavior of the original TA, while each constant is multiplied by 2. The consequence of this multiplication is that the final location can be reached in time strictly between t and $t+1$ (with $t \in \mathbb{N}$) by a public (resp. secret) run in $\mathcal{A}$ *iff* the target can be reached in time $2t+1$ in the TA $\mathcal{A}_n$ (resp. $\mathcal{A}_s$). Note that the correctness of this statement is a direct consequence of [BDR08, Lemma 5.5].

We then build the region automata $\mathcal{RA}_n$ and $\mathcal{RA}_s$ (of $\mathcal{A}_n$ and $\mathcal{A}_s$ respectively).

$\mathcal{RA}_n$ is a non-deterministic unary (the alphabet is restricted to a single letter) automaton with ε transitions the language of which is $\left\{\mathsf{tick}^k \mid \text{there is a run of duration } k \text{ in } \mathcal{A}_n\right\}$, and similarly for $\mathcal{RA}_s$ (recall that each constant was multiplied by 2 in $\mathcal{A}_n$ and $\mathcal{A}_s$).

We are interested in testing equality (resp. inclusion) of those languages for deciding the full (resp. weak) exp-ET-opacity decision problem.

[SM73, Theorem 6.1] establishes that language equality of unary automata is NP-complete and the same proof implies that inclusion is in NP. As the region automata are exponential, we get the result. □

Remark 11. *In Chapter 7, we established that the full* ($\leq +\infty$)*-ET-opacity decision problem is in* 5EXPTIME *(Proposition 7.3). Theorem 8.1 thus extends our former results in three ways: by including the parameter* Δ*, by reducing the complexity and by considering as well the* weak *notion of ET-opacity.*

■

Theorem 8.2 (Solvability of weak exp-ET-opacity Δ-computation problem)**.** *The weak exp-ET-opacity* Δ*-computation problem is solvable.*

Proof. First, we test whether $\mathcal{A}$ is weakly ($\leq +\infty$)-ET-opaque thanks to Theorem 8.1.

- If $\mathcal{A}$ is weakly ($\leq +\infty$)-ET-opaque then by definition (and monotonicity) of weak exp-ET-opacity, $\mathcal{A}$ is weakly ($\leq \Delta$)-ET-opaque for all $\Delta \in \mathbb{N}^{\infty}$.

- Otherwise, there exists a duration $d \in \mathbb{R}_{\geq 0}$ such that $t \in DVisit^{priv}_{\leq +\infty}(\mathcal{A}) = DVisit^{priv}(\mathcal{A})$ and $d \notin DVisit^{priv}_{>+\infty}(\mathcal{A}) \cup DVisit^{\neg priv}(\mathcal{A}) = DVisit^{\neg priv}(\mathcal{A})$. d can be computed as a smallest word contradicting the inclusion of the language of the two exponential automata described in Theorem 8.1. Hence, d is at most doubly exponential.

  For all $\Delta > t$, we thus have that $DVisit^{priv}_{\leq \Delta}(\mathcal{A}) \not\subseteq DVisit^{priv}_{>\Delta}(\mathcal{A}) \cup DVisit^{\neg priv}(\mathcal{A})$ and thus that $\mathcal{A}$ is not weakly $(\leq \Delta)$-ET-opaque.

  In order to synthesize the bounds $\Delta \in \mathbb{N}$ such that $\mathcal{A}$ is weakly $(\leq \Delta)$-ET-opaque, we therefore only have to test the finitely many integers below t using Theorem 8.1.

□

Corollary 8.1 (Decidability of weak exp-ET-opacity Δ-emptiness problem)**.** *The weak exp-ET-opacity Δ-emptiness problem is decidable.*

Proof. According to Theorem 8.2, the weak exp-ET-opacity Δ-computation problem is solvable. Therefore, to ask for the emptiness, one can compute the set of bounds Δ ensuring the weak $(\leq \Delta)$-ET-opacity and check its emptiness. □

In contrast to the weak exp-ET-opacity Δ-computation problem, we only show below that the full exp-ET-opacity Δ-emptiness problem is decidable; the computation problem remains open.

Theorem 8.3 (Decidability of the full exp-ET-opacity Δ-emptiness problem)**.** *The full exp-ET-opacity Δ-emptiness problem is decidable.*

Proof. Given a TA $\mathcal{A}$, using Theorem 8.2, we first compute the set of bounds Δ such that $\mathcal{A}$ is weakly $(\leq \Delta)$-ET-opaque. As full $(\leq \Delta)$-ET-opacity requires weak $(\leq \Delta)$-ET-opacity, if the computed set is finite, then we only need to check the bounds of this set for full $(\leq \Delta)$-ET-opacity and thus synthesize all the bounds achieving full $(\leq \Delta)$-ET-opacity—which immediately allows us to decide the emptiness.

However, if this set is infinite, by the proof of Theorem 8.2, $\mathcal{A}$ is weakly $(\leq \Delta)$-ET-opaque for any bound $\Delta \in \mathbb{N}^{\infty}$ (and therefore the secret durations are included in the non-secret ones). To achieve full $(\leq \Delta)$-ET-opacity, we only need to detect when the non-secret durations are included in the secret ones. As the set of secret (resp. non-secret) durations increases (resp. decreases) when Δ increases, there is a valuation of Δ achieving full $(\leq \Delta)$-ET-opacity of $\mathcal{A}$ *iff* $\mathcal{A}$ is fully $(\leq +\infty)$-ET-opaque. The latter can be decided with Theorem 8.1. □

We now move the previous results to $\mathbb{R}_{\geq 0}^{\infty}$.

Theorem 8.4. *All aforementioned results with* $\Delta \in \mathbb{N}^{\infty}$ *also hold for* $\Delta \in \mathbb{R}_{\geq 0}^{\infty}$.

Proof. Given a TA $\mathcal{A}$ and $\Delta \in \mathbb{R}_{\geq 0}^{\infty} \setminus \mathbb{N}^{\infty}$, we will show that $\mathcal{A}$ is fully (resp. weakly) $(\leq \Delta)$-ET-opaque *iff* $\mathcal{A}$ is fully (resp. weakly) $(\leq \lfloor \Delta \rfloor + \frac{1}{2})$-ET-opaque.

Constructing the TA $\mathcal{A}^{(2)}$ where every constant is multiplied by 2, we will thus have that $\mathcal{A}$ is fully (resp. weakly) $(\leq \Delta)$-ET-opaque *iff* $\mathcal{A}^{(2)}$ is fully (resp. weakly) $(\leq \Delta^{(2)})$-ET-opaque where $\Delta^{(2)} = 2\Delta$ if $\Delta \in \mathbb{N}$ and $\Delta^{(2)} = 2\lfloor \Delta \rfloor + 1$ otherwise. The previous results of this section applying on $\mathcal{A}^{(2)}$, they can be transposed to $\mathcal{A}$.

We now move to the proof that $\mathcal{A}$ is fully (resp. weakly) $(\leq \Delta)$-ET-opaque *iff* $\mathcal{A}$ is fully (resp. weakly) $(\leq \lfloor \Delta \rfloor + \frac{1}{2})$-ET-opaque. Let $\Delta \in \mathbb{R}_{\geq 0} \setminus \mathbb{N}$ such that $\mathcal{A}$ is fully (resp. weakly) $(\leq \Delta)$-ET-opaque and let $\Delta^{(2)} = \lfloor \Delta \rfloor + \frac{1}{2}$.

Given a run $\rho \in \mathit{Visit}^{\mathit{priv}}_{\leq \Delta}(\mathcal{A})$, let $\mathit{lt}_{\mathit{priv}}(\rho)$ be the time at which ρ enters for the last time the private location. We denote by $V_{\mathit{priv}}(\rho)$ the singleton $\{\mathit{lt}_{\mathit{priv}}(\rho)\}$ if $\mathit{lt}_{\mathit{priv}}(\rho) \in \mathbb{N}$ and the open interval $(\lfloor \mathit{lt}_{\mathit{priv}}(\rho) \rfloor, \lfloor \mathit{lt}_{\mathit{priv}}(\rho) \rfloor + 1)$ otherwise. By definition of the region automaton, one can build runs going through the same path as ρ in $\mathcal{RA}_{\mathcal{A}}$ but reaching the private location at any point within $V_{\mathit{priv}}(\rho)$.

Similarly, given $\mathit{lt}_f(\rho) = \mathit{dur}(\rho)$ the duration of ρ until the final location, we denote $V_f(\rho)$ the singleton $\{\mathit{lt}_f(\rho)\}$ if $\mathit{lt}_f(\rho) \in \mathbb{N}$ and the open interval $(\lfloor \mathit{lt}_f(\rho) \rfloor, \lfloor \mathit{lt}_f(\rho) \rfloor + 1)$ otherwise. Let RRun_{ρ} be the set of runs that follow the same path as ρ in $\mathcal{RA}_{\mathcal{A}}$.

The set of durations of runs of RRun_{ρ} which belong to $\mathit{Visit}^{\mathit{priv}}_{\leq \Delta}(\mathcal{A})$ is $V_f(\rho) \cap [0, \max_{\rho' \in \mathit{RRun}_{\rho}, \mathit{dur}(\rho') = \mathit{dur}(\rho)}(V_{\mathit{priv}}(\rho')) + \Delta]$, which is either $V_f(\rho)$ or the left-open interval $(\lfloor \mathit{lt}_f(\rho) \rfloor, \lfloor \mathit{lt}_f(\rho) \rfloor + \mathrm{fr}(\Delta)]$. We denote by $\mathit{DPriv}_{\Delta}(\rho)$ this set of durations.

Similarly, given a run $\rho \in \mathit{Visit}^{\mathit{priv}}_{> \Delta}(\mathcal{A})$ reaching the final location at time $\mathit{lt}_f(\rho)$, we can again rely on the region automaton to build a set of durations $\mathit{DPub}_{\Delta}(\rho)$ describing the durations of runs that follow the same path as ρ in $\mathcal{RA}_{\mathcal{A}}$ and that reach the final location more than Δ after entering the private location. This set is of the form $\{\mathit{lt}_f(\rho)\}$ if $\mathit{lt}_f(\rho) \in \mathbb{N}$, or $(\lfloor \mathit{lt}_f(\rho) \rfloor + \mathrm{fr}(\Delta), \lfloor \mathit{lt}_f(\rho) \rfloor + 1)$ or $(\lfloor \mathit{lt}_f(\rho) \rfloor, \lfloor \mathit{lt}_f(\rho) \rfloor + 1)$.

- Assume first that $\mathcal{A}$ is fully ($\leq \Delta$)-ET-opaque w.r.t. a bound Δ. As the set of durations reaching the final location is a union of intervals with integer bounds [BDR08, Proposition 5.3] and as $\mathcal{A}$ is fully ($\leq \Delta$)-ET-opaque, the set $\mathit{DVisit}^{\mathit{priv}}_{\leq\Delta}(\mathcal{A})$ and the set $\mathit{DVisit}^{\mathit{priv}}_{>\Delta}(\mathcal{A}) \cup \mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A})$ describes the same union of intervals with integer bounds.

  Let t be a duration within those sets. Then we will show that $t \in \mathit{DVisit}^{\mathit{priv}}_{\leq\Delta^{(2)}}(\mathcal{A})$ and $t \in \mathit{DVisit}^{\mathit{priv}}_{>\Delta^{(2)}}(\mathcal{A}) \cup \mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A})$. Note that if $t \in \mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A})$ the latter statement is directly obtained, we will thus ignore this case in the following.

  By definition of $\mathit{DVisit}^{\mathit{priv}}_{>\Delta}(\mathcal{A})$ and $\mathit{DVisit}^{\mathit{priv}}_{\leq\Delta}(\mathcal{A})$, there exists a run ρ_{priv} and a run ρ_{pub} such that $t \in \mathit{DPriv}_{\Delta}(\rho_{\mathit{priv}})$ and $t \in \mathit{DPub}_{\Delta}(\rho_{\mathit{pub}})$. Moreover, we can assume that those runs satisfy that $\mathit{DPriv}_{\Delta}(\rho_{\mathit{priv}})$ and $\mathit{DPub}_{\Delta}(\rho_{\mathit{pub}})$ do not depend on the bound Δ (i. e., they are equal to $V_f(\rho_{\mathit{priv}})$ and $V_f(\rho_{\mathit{pub}})$ respectively). Indeed, if such runs did not exist, the set $\mathit{DVisit}^{\mathit{priv}}_{\leq\Delta}(\mathcal{A})$ or the set $\mathit{DVisit}^{\mathit{priv}}_{>\Delta}(\mathcal{A}) \cup \mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A})$ would have $\lfloor t \rfloor + \mathrm{fr}(\Delta)$ as one of its bounds. As a consequence, $\mathit{DVisit}^{\mathit{priv}}_{>\Delta}(\mathcal{A}) \cup \mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A}) = \mathit{DVisit}^{\mathit{priv}}_{>\Delta^{(2)}}(\mathcal{A}) \cup \mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A})$ and $\mathit{DVisit}^{\mathit{priv}}_{\leq\Delta}(\mathcal{A}) = \mathit{DVisit}^{\mathit{priv}}_{\leq\Delta^{(2)}}(\mathcal{A})$. Thus $\mathcal{A}$ is fully ($\leq \Delta^{(2)}$)-ET-opaque.

- Assume now that $\mathcal{A}$ is weakly ($\leq \Delta$)-ET-opaque. We consider first the case where $\Delta \geq \Delta^{(2)}$. There we have by definition $\mathit{Visit}^{\mathit{priv}}_{\leq\Delta^{(2)}}(\mathcal{A}) \subseteq \mathit{Visit}^{\mathit{priv}}_{\leq\Delta}(\mathcal{A})$ and $\mathit{Visit}^{\mathit{priv}}_{>\Delta}(\mathcal{A}) \subseteq \mathit{Visit}^{\mathit{priv}}_{>\Delta^{(2)}}(\mathcal{A})$, and therefore $\mathcal{A}$ is weakly ($\leq \Delta^{(2)}$)-ET-opaque.

  Now assume that $\Delta < \Delta^{(2)}$. The same reasoning as for the full version mostly applies. As the set of durations reaching the final location is a union of intervals with integer bounds [BDR08, Proposition 5.3] and as $\mathcal{A}$ is weakly ($\leq \Delta$)-ET-opaque, the set $\mathit{DVisit}^{\mathit{priv}}_{>\Delta}(\mathcal{A}) \cup \mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A})$ describe the same union of intervals with integer bounds. By the same reasoning as before, $\mathit{DVisit}^{\mathit{priv}}_{>\Delta}(\mathcal{A}) \cup \mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A}) = \mathit{DVisit}^{\mathit{priv}}_{>\Delta^{(2)}}(\mathcal{A}) \cup \mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A})$. Moreover, given $t \in \mathit{DVisit}^{\mathit{priv}}_{\leq\Delta^{(2)}}(\mathcal{A})$, there exists ρ_{priv} such that $t \in \mathit{DPriv}_{\Delta^{(2)}}(\rho_{\mathit{priv}})$. Note that either $\mathit{DPriv}_{\Delta^{(2)}}(\rho_{\mathit{priv}}) = \mathit{DPriv}_{\Delta}(\rho_{\mathit{priv}})$ and is thus included in $\mathit{DVisit}^{\mathit{priv}}_{>\Delta}(\mathcal{A}) \cup \mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A})$ or $\mathit{DPriv}_{\Delta^{(2)}}(\rho_{\mathit{priv}}) = (\lfloor \mathit{lt}_f(\rho_{\mathit{priv}}) \rfloor, \lfloor \mathit{lt}_f(\rho_{\mathit{priv}}) \rfloor + \mathrm{fr}(\Delta^{(2)})]$ and $\mathit{DPriv}_{\Delta}(\rho_{\mathit{priv}}) = (\lfloor \mathit{lt}_f(\rho_{\mathit{priv}}) \rfloor, \lfloor \mathit{lt}_f(\rho_{\mathit{priv}}) \rfloor + \mathrm{fr}(\Delta)]$. As the latter is included in $\mathit{DVisit}^{\mathit{priv}}_{>\Delta}(\mathcal{A}) \cup \mathit{DVisit}^{\neg\mathit{priv}}(\mathcal{A})$ which only has integer bounds, then the former is included in it as well.

Remark that, in the above, Δ and $\Delta^{(2)}$ can be freely swapped and thus $\mathcal{A}$ is fully (resp. weakly) ($\leq \Delta^{(2)}$)-ET-opaque *iff* $\mathcal{A}$ is fully (resp. weakly) ($\leq \Delta$)-ET-opaque. □

8.3 Expiring execution-time opacity in parametric timed automata

We are now interested in the synthesis (and the emptiness) of the valuations set ensuring that a system is fully (resp. weakly) exp-ET-opaque. We define the following problems, where we ask for parameter valuations v and for valuations of Δ s.t. $v(\mathcal{P})$ is fully (resp. weakly) ($\leq \Delta$)-ET-opaque.

The full (resp. weak) exp-ET-opacity (Δ+p)-emptiness problem:
Input: A PTA $\mathcal{P}$
Problem: Decide the emptiness of the set of parameter valuations v and valuations of Δ such that $v(\mathcal{P})$ is fully (resp. weakly) ($\leq \Delta$)-ET-opaque.

The full (resp. weak) exp-ET-opacity (Δ+p)-synthesis problem:
Input: A PTA $\mathcal{P}$
Problem: Synthesize the set of parameter valuations v and valuations of Δ such that $v(\mathcal{P})$ is fully (resp. weakly) ($\leq \Delta$)-ET-opaque.

Remark 12. *A "full exp-ET-opacity decision problem" over PTAs is not formally defined; it aims to decide whether, given a parameter valuation v and a bound Δ, a valuated PTA is fully ($\leq \Delta$)-ET-opaque: it can directly reduce to the problem over a TA (which is decidable, Theorem 8.1).*

■

Example 8.3. *Consider again the PTA $\mathcal{P}$ in Figure 3.2.*
For this PTA, the answer to the weak exp-ET-opacity (Δ+p)-emptiness problem is false, as there exists such a valuation (e. g., the valuation given for Example 8.2).
Moreover, we can show that, for all Δ and v:

- $DVisit^{\neg priv}(v(\mathcal{P})) = [0,3]$
- *if* $v(p_1) > 3$ *or* $v(p_1) > v(p_2)$, *it is not possible to reach* ℓ_f *with a run visiting* ℓ_{priv} *and therefore* $DVisit^{priv}_{>\Delta}(v(\mathcal{P})) = DVisit^{priv}_{\leq\Delta}(v(\mathcal{P})) = \emptyset$
- *if* $v(p_1) \leq 3$ *and* $v(p_1) \leq v(p_2)$
  - $DVisit^{priv}_{>\Delta}(v(\mathcal{P})) = (v(p_1) + \Delta, v(p_2)]$

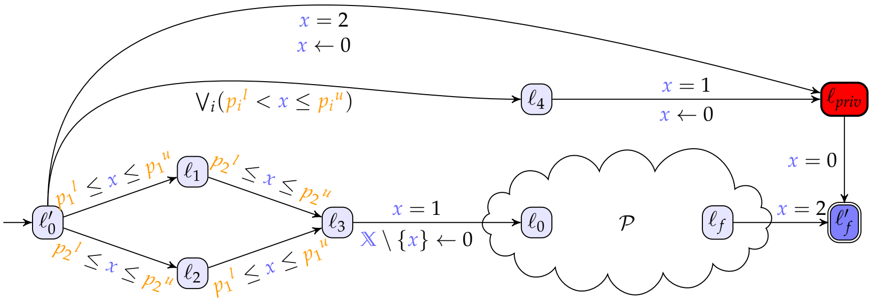

$x = 2$
$x \leftarrow 0$
$\bigvee_i ({p_i}^l < x \leq {p_i}^u)$
ℓ_4
$x = 1$
$x \leftarrow 0$
ℓ_{priv}
$x = 0$
${p_1}^l \leq x \leq {p_1}^u$
ℓ_1
${p_2}^l \leq x \leq {p_2}^u$
ℓ'_0
ℓ_3
$x = 1$
$\mathbb{X} \setminus \{x\} \leftarrow 0$
ℓ_0
$\mathcal{P}$
ℓ_f
$x = 2$
ℓ'_f
${p_2}^l \leq x \leq {p_2}^u$
ℓ_2
${p_1}^l \leq x \leq {p_1}^u$

Figure 8.3: Construction for the undecidability of full (resp. weak) exp-ET-opacity (Δ+p)-emptiness problem for L/U-PTAs (used in Theorem 8.5)

- *$DVisit^{priv}_{\leq\Delta}(v(\mathcal{P})) = [v(p_1), \min(\Delta + 3, v(p_2))]$*

Recall that the full exp-ET-opacity (Δ+p)-synthesis problem aims at synthesizing the valuations such that $DVisit^{priv}_{\leq\Delta}(v(\mathcal{P})) = DVisit^{priv}_{>\Delta}(v(\mathcal{P})) \cup DVisit^{\neg priv}(v(\mathcal{P}))$. The answer to this problem is therefore the set of valuations of timing parameters and of Δ s.t. $v(p_1) = 0 \land ((\Delta \leq 3 \land 3 \leq v(p_2) \leq \Delta + 3) \lor (v(p_2) < \Delta \land v(p_2) = 3))$.

8.3.1 *The subclass of lower/upper parametric timed automata*

Here, we show that both the full exp-ET-opacity (Δ+p)-emptiness problem and the weak exp-ET-opacity (Δ+p)-emptiness problem are undecidable for L/U-PTAs. This is both surprising (seeing from the existing decidability results for L/U-PTAs) and unsurprising, considering the undecidability of the full ET-opacity p-emptiness problem for this subclass (Theorem 7.4).

Theorem 8.5 (Undecidability of full (resp. weak) exp-ET-opacity (Δ+p)-emptiness problem)**.** *The full (resp. weak) exp-ET-opacity (Δ+p)-emptiness problem is undecidable for L/U-PTAs with at least 4 clocks and 4 parameters.*

Proof. We reduce from the problem of reachability-emptiness in constant time, which is undecidable for general PTAs with at least 4 clocks and 2 parameters (Lemma 7.1). That is, we showed that, given a constant time bound T, the emptiness over the parameter valuations set for which a location is reachable in exactly T time units, is undecidable.

Assume a PTA $\mathcal{P}$ with 2 parameters, say p_1 and p_2, and a target location ℓ_f. Fix $T = 1$.

The idea of our proof is that, as in [JLR15] and Proposition 7.4, we "split" each of the two parameters used in $\mathcal{P}$ 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 ${p_i}^l \leq x \leq {p_i}^u$. Therefore, the PTA $\mathcal{P}$ is exactly equivalent to our construction with duplicated parameters, provided ${p_1}^l = {p_1}^u$ 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 Lemma 7.1.

Consider the extension $\mathcal{P}'$ of $\mathcal{P}$ given in Figure 8.3, containing notably new locations ℓ'_0, ℓ_{priv}, ℓ'_f, ℓ_i for $i = 1, \cdots, 4$, and a number of guards as seen on the figure; we assume that x is an extra clock not used in $\mathcal{P}$. The guard on the 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.

Due to the fact that ℓ_{priv} must be exited in 0-time to reach ℓ'_f, note that, for any Δ, the system is fully (resp. weakly) ($\leq \Delta$)-ET-opaque *iff* it is fully (resp. weakly) (≤ 0)-ET-opaque.

Let us first make the following observations, for any parameter valuation v':

1. one can only take the upper most transition directly from ℓ'_0 to ℓ_{priv} at time 2, i. e., ℓ'_f is always reachable in time 2 via a run visiting location ℓ_{priv}: $2 \in \mathit{DVisit}^{priv}(v'(\mathcal{P}'))$;
2. the original PTA $\mathcal{P}$ 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 ℓ'_f is reachable by a public run (not visiting ℓ_{priv}), then its duration is necessarily exactly 2 (going through $\mathcal{P}$);
4. we have $\mathit{DVisit}^{priv}_{>0}(v'(\mathcal{P}')) = \emptyset$ as any run reaching ℓ'_f and visiting ℓ_{priv} can only do it immediately;
5. from Lemma 7.1, 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 due to the guards from ℓ'_0 to ℓ_0, there is no way to reach ℓ'_f with a public run; since ℓ'_f can still be reached for some execution times (notably $x = 2$ through the

upper transition from ℓ'_0 to ℓ_{priv}), then $\mathcal{P}'$ cannot be fully (resp. weakly) (≤ 0)-ET-opaque.

2. If ${p_1}^l < {p_1}^u$ or ${p_2}^l < {p_2}^u$, then one of the transitions from ℓ'_0 to ℓ_4 can be taken, and $DVisit^{priv}_{\leq 0}(v'(\mathcal{P}')) = \{1, 2\}$. Moreover, ℓ'_f might only be reached by a public run of duration 2 through $\mathcal{P}$. Therefore, $DVisit^{\neg priv}(v'(\mathcal{P}')) \subseteq [2, 2]$. Therefore $\mathcal{P}'$ cannot be fully (resp. weakly) (≤ 0)-ET-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 $\mathcal{P}$ (with duplicate parameters) is exactly the one of the original $\mathcal{P}$. Also, note that the transition from ℓ'_0 to ℓ'_f via ℓ_4 cannot be taken. In contrast, the upper transition from ℓ'_0 to ℓ_{priv} can still be taken.

   Now, assume there exists a parameter valuation for which there exists a run of $\mathcal{P}$ of duration 1 reaching ℓ_f. And, as a consequence, ℓ'_f 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 $\mathcal{P}$, which is public. From the above reasoning, all runs reaching ℓ'_f have duration 2; in addition, we exhibited a public and a secret run; therefore the modified automaton $\mathcal{P}'$ is fully (resp. weakly) (≤ 0)-ET-opaque for such a parameter valuation.

   Conversely, assume there exists no parameter valuation for which there exists a run of $\mathcal{P}$ of duration 1 reaching ℓ_f. In that case, $\mathcal{P}'$ is not fully (resp. weakly) (≤ 0)-ET-opaque for any parameter valuation: $DVisit^{priv}_{\leq 0}(v'(\mathcal{P}')) = [2, 2]$ and $2 \notin DVisit^{priv}_{>0}(v'(\mathcal{P}')) \cup DVisit^{\neg priv}(v'(\mathcal{P}')) = \emptyset$).

As a consequence, there exists a parameter valuation v' for which $v'(\mathcal{P}')$ is fully (resp. weakly) ($\leq \Delta$)-ET-opaque *iff* there exists a parameter valuation v for which there exists a run in $v(\mathcal{P})$ of duration 1 reaching ℓ_f—which is undecidable from Lemma 7.1.

The undecidability of the reachability-emptiness in constant time for PTAs holds from 4 clocks and 2 parameters. Here, we duplicate the parameters (which gives 4 parameters), and we add a fresh clock x, never reset (except from ℓ_{priv} to ℓ'_f); however, the construction of Lemma 7.1 also uses a special clock never reset. Since ours is only reset "after" the original $\mathcal{P}$, we can reuse the same clock. Therefore, our result holds from 4 clocks and 4 parameters. □

As the emptiness problems are undecidable, the synthesis problems are immediately intractable as well.

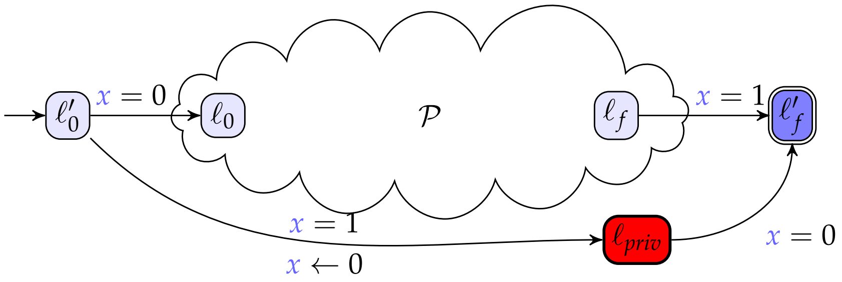


Figure 8.4: Construction for the undecidability of full (resp. weak) exp-ET-opacity (Δ+p)-emptiness problem for PTAs (used in Theorem 8.6)

Corollary 8.2. *The full (resp. weak) exp-ET-opacity (Δ+p)-synthesis problem is unsolvable for L/U-PTAs with at least* 4 *clocks and* 4 *parameters.*

8.3.2 *The full class of parametric timed automata*

The undecidability of the emptiness problems for L/U-PTAs proved above (Theorem 8.5) immediately implies undecidability for the larger class of PTAs. However, we provide below an original proof, with a smaller number of parameters.

Theorem 8.6 (Undecidability of full (resp. weak) exp-ET-opacity (Δ+p)-emptiness problem)**.** *The full (resp. weak) exp-ET-opacity (Δ+p)-emptiness problem is undecidable for general PTAs for at least* 4 *clocks and* 2 *parameters.*

Proof. We reduce again from the problem of reachability-emptiness in constant time, which is undecidable for general PTAs with at least 4 clocks and 2 parameters (Lemma 7.1).

Fix $T = 1$. Consider an arbitrary PTA $\mathcal{P}$, with initial location ℓ_0 and a given location ℓ_f. We add to $\mathcal{P}$ a new clock x (unused and therefore never reset in $\mathcal{P}$), and we add the following locations and transitions in order to obtain a PTA $\mathcal{P}'$, as in Figure 8.4: (*i*) a new initial location ℓ'_0, with an urgent outgoing transition to ℓ_0, and a transition to a new location ℓ_{priv} enabled after 1 time unit; (*ii*) a new final location ℓ'_f with incoming transitions from ℓ_{priv} (in 0-time) and from ℓ_f (after 1 time unit since the system start).

First, due to the guard "$x = 0$" from ℓ_{priv} to ℓ'_f, note that, for any Δ, the system is fully (resp. weakly) ($\leq$ Δ)-ET-opaque *iff* it is fully (resp. weakly) ($\leq$ 0)-ET-opaque. Also note that, for any valuation, $DVisit^{priv}_{\leq 0}(v(\mathcal{P}')) = [1, 1]$. For the same reason, note that $\mathit{DVisit}^{\mathit{priv}}_{>0}(v(\mathcal{P}')) = \emptyset$. Second, note that, due to the guard "$x = 1$" on the edge from ℓ_f and ℓ'_f (with x never reset on this path), $\mathit{DVisit}^{\neg\mathit{priv}}(v(\mathcal{P}'))$ can at most contain $[1, 1]$, i. e., $\mathit{DVisit}^{\neg\mathit{priv}}(v(\mathcal{P}')) \subseteq [1, 1]$.

Now, let us show that there exists a valuation v such that $v(\mathcal{P}')$ is fully (resp. weakly) (≤ 0)-ET-opaque *iff* there exists v such that ℓ_f is reachable in $v(\mathcal{P})$ in 1 time unit.

- $\Rightarrow$ Assume there exists a valuation v such that $v(\mathcal{P}')$ is fully (resp. weakly) (≤ 0)-ET-opaque.

  Recall that, from the construction of $\mathcal{P}'$, $\mathit{DVisit}^{\mathit{priv}}_{\leq 0}(v(\mathcal{P}')) = [1, 1]$. Therefore, from the definition of full (resp. weak) (≤ 0)-ET-opacity, there exist runs only of duration 1 (resp. there exists at least a run of duration 1) reaching ℓ'_f without visiting ℓ_{priv}. Since $\mathit{DVisit}^{\neg\mathit{priv}}(v(\mathcal{P}')) \subseteq [1, 1]$, then ℓ_f is reachable in exactly 1 time unit in $v(\mathcal{P})$.

- $\Leftarrow$ Assume there exists v such that ℓ_f is reachable in $v(\mathcal{P})$ in exactly 1 time unit. Therefore, ℓ'_f can also be reached in exactly 1 time unit: hence, $\mathit{DVisit}^{\neg\mathit{priv}}(v(\mathcal{P}')) = [1, 1]$.

  Now, recall that $\mathit{DVisit}^{\mathit{priv}}_{>0}(v(\mathcal{P}')) = \emptyset$ and $\mathit{DVisit}^{\mathit{priv}}_{\leq 0}(v(\mathcal{P}')) = [1, 1]$. Therefore, $\mathit{DVisit}^{\mathit{priv}}_{\leq 0}(v(\mathcal{P}')) = \mathit{DVisit}^{\mathit{priv}}_{>0}(v(\mathcal{P}')) \cup \mathit{DVisit}^{\neg\mathit{priv}}(v(\mathcal{P}'))$, which from Definition 8.1 means that $v(\mathcal{P}')$ is fully (≤ 0)-ET-opaque. Trivially, we also have that $\mathit{DVisit}^{\mathit{priv}}_{\leq 0}(v(\mathcal{P}')) \subseteq \mathit{DVisit}^{\mathit{priv}}_{>0}(v(\mathcal{P}')) \cup \mathit{DVisit}^{\neg\mathit{priv}}(v(\mathcal{P}'))$ and therefore $v(\mathcal{P}')$ is also weakly (≤ 0)-ET-opaque.

Therefore, there exists v such that $v(\mathcal{P}')$ is fully (resp. weakly) (≤ 0)-ET-opaque *iff* ℓ_f is reachable in $v(\mathcal{P})$ in 1 time unit—which is undecidable (Lemma 7.1). As a conclusion, full (resp. weak) exp-ET-opacity (Δ+p)-emptiness problem is undecidable.

Concerning the number of clocks and parameters, we use the same argument as in the proof of Theorem 8.5: the undecidability of the reachability-emptiness in constant time holds from 4 clocks and 2 parameters, and we add a fresh clock x, but which can be shared with the global clock of Lemma 7.1. Therefore, our construction requires 4 clocks and 2 parameters. □

As the emptiness problems are undecidable, the synthesis problems are immediately intractable as well.

Corollary 8.3. *The full (resp. weak) exp-ET-opacity (Δ+p)-synthesis problem is unsolvable for PTAs for at least 4 clocks and 2 parameters.*

Table 8.1: Summary of the results

		Decision	**Emptiness**	**Computation/Synthesis**
TA	Weak	√(Theorem 8.1)	√(Corollary 8.1)	√(Theorem 8.2)
	Full	√(Theorem 8.1)	√(Theorem 8.3)	?
L/U-PTA	Weak	√(Remark 12)	×(Theorem 8.5)	×(Corollary 8.2)
	Full	√(Remark 12)	×(Theorem 8.5)	×(Corollary 8.2)
PTA	Weak	√(Remark 12)	×(Theorem 8.6)	×(Corollary 8.3)
	Full	√(Remark 12)	×(Theorem 8.6)	×(Corollary 8.3)

8.4 CONCLUSION AND PERSPECTIVES

We defined and studied here a new version of ET-opacity where the secret has an expiration date: that is, we are interested in computing the set of expiration dates of the secret for which the attacker is unable to deduce whether the secret was visited *recently* (i. e., before its expiration date) prior to the system completion; the attacker has access only to the model and to the execution time of the system. We considered both the full exp-ET-opacity (the system must be exp-ET-opaque for all execution times) and the weak exp-ET-opacity (the set of execution times visiting the secret before its expiration date is included into the set of execution times reaching the final location without visiting it, or reaching it after the expiration date). Given a known constant expiration date, the decision problems are all decidable for TAs; in addition, we can effectively *compute* the set of expiration dates for which the system is weakly exp-ET-opaque (full exp-ET-opacity remains open). However, parametric versions of these problems, with unknown timing parameters, turned to be all undecidable, including for the L/U-PTA subclass of PTAs, previously known for some decidability results. This shows the hardness of the considered problem.

SUMMARY We summarize our results in Table 8.1. "√" denotes decidability, while "×" denotes undecidability; "?" denotes an open problem.

Perspectives

The main theoretical future work is the open problem in Table 8.1 (full exp-ET-opacity Δ-computation problem): it is unclear whether we can *compute* the exact set of expiration dates Δ for which a system is fully ($\leq \Delta$)-ET-opaque.

The proofs of undecidability in Section 8.3 require a minimal number of clocks and parameters. Smaller numbers might lead to decidability. In addition, the same proofs are based on an undecidability result (reachability emptiness in constant time (Lemma 7.1)) which uses *rational*-valued parameters. The undecidability of the emptiness problems of Section 8.3 over *integer*-valued parameters does not follow immediately, and remains to be shown.

While the non-parametric part can be (manually) encoded into existing problems (like in Section 7.6.2) using a TA transformation in order to reuse our implementation in IMITATOR, the implementation of the parametric problems remains to be done. Since the emptiness problem is undecidable, this implementation can only come in the form of a procedure without a guarantee of termination, or with an approximate result.

In addition to weak and full exp-ET-opacity, problems focusing on the exp-ET-opacity for *at least* one execution time might give a different decidability or complexity; for example, we highly suspect that the complexity of Theorem 8.1 would decrease in this latter situation.

9

UNTIMED CONTROL FOR EXECUTION-TIME OPACITY

Eu sei que o tempo não pára
O tempo é coisa rara
E a gente só repara
Quando ele já passou.

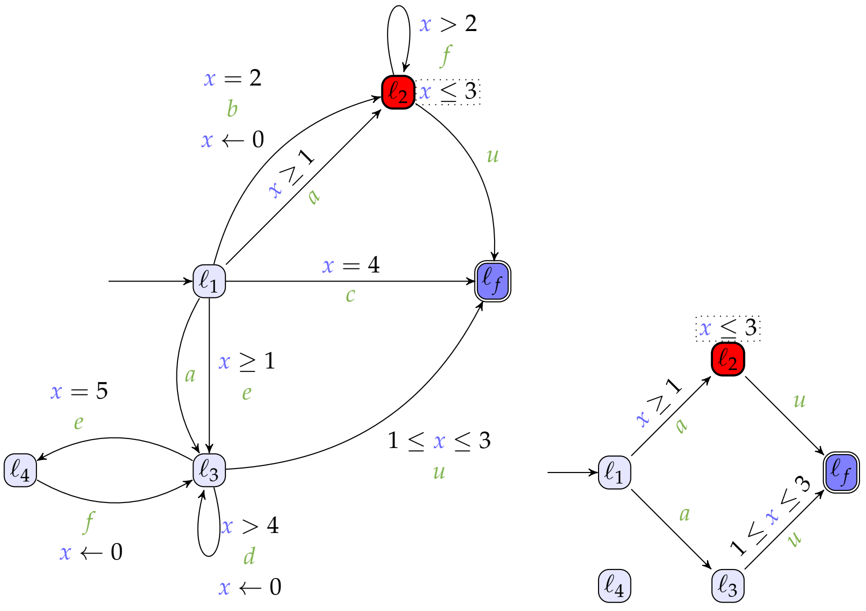
— Mariza, O tempo não para

In this chapter, we focus on the untimed control problem: exhibiting a set of allowed actions, such that the system restricted to those actions is fully ET-opaque. We introduce an algorithm addressing this problem. We then implement our algorithm into a prototype tool strategFTO and evaluate it on a set of case studies.

Motivation

In Chapter 7, we proposed a definition of opacity (ET-opacity) where the attacker only has access (in addition to the model knowledge) to the system *execution time*, i. e., the time from the initial location to a given location. The ET-opacity t-computation problem therefore asks "for which execution times is the attacker unable to deduce whether a private location was visited?" The full ET-opacity decision problem asks whether the system is ET-opaque for all execution times, i. e., the attacker is never able to deduce whether the private location was visited by an execution. We proved in Chapter 7 that this latter problem is decidable (in NEXPTIME; Theorem 8.1), and we proposed a practical algorithm using PTAs, implemented in IMITATOR.

If a system is not fully ET-opaque, there may be ways to tune it to enforce ET-opacity. For instance, one could change internal delays, or add some sleep() or Wait() statements in the program (see e. g., Section 7.6.5). In this chapter, we consider a static (untimed) form of control of the system. This indicates whether there is a way of restricting the behavior of users to ensure full ET-opacity. With that mindset, we assume the set of actions of the TA is partitioned into a set of *controllable* actions (that can be disabled) and a set of *uncontrollable* actions (that cannot be disabled).



(a) A TA example

(b) $Control(\mathcal{A}, \{u, a\})$

Figure 9.1: Running example for control

Contributions of the chapter

We address the following goal: exhibit a controller (i. e., a subset of the system controllable actions to be kept in addition to the uncontrollable actions, while other controllable actions are disabled) guaranteeing the system to be fully ET-opaque. We propose an algorithm exhibiting a set of controllers ensuring full ET-opacity, implemented into a tool strategFTO, calling IMITATOR for computing suitable ET-opaque execution times, and PolyOp [BHZ08] for additional polyhedra operations. This contribution can be seen as a preliminary step towards more complex controllers, such as *timed* controllers making a decision to enforce full ET-opacity at runtime, which would probably require timed games. This is left as future work.

Organization of the chapter

We present our notion of (untimed) control for full ET-opacity in Section 9.1 and its implementation in Section 9.2. We conclude in Section 9.3.

9.1 UNTIMED CONTROL FOR FULL ET-OPACITY

In this section, we introduce an untimed control for controlling ET-opacity. We assume $\Sigma = \Sigma_c \uplus \Sigma_u$ (i. e., $\Sigma = \Sigma_c \cup \Sigma_u$ with

$\Sigma_c \cap \Sigma_u = \emptyset$) where Σ_c (resp. Σ_u) denote controllable (resp. uncontrollable) actions.

Definition 9.1 (Strategy)**.** A (static, untimed) *strategy* of a TA $\mathcal{A}$ is a set of actions $\sigma \subseteq \Sigma$ that contains at least all uncontrollable actions (i. e., $\Sigma_u \subseteq \sigma \subseteq \Sigma$).

A strategy induces a restriction of $\mathcal{A}$ where only the edges labeled by actions of σ are allowed:

Definition 9.2 (Controlled TA)**.** Given $\mathcal{A} = (\Sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, I, E)$ with $\Sigma = \Sigma_u \uplus \Sigma_c$ and a strategy $\sigma \subseteq \Sigma$, the *control* of $\mathcal{A}$ using σ is the TA $\mathcal{A}' = \mathit{Control}(\mathcal{A}, \sigma) = (\sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, I, E')$ where $E' = \{(\ell, g, a, R, \ell') \in E \mid a \in \sigma\}$.

Example 9.1. *Consider the TA in Figure 9.1a, using one clock x. ℓ_1 is the initial location, while ℓ_f is the final location, i. e., a location in which an attacker can measure the execution time from the initial location. ℓ_2 is the private location, i. e., a secret to be preserved: the attacker should not be able to deduce whether it was visited or not. ℓ_2 has an invariant $x \leq 3$ (boxed); other locations invariants are true. Fix $\sigma = \{u, a\}$. Then $\mathit{Control}(\mathcal{A}, \sigma)$ is in Figure 9.1b.*

Strategies represent some modifications of the system that can be implemented to ensure full ET-opacity. We therefore define fully ET-opaque strategies as strategies ensuring full ET-opacity.

Definition 9.3 (Fully ET-opaque strategy)**.** A strategy σ is *fully ET-opaque* if $\mathit{Control}(\mathcal{A}, \sigma)$ is fully ET-opaque.

Note that a strategy (even a maximal one) might achieve full ET-opacity by blocking all runs (both private or public) from reaching the target. If reaching the target means completing a task, this might not be something one would desire. We call a fully ET-opaque strategy allowing to reach the target for at least some execution time an *effective fully ET-opaque strategy*.

We define two slightly different problems: taking a TA $\mathcal{A}$ as input, the (resp. effective) fully ET-opaque strategy-emptiness problem asks whether the set of (resp. effective) fully ET-opaque strategy for $\mathcal{A}$ is empty.

The fully ET-opaque strategy-emptiness problem:
Input: A TA $\mathcal{A}$
Problem: Decide the emptiness of the set of fully ET-opaque strategy for $\mathcal{A}$.

The effective fully ET-opaque strategy-emptiness problem:
Input: A TA $\mathcal{A}$
Problem: Decide the emptiness of the set of effective fully ET-opaque strategy for $\mathcal{A}$.

Note that, due to the presence of uncontrollable actions, the first problem (fully ET-opaque strategy-emptiness problem) is not trivial. (If uncontrollable actions were not part of our definitions, choosing $\sigma = \emptyset$ would always yield an acceptable fully ET-opaque strategy.)

We will also refine those problems by considering a notion of *maximal* (i. e., most permissive) strategy w.r.t. full ET-opacity based on the number of actions belonging to the strategy: given $\mathcal{A}$, a fully ET-opaque strategy σ is maximal if for all strategy σ', if σ' is fully ET-opaque then $|\sigma'| \leq |\sigma|$. We define similarly *minimal* strategies (least permissive, i. e., disabling as many actions as possible) as well as maximal (resp. minimal) effective fully ET-opaque strategy, i. e., the set of largest (resp. smallest) effective fully ET-opaque strategies.

Example 9.2. *Consider again the TA $\mathcal{A}$ in Figure 9.1a. Assume $\Sigma_u = \{u\}$ and $\Sigma_c = \{a, b, c, d, e, f\}$. Fix $\sigma_1 = \{u, b, c\}$. We have $DVisit^{priv}(Control(\mathcal{A}, \sigma_1)) = [2, 5]$ while $DVisit^{\neg priv}(Control(\mathcal{A}, \sigma_1)) = [4, 4]$; therefore, σ_1 is not fully ET-opaque.*
Now fix $\sigma_2 = \{u, a, f\}$. We have $DVisit^{priv}(Control(\mathcal{A}, \sigma_2)) = DVisit^{\neg priv}(Control(\mathcal{A}, \sigma_2)) = [1, 3]$; therefore, σ_2 is fully ET-opaque.
In fact, it can be shown that the set of effective fully ET-opaque strategies for $\mathcal{A}$ is $\{\{u, a\}, \{u, a, e\}, \{u, a, f\}\}$; therefore, $\{u, a\}$ is the only minimal strategy, while $\{u, a, e\}, \{u, a, f\}$ are the two maximal strategies. In addition, $\{u, f\}$ is an example of a fully ET-opaque strategy that is not effective, as ℓ_f is always unreachable, whether ℓ_{priv} is visited or not.

9.1.1 *Complexity*

Proposition 9.1 (Complexity)**.** *Given a TA $\mathcal{A}$, one can compute the set of fully ET-opaque strategy over $\mathcal{A}$ in* NEXPTIME.

Proof. The full ET-opacity decision problem (i. e., checking if a given TA is fully ET-opaque) is decidable for TAs in (at most) NEXPTIME (Proposition 7.3, Theorem 8.1, and Remark 11). Moreover, reachability of the final state can be decided in PSPACE [AD94]. Thus, for any given strategy, one can check in NEXPTIME whether it is a (resp. effective) fully ET-opaque strategy.

Computing the list of (resp. effective) fully ET-opaque strategies can be done naively by testing each possible strategy one by one and keeping the ones that satisfy the property we want. As there is an exponential number of possible strategies and repeating exponentially many times a NEXPTIME algorithm remains in NEXPTIME, this algorithm is in NEXPTIME. □

As a corollary of the above, the (resp. effective) fully ET-opaque strategy-emptiness problem is in NEXPTIME as well. More precisely, the above proof establishes that the complexity class of the (resp. effective) fully ET-opaque strategy-emptiness problem is the maximum between PSPACE and the complexity of the full ET-opacity decision problem. As the latter is PSPACE-hard (being trivially harder than reachability), the two problems lie in the same complexity class. From a theoretical point of view, one thus cannot do better (in terms of complexity) than the naive enumeration approach described here to solve the control problem.

Finding the maximal (resp. minimal) strategies can be done slightly more efficiently by starting from the set with every (resp. no) controllable action and enumerating the potential strategies by decreasing (resp. increasing) order as one could then potentially stop before full enumeration. In the worst case, this will however have the same complexity as the full enumeration.

9.2 IMPLEMENTATION AND EXPERIMENTS

9.2.1 *Implementation in strategFTO*

We implemented our strategy generation in strategFTO, an entirely automated open-source tool written in Java.[1] Our tool iteratively

[1] Source code is available at https://github.com/DylanMarinho/Controlling-TA. Models and experiment results are available at 10.5281/zenodo.7181848.

Algorithm 9.1: synthCtrl($\mathcal{A}$) Exhibit all fully ET-opaque strategies

```
1 $\mathcal{S} \leftarrow \emptyset$
2 foreach $s \subseteq \Sigma_c$ do
3     $\sigma \leftarrow s \cup \Sigma_u$
      /* Compute execution times */
4     $\lambda_1 \leftarrow DVisit^{\neg priv}(Control(\mathcal{A}, \sigma))$
5     $\lambda_2 \leftarrow DVisit^{priv}(Control(\mathcal{A}, \sigma))$
      /* Check for full ET-opacity */
6     if $\lambda_1 = \lambda_2$ then $\mathcal{S} \leftarrow \mathcal{S} \cup \{\sigma\}$ ;
7 return $\mathcal{S}$
```

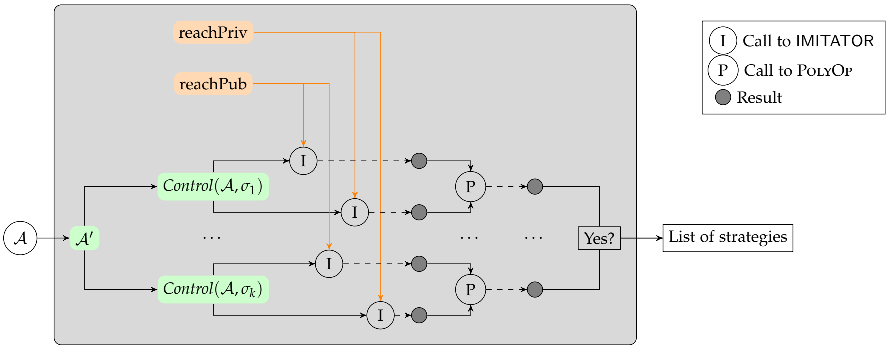


Figure 9.2: Overview of the tool strategFTO

constructs strategies, then checks full ET-opacity by computing the private and public execution times and by checking their equality (Algorithm 9.1).

We give our strategy synthesis algorithm in Algorithm 9.1. The exhibition of these execution times ($DVisit^{\neg priv}(\mathcal{A})$ and $DVisit^{priv}(\mathcal{A})$, Lines 4 and 5) is done in our implementation by an automated model modification (following the procedure described in Sections 7.5 and 7.6.4, but which was not entirely automated in the aforementioned experiments) followed by a synthesis problem using PTAs. The synthesis of the execution times itself is done by a call to IMITATOR 3.3 *"Cheese Caramel au beurre salé"*.

strategFTO then checks whether both sets of execution times are equal; this is done by a call to another external tool—PolyOp 1.2[2], that performs polyhedral operations as a simple interface for PPL.

In Figure 9.2, we give an overview of the tool, with calls to IMITATOR and PolyOp.

ALGORITHMS We implement not only the exhibition of all fully ET-opaque strategies (denoted by synthCtrl($\mathcal{A}$), in Algorithm 9.1), but also the following variants:

[2] https://github.com/etienneandre/PolyOp

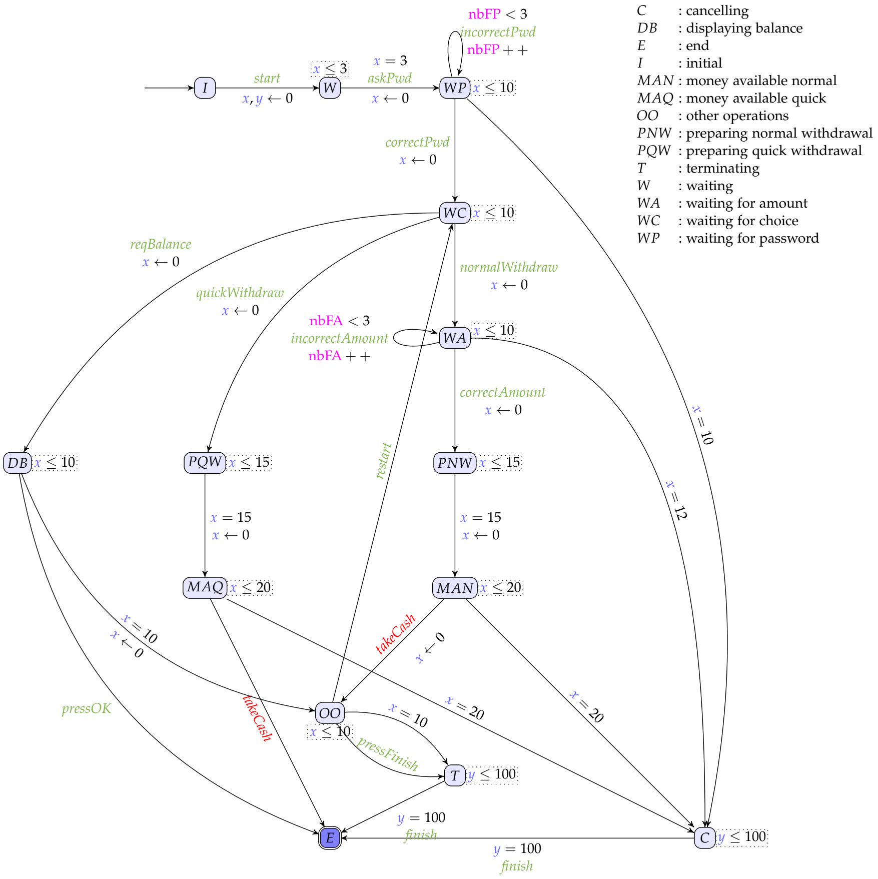


Figure 9.3: ATM benchmark

1. synthMaxCtrl($\mathcal{A}$): synthesize all maximal strategies for $\mathcal{A}$;
2. synthMinCtrl($\mathcal{A}$): synthesize all minimal strategies;
3. witnessMaxCtrl($\mathcal{A}$): witness *one* maximal strategy;
4. witnessMinCtrl($\mathcal{A}$): witness *one* minimal strategy.

We implemented these other algorithms by changing the exploration order of the strategies, and/or by triggering immediate termination upon the first exhibition of a strategy.

INPUT MODEL The input TA model is given in the IMITATOR input syntax and can therefore include all the extensions described in Section 4.1.

9.2.2 *Proof of concept benchmark*

As a proof of concept, we consider the TA model of an ATM (given in Figure 9.3). The idea is that (as per our definition of ET-opacity) the attacker only has access to the execution time, i. e., the time from the beginning of the program to reaching the end state. The secret is whether the ATM user has actually obtained cash (action *takeCash*).[3] The TA uses two clocks: x for "local" actions, and y for a global time measurement. First, the user starts the process (action *start*), then the ATM displays a welcome screen for 3 time units, followed by another screen requesting the password (action *askPwd*). Then, the user can submit a correct (action *correctPwd*) or incorrect (*incorrectPwd*) password; if no password is input within 10 time units, the system moves to a cancelling phase. The same happens if 3 incorrect password have been input. After inputting the correct password, the user has the choice between a fixed-amount quick withdrawal (*quickWithdraw*), a normal withdrawal (*normalWithdraw*) or a balance request (*reqBalance*).

The quick withdrawal triggers a 15-time unit preparation followed by the availability of the money, which the user can take immediately (action *takeCash*), thus terminating the procedure. If the user does not take the money, the system moves to the cancelling phase.

The normal withdrawal asks the user to input the desired amount; similar to the password, after 3 wrong amounts (action *incorrectAmount*), or upon timeout, the system moves to cancelling phase. After the user retrieves cash (action *takeCash*), they are asked whether they would like to perform another operation; if so (action *restart*), the system goes back to the choice location. Otherwise (action *pressFinish*), or unless a 10-time unit timeout is reached, the system moves to the terminating location. The balance request triggers the balance display, from which the user can immediately terminate the process (action *pressOK*), or go back to the choice menu.

The rationale is that, in the regular terminating and cancelling phases, the ATM terminates after constant time (invariant $y \leq 100$), avoiding leaking information. However, some actions may lead to quicker termination (quick withdrawal) or slower termination (multiple choices).

The uncontrollable actions are most of the user actions: *correctAmount*, *incorrectAmount*, *correctPwd*, *incorrectPwd*, *pressFinish*, *takeCash*. The controllable actions are the system actions (*askPwd*, *start*, *finish*) and some of the users actions that can be controlled by disabling the associated choice (*reqBalance*, *pressOK*, *quickWithdraw*, *restart*).

9.2.3 *Experiments*

We first exhibit in Table 9.1 strategies for our benchmark from Figure 9.3 as computed by strategFTO, for all our algorithms. For space

[3] strategFTO allows not only private locations, but also actions. Formally, we could use an observer with three locations $\left\{\ell^o_0, \ell^o_f, \ell^o_{priv}\right\}$. It is synchronized with the model with a private action (only) on the transition from ℓ^o_0 to ℓ^o_{priv}: when the private action occurs, the observer goes to ℓ^o_{priv}. We can therefore check ET-opacity on the synchronized TA with ℓ^o_{priv}.

Table 9.1: Strategy synthesis for Figure 9.3

Actions to disable	synthMinCtrl($\mathcal{A}$)	witnessMinCtrl($\mathcal{A}$)	synthMaxCtrl($\mathcal{A}$)	witnessMaxCtrl($\mathcal{A}$)	synthCtrl($\mathcal{A}$)
restart, *pressOK*			√	√	√
restart, *reqBalance*			√		√
restart, *pressOK*, *quickWithdraw*					√
restart, *pressOK*, *reqBalance*					√
restart, *quickWithdraw*, *reqBalance*					√
restart, *pressOK*, *quickWithdraw*, *reqBalance*	√	√			√

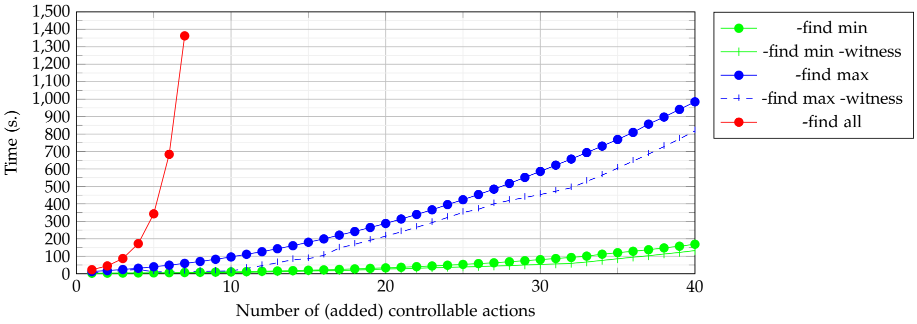


Figure 9.4: Execution times for scalability (in seconds; TO set at 1,800 s)

concern, we tabulate the actions to *disable*; the strategy is therefore Σ minus these actions. Also note that, for witnessMaxCtrl(•) and witnessMinCtrl(•), the *order* in which we compute the subsets of Σ in Algorithm 9.1 has an impact on the result, as the algorithm stops as soon as *one* strategy is found. According to Table 9.1, the maximal strategies (i. e., the most permissive, disabling the least number of actions) are to disable either *restart* and *pressOK*, or *restart* and *reqBalance*. This is natural, as *restart* allows the user to restart a second operation, thus violating the constant-time nature of Figure 9.3, while *pressOK* and *reqBalance*, if enabled together, allow a quick exit, shorter than a cash withdrawal operation—thus giving hint to the attacker that the *takeCash* secret did *not* occur.

SCALABILITY Then, we test the scalability of strategFTO w.r.t. the number of actions. We modify Figure 9.3 by adding an increasingly large numbers of controllable actions; these actions do not play a role in the control (we basically add unguarded self-loops) but they will impact the computation time, as we will need to consider an increasingly (and exponentially) larger number of subsets of actions, from Algorithm 9.1. We add from 1 to 40 such actions, resulting (by adding the actions in Figure 9.3) in a model with a number of controllable actions from 11 to 50. We plot these results in Figure 9.4 and present raw results in Table 9.2. From our results in Figure 9.4, we see that, without surprise, the execution time for synthCtrl(•) is exponential in the number of actions. However, synthMaxCtrl(•) and synthMinCtrl(•) behave much better, by remaining respectively below 15 minutes and 3 minutes, even for up to 50 controllable actions. In addition, it is important to notice that witnessMaxCtrl(•) and witnessMinCtrl(•)

Table 9.2: Execution times for scalability (in seconds; TO set at 1,800 s)

Number of added actions	synthMinCtrl($\mathcal{A}$) -find min	witnessMinCtrl($\mathcal{A}$) -find min -witness	synthMaxCtrl($\mathcal{A}$) -find max	witnessMaxCtrl($\mathcal{A}$) -find max -witness	synthCtrl($\mathcal{A}$) -find all
1	2.89	2.04	12.61	5.92	22.98
2	3.19	2.44	17.81	11.30	44.68
3	3.84	2.74	23.99	17.58	87.07
4	4.43	2.85	31.15	24.92	172.26
5	4.90	3.77	39.69	10.34	342.95
6	6.07	4.09	48.78	11.12	683.77
7	7.02	4.54	59.14	12.35	1,362.48
8	8.34	4.69	70.09	13.46	TO
9	9.32	5.63	82.45	14.52	TO
10	10.51	5.91	95.86	15.65	TO
11	12.04	7.66	111.16	30.49	TO
12	13.99	9.43	126.11	46.00	TO
13	15.54	10.94	143.15	62.50	TO
14	17.58	12.83	160.41	80.32	TO
15	19.88	15.03	180.24	85.64	TO
16	21.94	17.38	199.34	105.15	TO
17	24.39	17.63	221.04	146.98	TO
18	27.64	20.53	241.70	168.79	TO
19	30.49	23.65	264.72	191.16	TO
20	33.43	26.59	287.85	215.33	TO
21	36.58	28.60	313.34	239.52	TO
22	40.46	30.92	339.24	265.90	TO
23	44.31	33.92	366.07	292.51	TO
24	48.52	36.43	395.95	322.16	TO
25	53.21	38.30	423.86	350.92	TO
26	58.02	41.21	453.59	368.86	TO
27	62.36	43.57	484.28	400.39	TO
28	68.56	46.60	517.03	419.43	TO
29	74.39	49.74	551.58	436.35	TO
30	80.64	53.15	586.03	453.37	TO
31	86.89	55.72	621.83	472.63	TO
32	92.91	59.14	656.75	492.10	TO
33	100.67	67.00	693.82	528.71	TO
34	111.66	76.17	730.93	564.68	TO
35	120.37	85.48	768.87	604.42	TO
36	128.35	94.59	809.36	645.01	TO
37	137.28	102.77	856.98	685.03	TO
38	147.68	112.83	897.39	728.48	TO
39	157.42	121.77	940.98	771.68	TO
40	168.74	132.45	984.65	818.25	TO

do not decrease the time very much compared to the full versions synthMaxCtrl(•) and synthMinCtrl(•). This is because, at a given size, the number of strategies to be tested remains relatively small.

9.3 CONCLUSION AND PERSPECTIVES

We introduced in this chapter a control problem where the goal is to exhibit strategies to guarantee the full ET-opacity of a system modeled by a TA where the attacker only has access to the execution time. We implemented our algorithm into a prototype tool strategFTO. Even though it relies on a simple enumeration of the subsets, our tool strategFTO shows good performance for synthesizing maximal or minimal strategies, with very reasonable times, even for several dozens of controllable actions.

Perspectives

We plan to further optimize our implementation by maintaining a set of non-effective strategies, i. e., for which ℓ_f is unreachable: any strategy strictly included into a known non-effective strategy will necessarily be non-effective too, and therefore no ET-opacity analysis is needed for this strategy. An option to efficiently represent this strategies set could be to store it using Binary Decision Diagrams [Bry86].

We also plan to strengthen strategies so that their choice may depend on how long has passed since the start of the execution. As these strategies still need a finite representation to be handled, this requires establishing exactly what strategies need to remember to chose optimally.

The next step is to move towards a *timed* controller, where strategies are not disabled in the entire system, but depending on the current location and timestamp at runtime. Considering other notions around ET-opacity (as ∃-ET-opacity, weak ET-opacity or expiration secrecy notions) and exhibiting (timed/untimed) controllers is also a future work. Runtime enforcement, as in [FM15], is also planned.

Our ultimate goal will be to extend TAs to PTAs, and use automated parameter synthesis techniques (e. g., [JLR15; AAPP21; AMP21a]), with a parametric timed controller [JLR19; Gol21].

GENERAL CONCLUSION

Imagine there's no heaven
It's easy if you try
No hell below us
Above us only sky
Imagine all the people
Living for today

Imagine there's no countries
It isn't hard to do
Nothing to kill or die for
And no religion too
Imagine all the people
Living life in peace

Imagine no possessions
I wonder if you can
No need for greed or hunger
A brotherhood of man
Imagine all the people
Sharing all the world

You may say I'm a dreamer
But I'm not the only one
I hope someday you'll join us
And the world will be as one

— John Lennon, Imagine

10

GENERAL CONCLUSION

> *The bitterest tears shed over graves*
> *are for words left unsaid and deeds left undone.*
>
> — Harriet Beecher Stowe

In this thesis, we presented theoretical and algorithmic contributions to the analysis of safety and security properties in timed systems under uncertainty.

10.1 CONTRIBUTION SUMMARY

Part I: Zone Merging in parametric timed automata

In Part I, we studied efficient verification of PTAs.

First, we introduced a benchmark library about PTAs, composed of 56 benchmarks with 119 different models and 216 properties. This library provides several features, including multiple kinds of properties (about reachability and liveness), classical extensions of PTAs (such as stopwatches) or unsolvable toy benchmarks. We used the framework of the IMITATOR model checker, as a *de facto* standard for PTAs verification, with useful tools to allow the translation of the models into other model checkers (such as UPPAAL) or specifications (such as JANI).

Second, we investigated merging techniques for the efficient verification of reachability properties in PTAs. We introduced different merging heuristics and implemented them in IMITATOR. They are mainly focused on the states considered for merging and on the techniques to update the state-space after a successful merge. We evaluated and compared each combination of options with the previously implemented version. Our method leads to a gain of 62% of the average computation time over the aforementioned benchmark library.

Part II: Execution-time opacity

Execution-time opacity

In Part II, we first introduced the ET-opacity concept: a timed system is ET-opaque if it is impossible for an attacker to decide whether an execution of the system visits a private location only knowing the system execution time. We considered two levels of this definition:

i) ∃-ET-opacity: there exists such an execution time;

ii) full ET-opacity: for each possible execution time, it is possible to visit and not to visit the private location.

We studied many problems around this definition, such as decision problems (asking to decide whether a system is ET-opaque) or the ET-opacity t-computation problem (asking to compute the execution times ensuring ET-opacity). We also introduced parametric problems, asking to synthesize internal constants in the system to make the system ET-opaque. We showed that these problems are decidable in TAs, but become undecidable for PTAs. The subclass of L/U-PTAs was studied and remains at the frontier between decidability and undecidability (in this subclass, e. g., the emptiness of parameter valuations for ∃-ET-opacity is decidable, but their synthesis is intractable).

Expiring execution-time opacity

Secondly, we extended the previous definition, with the introduction of an expiration date of the secret (exp-ET-opacity). With this concept, runs are not simply separated in term of visit of the private location but we also consider the time spent from its last visit; a run for which the visit was performed "too early" (i. e., more than the expiration date before the end of the run) is not considered as private.

We were interested in decision (for a given bound, decide if the system is exp-ET-opacity) and computation problems (compute the expiring bounds). We proved that these problems are decidable, except the full exp-ET-opacity Δ-computation problem which remains open. We introduced parametric extensions of these problems, which are undecidable even for the subclass of L/U-PTAs.

Untimed control

Thirdly, we were interested in untimed control for full ET-opacity, i. e., in finding a subset of actions to disable to ensure ET-opacity. We proposed an algorithm and implemented a tool, strategFTO, synthesizing all the strategies of a given timed system. This tool was tested on a proof-of-concept benchmark and we performed scalability experiments, studying the evolution of its computation time when the number of actions is increased.

10.2 PERSPECTIVES

10.2.1 *Parametric timed model checking*

In this thesis, we have only studied some form of merging, in a reachability context and only merging two-by-two areas. Extending our work to a more general context, for example on liveness properties, would be a considerable improvement.

We could also investigate a generic format that could be interoperable with all parametric timed verification tools, for example by extending the works leading to the JANI specification. This format could then be integrated with other tools, for example scheduling ones, in order to check these problems with existing tools.

10.2.2 *Other kinds of parameters*

In this thesis, we focused on timed systems and timed parameters. However, other types of parametrization exist: probabilistic systems, systems with costs, systems with a parametric number of components ("parameterized verification"). Extending the work we have done, as well as extending our notions, to such models could be an interesting continuation.

First, we may want to add these kinds of parameters in our models, and extend our definition to consider TAs and PTAs with costs [LRS21] or probabilities [EDLR16; ADF20]. These extensions may allow us to define more general problems, such as attacks based on the frequency (i. e., the probability) to visit a private location. Parametrized verification will allow us to verify a network of PTAs, with an arbitrary number of components, as done in [ADFL19].

Second, we may study the equivalent problems in untimed systems, where parameters represent other types of data, and therefore allow us to model other types of attacks (e. g., considering the total cost of a system execution). As the considered formalism varies, our study could lead to different decidability results and other algorithms.

10.2.3 *Execution-time opacity*

Theory

In Part II, we proved a number of (un)decidability results related to ET-opacity problems. However, some of them remain open, including the full exp-ET-opacity Δ-computation problem or the existential extension of exp-ET-opacity (i. e., for *at least* one execution time). In addition, the decidability of several "low-dimensional" problems (i. e., with "small" number of clocks or parameters) remains open, as well as the study of more restricted classes, hoping to exhibit a decidable one.

We have shown that these problems were generally decidable on TAs (i. e., for non-parametric models). However, Franck Cassez [Cas09] has shown that the general notion of timed opacity (allowing the observation of certain actions by the attacker) was undecidable, even for subclasses of TAs with restricted expressiveness. We could therefore question how far we could extend an attacker (so that they would be more powerful than the one we considered), while keeping the decidability of the studied problems. For example, could they have a certain view of the system at given times, i. e., know information about the system at some time steps? We could also work on semi-algorithms considering the definition introduced by Franck Cassez, as we did for our notion.

Moreover, we could apply the definition of ET-opacity to other formalisms, such as time Petri nets or hybrid automata. They could allow to define slightly different problems or to have a more general concept; for example, considering probabilistic extensions of our ET-opacity could be an interesting extension to look at.

Applications

Although we present experiments with each of the contributions, the translation of programs required some non-trivial creativity: while the translation from programs to quantitative extensions of automata is orthogonal to our work, proposing automated translations of TAs dedicated to timing analysis was not studied here but could be investigated. Some works were performed (e. g., [LKP19]) and can provide the basis for a future work.

Furthermore, although an algorithm has been presented to solve the full ET-opacity p-synthesis problem, it lacks the ability to compute, in practice, expiration dates or secure execution times (for all versions of ET-opacity and exp-ET-opacity). Reflection on these topics could further extend our work into practical considerations.

RÉSUMÉ EN FRANÇAIS

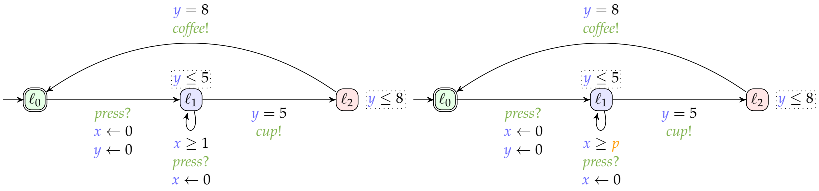
Les systèmes temps-réels sont présents dans de multiples champs d'applications, comme les transports, les télécommunications ou l'industrie. Cependant, des accidents peuvent arriver et il est nécessaire d'avoir confiance en ces systèmes afin de les éviter. Il est donc nécessaire de pouvoir prouver formellement que leur comportement sera conforme à une spécification. Celle-ci peut être de deux natures, s'intéressant (*i*) à la sûreté du système, montrant qu'il aura toujours un comportement attendu, (*ii*) mais aussi la sécurité, montrant qu'il sera résistant à certaines attaques. Pour cela, le formalisme des automates temporisés (*timed automata*, TAs) [AD94] est assez commun. Néanmoins, cette modélisation peut être imparfaite, en raison de la nature du système, de simplifications devant être faites ou d'imprécisions. Nous étudions donc ces systèmes temporisés sous incertitude, c'est à dire à l'aide de paramètres. L'extension naturelle étudiée est alors le formalisme des automates temporisés paramétrés (*parametric timed automata*, PTAs) [AHV93].

> Cette thèse présente plusieurs contributions, à la fois sur des techniques permettant une vérification efficace de ces systèmes ainsi que sur une formalisation de propriétés de sécurité.

A LES AUTOMATES TEMPORISÉS

Plus précisément, un exemple de TA et de PTA est donné en Figure A.1.

Un TA est composé de localités (ℓ_0, ℓ_1, ℓ_2), d'actions (*press?*, *cup!*, *coffee!*) et d'un ensemble d'horloges (x, y). Ces horloges sont des variables réelles qui évoluent linéairement. Elles peuvent ensuite :



(a) Une machine à café modélisée avec un TA

(b) Une paramétrisation de la machine à café

FIGURE A.1 : Exemple de TA et de PTA modélisant une machine à café

Tableau B.1 : Quelques améliorations de la librairie

Version de	Taille			Métriques		Format		Catégories	Propriétés			Analyse
librairie	Bench.	Modèles	Propriétés	Statiques	Sémantiques	.imi	JANI	Insolvables	EF	TPS	Vivacité	Résultats
1.0 [And18]	34	80	122	√	×	2.12	×	×	√	√	×	×
2.0 [AMP21c]	56	119	216	√	√	3.0	√	√	√	√	√	√

- être comparées au sein de gardes (propriétés à vérifier pour activer une transition) ;
- être comparées au sein d'invariants (propriétés à vérifier pour rester dans une localité) ;
- être réinitialisées le long d'une transition.

Un PTA étend naturellement un TA en offrant la possibilité d'utiliser des paramètres (p) au sein de gardes ou d'invariants.

B VÉRIFICATION EFFICACE

Cette première partie s'articule en deux contributions, en proposant :

1) une librairie de *benchmarks* pour le model-checking temporisé paramétré ;
2) une notion permettant la fusion d'états dans le graphe de zones paramétrées (*parametric zone graph*, PZG) [AD94] d'un PTA.

B.1 *Une librairie de benchmarks*

La librairie proposée étend une version précédente [And19a]. Cette extension propose un certain nombre d'améliorations, résumés dans le Tableau B.1. Ainsi, nous proposons désormais des métriques sémantiques (p. ex., temps de calcul et taille de l'espace d'états), une exportation vers le format JANI, ainsi que des propriétés de vivacité (*liveness*).

B.2 *Fusion de zones*

Nous avons également introduit une notion de fusion de zones dans les PZGs.

La fusion de zones est définie comme suit :

Définition (Fusion). Deux états symboliques $\mathbf{s}_1 = (\ell_1, \mathbf{C}_1)$, $\mathbf{s}_2 = (\ell_2, \mathbf{C}_2)$ sont *fusionnables*, désigné par le prédicat $\mathit{is\_mergeable}(\mathbf{s}_1, \mathbf{s}_2)$, si $\ell_1 = \ell_2$ et $\mathbf{C}_1 \cup \mathbf{C}_2$ est convexe.
Dans ce cas, leur *fusion* est définie comme étant $(\ell_1, \mathbf{C}_1 \cup \mathbf{C}_2)$.

Il est alors possible d'inclure cette notion de fusion lors de la construction d'un PZG. Cette méthode ayant été implémentée, nous avons comparé différentes heuristiques et avons montré qu'en pratique cette méthode permet de réduire en moyenne de 62% le temps de calcul.

c Opacité par rapport au temps d'exécution

Nous nous intéressons ensuite à une notion d'opacité sur les PTAs. Cette partie se décompose en trois contributions :

1) la formalisation de la notion d'opacité par rapport au temps d'exécution (*execution-time opacity*, ET-opacity), avec des résultats de décidabilité ainsi que des expérimentations ;

2) une première extension, considérant une opacité par rapport au temps d'exécution avec expiration (*expiring ET-opacity*, exp-ET-opacity), c'est-à-dire que les secrets pourront expirer après un certain délai ;

3) une seconde extension, introdusant la notion de contrôle non temporisé pour l'ET-opacity.

c.1 *La notion de l'ET-opacity*

Définitions

Étant donné un TA $\mathcal{A}$, nous commençons par définir deux ensembles particuliers de durées :

- $DVisit^{priv}(\mathcal{A})$ comme étant l'ensemble des durées des chemins possibles pour lesquels ℓ_{priv} est visitée ;
- $DVisit^{\neg priv}(\mathcal{A})$ comme étant l'ensemble des durées des chemins possibles pour lesquels ℓ_{priv} *n'est pas* visitée.

Dans notre formalisme, un attaquant cherche à déterminer un secret (exprimé en terme de visite d'une localité) en ne connaissant que le temps d'exécution total du système (ainsi que le modèle). Nous définissons formellement cette notion d'ET-opacity ainsi :

Définition (Opacité par rapport au temps d'exécution)**.** Étant donné un TA $\mathcal{A}$ avec un localité privée ℓ_{priv} et un ensemble de temps d'exécutions D, on dit que $\mathcal{A}$ est *opaque par rapport au temps d'exécution pour les temps d'exécutions D* si

$$D \subseteq DVisit^{priv}(\mathcal{A}) \cap DVisit^{\neg priv}(\mathcal{A}).$$

Ainsi, étant donnée une durée $d \in DVisit^{priv}(\mathcal{A}) \cap DVisit^{\neg priv}(\mathcal{A})$, un attaquant ne pourra pas déduire la visite ou non de la localité privée ℓ_{priv} pour un chemin de cette durée, ces deux cas étant possibles.

Nous nous intéressons ensuite à l'existence d'une telle durée d. On dit alors que le TA est ∃-ET-opaque s'il est ET-opaque pour un ensemble non vide de temps d'exécution.

Définition (∃-ET-opacity)**.** Étant donné un TA $\mathcal{A}$, on dit que $\mathcal{A}$ is ∃-*ET-opaque* si $\mathit{DVisit}^{priv}(\mathcal{A}) \cap \mathit{DVisit}^{\neg priv}(\mathcal{A}) \neq \emptyset$.

Puis, nous nous intéressons à deux niveaux de définitions, dépendant de l'inclusion simple ou de l'égalité entre ces deux ensembles :

Définition (ET-opacity faible)**.** Étant donné un TA $\mathcal{A}$, on dit que $\mathcal{A}$ is *fortement ET-opaque* si $\mathit{DVisit}^{priv}(\mathcal{A}) \subseteq \mathit{DVisit}^{\neg priv}(\mathcal{A})$.

Définition (ET-opacity forte)**.** Étant donné un TA $\mathcal{A}$, on dit que $\mathcal{A}$ is *fortement ET-opaque* si $\mathit{DVisit}^{priv}(\mathcal{A}) = \mathit{DVisit}^{\neg priv}(\mathcal{A})$.

Problèmes

Nous considérons ensuite plusieurs problèmes découlant de cette définition :

- des problèmes de décisions, afin de déterminer si un TA est ET-opaque ou non ;
- des problèmes de vide (*p-emptiness*) afin de déterminer le vide de l'ensemble des valuations de paramètres tels que le TA évalué vérifie la propriété ;
- des problèmes de synthèse (*p-synthesis*) afin de synthétiser l'ensemble des valuations de paramètres tels que le TA évalué vérifie la propriété.

Résultats de décidabilité

Le Tableau C.2 présente les différents résultats de décidabilité présentés dans cette thèse. Une case verte désigne un problème décidable, lorsqu'une case rouge désigne un problème indécidable.

Tableau C.2 : Résultats de décidabilité pour l'ET-opacity

		∃-ET-opaque	**Faiblement ET-opaque**	**Fortement ET-opaque**
Decision	TA	√	√	√
p-emptiness	L/U-PTA	√	×	×
	PTA	×	×	×
p-synthesis	L/U-PTA	×	×	×
	PTA	×	×	×

Tableau C.3 : Résultats de décidabilité pour l'exp-ET-opacity

		Décision	**Vide**	**Calcul/synthèse**
TA	Faible	√	√	√
	Forte	√	√	?
L/U-PTA	Faible	√	×	×
	Forte	√	×	×
PTA	Faible	√	×	×
	Forte	√	×	×

C.2 *La notion de l'exp-ET-opacity*

Nous étendons ensuite cette définition au cas des secrets avec expiration : dans ce formalisme, après un certain délai, trouver un secret est inutile pour l'attaquant. Nous nous intéressons alors au problème de décision ainsi qu'à celui du calcul de date d'expiration permettant d'assurer qu'un TA est exp-ET-opaque. Une extension paramétrée est également étudiée, avec la synthèse des paramètres dans un PTA. Pour les différents problèmes, nous montrons des résultats de décidablité et proposons quelques algorithmes pour les résoudre.

Le Tableau C.3 présente les différents résultats de décidabilité présentés dans cette thèse. Une case verte désigne un problème décidable, lorsqu'une case rouge désigne un problème indécidable et une case jaune un problème ouvert.

C.3 *Contrôle non temporisé pour l'ET-opacity*

Nous présentons également une première version de contrôle non-temporisé associé à notre formalisme d'ET-opacity. Nous cherchons alors à mettre en évidence un ensemble d'actions de sorte qu'un TA restreint à cet ensemble soit ET-opaque ; un algorithme et une implémentation sont proposés.

ACRONYMES DU RÉSUMÉ EN FRANÇAIS

ET-opacity

Opacité par rapport au temps d'exécution (*execution-time opacity*) 151

ET-opaque

Opaque par rapport au temps d'exécution (*execution-time opaque*) 151

exp-ET-opacity

Opacité par rapport au temps d'exécution avec expiration (*expiring ET-opacity*) 151

exp-ET-opaque

Opaque par rapport au temps d'exécution avec expiration (*expiring ET-opaque*) 152

PTA

Automate temporisé paramétré (*parametric timed automaton*) 149

PZG

Graphe de zones paramétrées (*parametric zone graph*) 150

TA

Automate temporisé (*timed automaton*) 149

BIBLIOGRAPHY

[AA22] Johan Arcile and Étienne André. “Zone Extrapolations in Parametric Timed Automata”. In: *NFM* (June 24–27, 2022). Ed. by Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez. Vol. 13260. Lecture Notes in Computer Science. Pasadena, CA, USA: Springer, 2022, pp. 451–469. DOI: 10.1007/978-3-031-06773-0_24 (cit. on p. 22).

[AA23] Johan Arcile and Étienne André. “Timed automata as a formalism for expressing security: A survey on theory and practice”. In: *ACM Computing Surveys* 55.6 (July 2023), pp. 1–36. DOI: 10.1145/3534967 (cit. on p. 20).

[AAGR19] Étienne André, Paolo Arcaini, Angelo Gargantini, and Marco Radavelli. “Repairing Timed Automata Clock Guards through Abstraction and Testing”. In: *TAP* (Oct. 9–11, 2019). Ed. by Dirk Beyer and Chantal Keller. Vol. 11823. Lecture Notes in Computer Science. Porto, Portugal: Springer, 2019, pp. 129–146. DOI: 10.1007/978-3-030-31157-5_9 (cit. on p. 41).

[AAPP21] Étienne André, Jaime Arias, Laure Petrucci, and Jaco van de Pol. “Iterative Bounded Synthesis for Efficient Cycle Detection in Parametric Timed Automata”. In: *TACAS* (Mar. 27–Apr. 1, 2021). Ed. by Jan Friso Groote and Kim G. Larsen. Vol. 12651. Lecture Notes in Computer Science. Virtual: Springer, 2021, pp. 311–329. DOI: 10.1007/978-3-030-72016-2_17 (cit. on pp. 22, 41, 46, 48, 49, 54, 141).

[ABBCR16] Lăcrămioara Aştefănoaei, Saddek Bensalem, Marius Bozga, Chih-Hong Cheng, and Harald Ruess. “Compositional Parameter Synthesis”. In: *FM* (Nov. 7–11, 2016). Ed. by John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou. Vol. 9995. Lecture Notes in Computer Science. Limassol, Cyprus: Springer, 2016, pp. 60–68. DOI: 10.1007/978-3-319-48989-6_4 (cit. on p. 41).

[ABBDE16] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. “Verifying Constant-Time Implementations”. In: *USENIX Security 16* (Aug. 10–12, 2016). Ed. by Thorsten Holz and Stefan Savage. Austin, TX, USA: USENIX Association, 2016, pp. 53–70 (cit. on p. 13).

[ABBL03] Luca Aceto, Patricia Bouyer, Augusto Burgueño, and Kim Guldstrand Larsen. “The power of reachability testing for timed automata”. In: *Theoretical Computer Science* 300.1-3 (2003), pp. 411–475. DOI: 10.1016/S0304-3975(02)00334-1 (cit. on p. 48).

[ABLM22a] Étienne André, Shapagat Bolat, Engel Lefaucheux, and Dylan Marinho. *Experimental data for paper "strategFTO: Untimed control for timed opacity"*. Oct. 2022. DOI: 10.5281/zenodo.7181848 (cit. on p. 12).

[ABLM22b] Étienne André, Shapagat Bolat, Engel Lefaucheux, and Dylan Marinho. “strategFTO: Untimed control for timed opacity”. In: *Proceedings of the 8th ACM SIGPLAN International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2022)* (Dec. 7, 2022). Ed. by Cyrille Artho and Peter Ölveczky. Auckland, New Zealand: ACM, 2022, pp. 27–33. DOI: 10.1145/3563822.3568013 (cit. on p. 11).

[ABPP19] Étienne André, Vincent Bloemen, Laure Petrucci, and Jaco van de Pol. “Minimal-Time Synthesis for Parametric Timed Automata”. In: *TACAS, Part II* (Apr. 8–11, 2019). Ed. by Tomáš Vojnar and Lijun Zhang. Vol. 11428. Lecture Notes in Computer Science. Prague, Czech Repubic: Springer, 2019, pp. 211–228. DOI: 10.1007/978-3-030-17465-1_12 (cit. on pp. 48, 49, 94).

[ACEF09] 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. 34).

[ACFJL21] Étienne André, Emmanuel Coquard, Laurent Fribourg, Jawher Jerray, and David Lesens. “Parametric Schedulability Analysis of a Launcher Flight Control System under Reactivity Constraints”. In: *Fundamenta Informormaticae* 182.1 (2021), pp. 31–67. DOI: 10.3233/FI-2021-2065 (cit. on p. 18).

[ACN15] Étienne André, Camille Coti, and Hoang Gia Nguyen. “Enhanced Distributed Behavioral Cartography of Parametric Timed Automata”. In: *ICFEM* (Nov. 3–6, 2015). Ed. by Michael Butler, Sylvain Conchon, and Fatiha Zaïdi. Vol. 9407. Lecture Notes in Computer Science. Paris, France: Springer, Nov. 2015, pp. 319–335. DOI: 10.1007/978-3-319-25423-4_21 (cit. on p. 41).

[AD16] Parosh Aziz Abdulla and Giorgio Delzanno. “Parameterized verification”. In: *International Journal on Software Tools for Technology Transfer* 18.5 (2016), pp. 469–473. DOI: 10.1007/s10009-016-0424-3 (cit. on p. 24).

[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. vii, 5, 14, 16, 22, 28–30, 32, 135, 149, 150).

[ADF20] Étienne André, Benoît Delahaye, and Paulin Fournier. "Consistency in Parametric Interval Probabilistic Timed Automata". In: *Journal of Logical and Algebraic Methods in Programming* 110 (2020). DOI: 10.1016/j.jlamp.2019.04.007 (cit. on p. 147).

[ADFL19] Étienne André, Benoît Delahaye, Paulin Fournier, and Didier Lime. "Parametric Timed Broadcast Protocols". In: *VMCAI* (Jan. 13–15, 2019). Ed. by Constantin Enea and Ruzica Piskac. Vol. 11388. Lecture Notes in Computer Science. Lisbon, Portugal: Springer, 2019, pp. 491–512. DOI: 10.1007/978-3-030-11245-5_23 (cit. on pp. 24, 147).

[ADM04] Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. "Multi-Clock Timed Networks". In: *(LICS 2004)* (July 14–17, 2004). Turku, Finland: IEEE Computer Society, 2004, pp. 345–354 (cit. on p. 24).

[ADRST16] Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. "Parameterized verification of time-sensitive models of ad hoc network protocols". In: *Theoretical Computer Science* 612 (2016), pp. 1–22. DOI: 10.1016/j.tcs.2015.07.048 (cit. on p. 24).

[AETYM21] Ikhlass Ammar, Yamen El Touati, Moez Yeddes, and John Mullins. "Bounded opacity for timed systems". In: *Journal of Information Security and Applications* 61 (Sept. 2021), pp. 1–13. ISSN: 2214-2126. DOI: 10.1016/j.jisa.2021.102926 (cit. on pp. 8, 77, 78, 111, 112).

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. "Event-Clock Automata: A Determinizable Class of Timed Automata". In: *Theoretical Computer Science* 211.1-2 (Jan. 1999), pp. 253–273. DOI: 10.1016/S0304-3975(97)00173-4 (cit. on pp. 8, 15).

[AFKS12] 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 p. 42).

[AFS13] Étienne André, Laurent Fribourg, and Romain Soulat. "Merge and Conquer: State Merging in Parametric

Timed Automata". In: *ATVA* (Oct. 15–18, 2013). Ed. by Dang-Van Hung and Mizuhito Ogawa. Vol. 8172. Lecture Notes in Computer Science. Ha Noi, Viet Nam: Springer, Oct. 2013, pp. 381–396. DOI: 10.1007/978-3-319-02444-8_27 (cit. on pp. 22, 53, 54, 57, 64, 69).

[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 pp. vii, 7, 16, 17, 31, 32, 37, 86–88, 108, 149).

[AJ03] Parosh Aziz Abdulla and Bengt Jonsson. "Model checking of systems with many identical timed processes". In: *Theoretical Computer Science* 290.1 (2003), pp. 241–264. DOI: 10.1016/S0304-3975(01)00330-9 (cit. on p. 24).

[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. Singapore: IEEE, 2020, pp. 37–42. DOI: 10.1109/ICECCS51672.2020.00012 (cit. on p. 19).

[AL17a] Étienne André and Didier Lime. "Liveness in L/U-Parametric Timed Automata". In: *ACSD* (June 25–30, 2017). Ed. by Alex Legay and Klaus Schneider. Zaragoza, Spain: IEEE, 2017, pp. 9–18. DOI: 10.1109/ACSD.2017.19 (cit. on pp. 17, 87, 94).

[AL17b] Étienne André and Shang-Wei Lin. "Learning-based compositional parameter synthesis for event-recording automata". In: *FORTE* (June 17–22, 2017). Ed. by Ahmed Bouajjani and Alexandra Silva. Vol. 10321. Lecture Notes in Computer Science. Neuchâtel, Switzerland: Springer, 2017, pp. 17–32. DOI: 10.1007/978-3-319-60225-7_2 (cit. on p. 41).

[ALM20] Étienne André, Didier Lime, and Nicolas Markey. "Language Preservation Problems in Parametric Timed Automata". In: *Logical Methods in Computer Science* 16.1 (Jan. 2020), 5:1–5:31. DOI: 10.23638/LMCS-16(1:5)2020 (cit. on pp. 49, 87, 91, 94–96).

[ALM23] Étienne André, Engel Lefaucheux, and Dylan Marinho. "Expiring opacity problems in parametric timed automata". In: *Proceedings of the 27th International Conference on Engineering of Complex Computer Systems (ICECCS 2023)* (June 12–16, 2023). Ed. by Yamine Ait-Ameur and Ferhat Khendek. Vol. 13260. Toulouse, France: Springer, 2023, pp. 451–469. DOI: 10.1109/ICECCS59891.2023.00020 (cit. on p. 11).

[ALMS22] 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* 31.4 (2022), 64:1–64:36. DOI: 10.1145/3502851 (cit. on pp. 10, 79, 108, 184).

[ALR15] Étienne André, Didier Lime, and Olivier H. Roux. “Integer-Complete Synthesis for Bounded Parametric Timed Automata”. In: *RP* (Sept. 21–23, 2015). Ed. by Mikołaj Bojańczyk, Sławomir Lasota, and Igor Potapov. Vol. 9328. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Sept. 2015, pp. 7–19. DOI: 10.1007/978-3-319-24537-9_2 (cit. on p. 22).

[ALR16] Étienne André, Didier Lime, and Olivier H. Roux. “Decision Problems for Parametric Timed Automata”. In: *ICFEM* (Nov. 14–18, 2016). Ed. by Kazuhiro Ogata, Mark Lawford, and Shaoying Liu. Vol. 10009. Lecture Notes in Computer Science. Tokyo, Japan: Springer, 2016, pp. 400–416. DOI: 10.1007/978-3-319-47846-3_25 (cit. on p. 17).

[ALR18] Étienne André, Didier Lime, and Mathias Ramparison. “TCTL model checking lower/upper-bound parametric timed automata without invariants”. In: *FORMATS* (Sept. 4–6, 2018). Ed. by David N. Jansen and Pavithra Prabhakar. Vol. 11022. Lecture Notes in Computer Science. Beijing, China: Springer, 2018, pp. 1–17. DOI: 10.1007/978-3-030-00151-3_3 (cit. on pp. 87, 94).

[ALR19] Étienne André, Didier Lime, and Mathias Ramparison. “Parametric updates in parametric timed automata”. In: *FORTE* (June 17–21, 2019). Ed. by Jorge A. Pérez and Nobuko Yoshida. Vol. 11535. Lecture Notes in Computer Science. Copenhagen, Denmark: Springer, 2019, pp. 39–56. DOI: 10.1007/978-3-030-21759-4_3 (cit. on p. 45).

[ALR22] Étienne André, Didier Lime, and Olivier H. Roux. “Reachability and liveness in parametric timed automata”. In: *Logical Methods in Computer Science* 18.1 (Feb. 2022), 31:1–31:41. DOI: 10.46298/lmcs-18(1:31)2022 (cit. on p. 36).

[ALRS21] Étienne André, Didier Lime, Mathias Ramparison, and Mariëlle Stoelinga. “Parametric analyses of attack-fault trees”. In: *Fundamenta Informormaticae* 182.1 (Sept. 2021), pp. 69–94. DOI: 10.3233/FI-2021-2066 (cit. on p. 17).

[Alu+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. “The Algorithmic Analysis of Hybrid Systems”. In:

Theoretical Computer Science 138.1 (1995), pp. 3–34. DOI: 10.1016/0304-3975(94)00202-T (cit. on pp. 41–43).

[AM04] Rajeev Alur and P. Madhusudan. "Decision Problems for Timed Automata: A Survey". In: Lecture Notes in Computer Science 3185 (2004). Ed. by Marco Bernardo and Flavio Corradini, pp. 1–24. DOI: 10.1007/978-3-540-30080-9_1 (cit. on pp. 14, 15).

[Amp+19] Elvio Amparore, Bernard Berthomieu, Gianfranco Ciardo, Silvano Dal Zilio, Francesco Gallà, Lom Messan Hillah, Francis Hulin Hubard, Peter Gjøl Jensen, Loïg Jezequel, Fabrice Kordon, Didier Le Botlan, Torsten Liebke, Jeroen Meijer, Andrew Miner, Emmanuel Paviot-Adet, Jiří Srba, Yann Thierry-Mieg, Tom van Dijk, and Karsten Wolf. "Presentation of the 9th Edition of the Model Checking Contest". In: *TACAS* (Apr. 6–11, 2019). Ed. by Dirk Beyer, Marieke Huisman, Fabrice Kordon, and Bernhard Steffen. Vol. 11429. Lecture Notes in Computer Science. Prague, Czech Republic: Springer, 2019, pp. 50–68. DOI: 10.1007/978-3-030-17502-3_4 (cit. on p. 24).

[AMP21a] Étienne André, Dylan Marinho, and Jaco van de Pol. "A Benchmarks Library for Extended Parametric Timed Automata". In: *Proceedings of the 15th International Conference on Tests and Proofs (TAP 2021), Held as Part of STAF 2021* (June 21–22, 2021). Ed. by Frédéric Loulergue and Franz Wotawa. Vol. 12740. Lecture Notes in Computer Science. Virtual Event: Springer, 2021, pp. 39–50. DOI: 10.1007/978-3-030-79379-1_3 (cit. on pp. 9, 141).

[AMP21b] Étienne André, Dylan Marinho, and Jaco van de Pol. *The IMITATOR benchmarks library 2.0: A benchmarks library for extended parametric timed automata*. Version 2.0. Apr. 2021. DOI: 10.5281/zenodo.4730980 (cit. on pp. 11, 46).

[AMP21c] Étienne André, Dylan Marinho, and Jaco van de Pol. *The IMITATOR benchmarks library v2.0*. 2021. URL: https://www.imitator.fr/static/library2/ (cit. on pp. 43, 46, 150).

[AMPP22a] Étienne André, Dylan Marinho, Laure Petrucci, and Jaco van de Pol. "Efficient Convex Zone Merging in Parametric Timed Automata". In: *Proceedings of the 20th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2022)* (Sept. 13–15, 2022). Ed. by Sergiy Bogomolov and David Parker. Vol. 13465. Lecture Notes in Computer Science. Warsaw, Poland: Springer, 2022, pp. 200–218. DOI: 10.1007/978-3-031-15839-1_12 (cit. on p. 10).

[AMPP22b] Étienne André, Dylan Marinho, Laure Petrucci, and Jaco van de Pol. *Data for paper "Efficient Convex Zone Merging in Parametric Timed Automata"*. Version 1.0. July 2022. DOI: 10.5281/zenodo.6806915 (cit. on p. 11).

[And13] Étienne André. "Observer Patterns for Real-Time Systems". In: *ICECCS* (July 17–19, 2013). Ed. by Yang Liu and Andrew Martin. Singapore: IEEE Computer Society, July 2013, pp. 125–134. DOI: 10.1109/ICECCS.2013.26 (cit. on pp. 48, 49).

[And16] Étienne André. "Parametric Deadlock-Freeness Checking Timed Automata". In: *ICTAC* (Oct. 24–28, 2016). Ed. by Augusto Cesar Alves Sampaio and Farn Wang. Vol. 9965. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, 2016, pp. 469–478. DOI: 10.1007/978-3-319-46750-4_27 (cit. on pp. 41, 48, 49).

[And18] Étienne André. *The IMITATOR benchmarks library v1.0*. 2018. URL: https://www.imitator.fr/library1.html (cit. on pp. 24, 42, 43, 150).

[And19a] Étienne André. "A benchmark library for parametric timed model checking". In: *FTSCS* (Nov. 16, 2018). Ed. by Cyrille Artho and Peter Csaba Ölveczky. Vol. 1008. Communications in Computer and Information Science. Gold Coast, Australia: Springer, 2019, pp. 75–83. DOI: 10.1007/978-3-030-12988-0_5 (cit. on pp. 24, 42, 46, 150).

[And19b] 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 pp. 17, 53, 86–88, 108).

[And21a] Étienne André. "IMITATOR 3: Synthesis of timing parameters beyond decidability". In: *CAV* (July 18–23, 2021). Ed. by Rustan Leino and Alexandra Silva. Lecture Notes in Computer Science. virtual: Springer, 2021. DOI: 10.1007/978-3-030-81685-8_26 (cit. on pp. 7, 17, 37, 42, 43).

[And21b] Étienne André. *IMITATOR user manual*. 3.1. 2021 (cit. on p. 46).

[ANP17] Étienne André, Hoang Gia Nguyen, and Laure Petrucci. "Efficient parameter synthesis using optimized state exploration strategies". In: *ICECCS* (Nov. 6–8, 2017). Ed. by Zhenjiang Hu and Guangdong Bai. Fukuoka, Japan: IEEE, 2017, pp. 1–10. DOI: 10.1109/ICECCS.2017.28 (cit. on p. 22).

[ARCH21] *ARCH Benchmarks*. 2021. URL: https://cps-vo.org/group/ARCH/benchmarks (cit. on p. 24).

[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. 100).

[Bac+21] Giovanni Bacci, Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Pierre-Alain Reynier. "Optimal and robust controller synthesis using energy timed automata with uncertainty". In: *Formal Aspects of Computing* 33.1 (2021), pp. 3–25. DOI: 10.1007/s00165-020-00521-4 (cit. on p. 21).

[BB07] Andrew Bortz and Dan Boneh. "Exposing private information by timing Web applications". In: *WWW* (May 8–12, 2007). Ed. by Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy. Banff, Alberta, Canada: ACM, 2007, pp. 621–628. DOI: 10.1145/1242572.1242656 (cit. on p. 19).

[BBBČ16] Peterand Bezděk, Nikolaand Beneš, Jiří Barnat, and Ivana Černá. "LTL Parameter Synthesis of Parametric Timed Automata". In: *SEFM* (July 4–8, 2016). Ed. by Rocco De Nicola and eva Kühn. Vol. 9763. Lecture Notes in Computer Science. Vienna, Austria: Springer, 2016, pp. 172–187. DOI: 10.1007/978-3-319-41591-8_12 (cit. on pp. 22, 41).

[BBLP06] Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. "Lower and upper bounds in zone-based abstractions of timed automata". In: *International Journal on Software Tools for Technology Transfer* 8.3 (Sept. 2006), pp. 204–215. DOI: 10.1007/s10009-005-0190-0 (cit. on p. 22).

[BBLS15] Nikola Beneš, Peter Bezděk, Kim Gulstrand Larsen, and Jiří Srba. "Language Emptiness of Continuous-Time Parametric Timed Automata". In: *ICALP, Part II* (July 6–10, 2015). Ed. by Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann. Vol. 9135. Lecture Notes in Computer Science. Kyoto, Japan: Springer, July 2015, pp. 69–81. DOI: 10.1007/978-3-662-47666-6_6 (cit. on pp. 86, 87).

[BC11] Jean-Luc Béchennec and Franck Cassez. "Computation of WCET using Program Slicing and Real-Time Model-Checking". In: *CoRR* abs/1105.1633 (2011). arXiv: 1105.1633 (cit. on p. 16).

[BCD05] Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza. "Fault Diagnosis Using Timed Automata". In: *FoSSaCS* (Apr. 4–8, 2005). Ed. by Vladimiro Sassone. Vol. 3441. Lecture Notes in Computer Science. Edinburgh, UK: Springer, 2005, pp. 219–233. DOI: 10.1007/978-3-540-31982-5_14 (cit. on p. 19).

[BCLR15] 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 pp. 19, 20, 103, 104, 107).

[BDFST02] Roberto Barbuti, Nicoletta De Francesco, Antonella Santone, and Luca Tesei. "A Notion of Non-Interference for Timed Automata". In: *Fundamenta Informormaticae* 51.1-2 (2002), pp. 1–11 (cit. on p. 19).

[BDR08] Véronique Bruyére, Emmanuel Dall'Olio, and Jean-Francois Raskin. "Durations and parametric model-checking in timed automata". In: *ACM Transactions on Computational Logic* 9.2 (Apr. 2008), 12:1–12:23. DOI: 10.1145/1342991.1342996 (cit. on pp. 30, 31, 74, 75, 84, 119, 122).

[BDR11] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. "Secure information flow by self-composition". In: *Mathematical Structures in Computer Science* 21.6 (2011), pp. 1207–1252. DOI: 10.1017/S0960129511000193 (cit. on p. 97).

[Bea03] Danièle Beauquier. "On probabilistic timed automata". In: *Theoretical Computer Science* 292.1 (2003), pp. 65–84. DOI: 10.1016/S0304-3975(01)00215-8 (cit. on p. 21).

[Beh+07] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. "UPPAAL-Tiga: Time for Playing Games!" In: *CAV 2007* (July 3–7, 2007). Ed. by Werner Damm and Holger Hermanns. Vol. 4590. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2007, pp. 121–125. DOI: 10.1007/978-3-540-73368-3_14 (cit. on p. 21).

[Ber+01] Valérie Bertin, Etienne Closse, Michel Poize, Jacques Pulou, Joseph Sifakis, Patrick Venier, Daniel Weil, and Sergio Yovine. "TAXYS=Esterel+Kronos. A tool for verifying real-time properties of embedded systems". In: *CDC* (Dec. 4–7, 2001). Orlando, FL, USA: IEEE, 2001, pp. 2875–2880. DOI: 10.1109/.2001.980712 (cit. on p. 15).

[BF13] Nathalie Bertrand and Paulin Fournier. "Parameterized Verification of Many Identical Probabilistic Timed Processes". In: *FSTTCS* (Dec. 12–14, 2013). Ed. by Anil Seth and Nisheeth K. Vishnoi. Vol. 24. LIPIcs. Guwahati, India: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013, pp. 501–513. DOI: 10.4230/LIPIcs.FSTTCS.2013.501 (cit. on p. 24).

[BFFPS20] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. "Reachability Analysis of Linear Hybrid Systems via Block De-composition". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (Nov. 2020), pp. 4018–4029. DOI: 10.1109/TCAD.2020.3012859 (cit. on p. 22).

[BFS14] Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. "Playing with Probabilities in Reconfigurable Broadcast Networks". In: *FOSSACS 2014* (Apr. 5–13, 2014). Ed. by Anca Muscholl. Vol. 8412. Lecture Notes in Computer Science. Grenoble, France: Springer, 2014, pp. 134–148. DOI: 10.1007/978-3-642-54830-7_9 (cit. on p. 24).

[BHZ08] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. "The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems". In: *Science of Computer Programming* 72.1–2 (2008), pp. 3–21. DOI: 10.1016/j.scico.2007.08.001 (cit. on pp. 37, 132).

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008. ISBN: 978-0-262-02649-9 (cit. on pp. 4, 26, 56, 183).

[BKMR08] Jeremy W. Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. "Opacity generalised to transition systems". In: *International Journal of Information Security* 7.6 (2008), pp. 421–435. DOI: 10.1007/s10207-008-0058-x (cit. on pp. 7, 19).

[BL09] Laura Bozzelli and Salvatore La Torre. "Decision problems for lower/upper bound parametric timed automata". In: *Formal Methods in System Design* 35.2 (2009), pp. 121–151. DOI: 10.1007/s10703-009-0074-0 (cit. on pp. 36, 87, 88, 93).

[BO14] Daniel Bundala and Joël Ouaknine. "Advances in Parametric Real-Time Reasoning". In: *MFCS, Part I* (Aug. 25–29, 2014). Ed. by Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik. Vol. 8634. Lecture Notes in Computer Science. Budapest, Hungary: Springer, 2014, pp. 123–134. ISBN: 978-3-662-44521-1. DOI: 10.1007/978-3-662-44522-8 (cit. on p. 108).

[BPS20] Borzoo Bonakdarpour, Pavithra Prabhakar, and César Sánchez. "Model Checking Timed Hyperproperties in Discrete-Time Systems". In: *NFM 2020* (May 11–15, 2020). Ed. by Ritchie Lee, Susmit Jha, and Anastasia Mavridou. Vol. 12229. Lecture Notes in Computer Science. Moffett Field, CA, USA: Springer, 2020, pp. 311–328. DOI: 10.1007/978-3-030-55754-6_18 (cit. on p. 14).

[BPT19] Sandrine Blazy, David Pichardie, and Alix Trieu. "Verifying constant-time implementations by abstract interpretation". In: *Journal of Computer Security* 27.1 (2019), pp. 137–163. DOI: 10.3233/JCS-181136 (cit. on p. 13).

[Bry86] Randal E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Trans. Computers* 35.8 (1986), pp. 677–691. DOI: 10.1109/TC.1986.1676819 (cit. on p. 141).

[BSBM06] Ramzi Ben Salah, Marius Bozga, and Oded Maler. "On Interleaving in Timed Automata". In: *CONCUR* (Aug. 27–30, 2006). Ed. by Christel Baier and Holger Hermanns. Vol. 4137. Lecture Notes in Computer Science. Bonn, Germany: Springer, 2006, pp. 465–476. ISBN: 3-540-37376-4. DOI: 10.1007/11817949_31 (cit. on p. 23).

[BSBM09] Ramzi Ben Salah, Marius Bozga, and Oded Maler. "Compositional timing analysis". In: *EMSOFT* (Oct. 12–16, 2009). Ed. by Samarjit Chakraborty and Nicolas Halbwachs. Grenoble, France: ACM, 2009, pp. 39–48. DOI: 10.1145/1629335.1629342 (cit. on p. 23).

[BT03] Roberto Barbuti and Luca Tesei. "A Decidable Notion of Timed Non-Interference". In: *Fundamenta Informormaticae* 54.2-3 (2003), pp. 137–150 (cit. on p. 19).

[Bud+17] Carlos E. Budde, Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges, and Andrea Turrini. "JANI: Quantitative Model and Tool Interaction". In: *TACAS, Part II* (Apr. 22–29, 2017). Ed. by Axel Legay and Tiziana Margaria. Vol. 10206. Lecture Notes in Computer Science. Uppsala, Sweden: Springer, 2017, pp. 151–168. DOI: 10.1007/978-3-662-54580-5_9 (cit. on pp. 23, 43, 46).

[BZ18] Anna Becchi and Enea Zaffanella. "An Efficient Abstract Domain for Not Necessarily Closed Polyhedra". In: *SAS* (Aug. 29–31, 2018). Ed. by Andreas Podelski. Vol. 11002. Lecture Notes in Computer Science. Freiburg, Germany: Springer, 2018, pp. 146–165. DOI: 10.1007/978-3-319-99725-4_11 (cit. on p. 69).

[BZ20] Anna Becchi and Enea Zaffanella. "PPLite: Zero-overhead encoding of NNC polyhedra". In: *Information and Computation* 275 (Dec. 2020), pp. 1–36. DOI: 10.1016/j.ic.2020.104620 (cit. on p. 69).

[CÁF11] Xin Chen, Erika Ábrahám, and Goran Frehse. "Efficient Bounded Reachability Computation for Rectangular Automata". In: *RP* (Sept. 28–30, 2011). Ed. by Giorgio Delzanno and Igor Potapov. Vol. 6945. Lecture Notes in Computer Science. Genoa, Italy: Springer, 2011, pp. 139–152. DOI: 10.1007/978-3-642-24288-5_13 (cit. on p. 22).

[CAS01] Aurore Collomb-Annichini and Mihaela Sighireanu. "Parameterized Reachability Analysis of the IEEE 1394 Root Contention Protocol using TReX". In: *RT-TOOLS*. Alborg, Danemark, 2001 (cit. on p. 18).

[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 pp. 7, 8, 77, 78, 83, 111, 148).

[CDFLL05] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. "Efficient On-the-Fly Algorithms for the Analysis of Timed Games". In: *CONCUR 2005* (Aug. 23–26, 2005). Ed. by Martín Abadi and Luca de Alfaro. Vol. 3653. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, 2005, pp. 66–80. DOI: 10.1007/11539452_9 (cit. on p. 21).

[CE81] Edmund M. Clarke and E. Allen Emerson. "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic". In: *Logics of Programs, Workshop* (May 1981). Ed. by Dexter Kozen. Vol. 131. Lecture Notes in Computer Science. Yorktown Heights, New York, USA: Springer, 1981, pp. 52–71. DOI: 10.1007/BFb0025774 (cit. on p. 182).

[CH23] Marek Chalupa and Thomas A. Henzinger. "Monitoring Hyperproperties with Prefix Transducers". In: *Proceedings of the 23rd International Conference on Runtime Verification, RV 2023* (Oct. 3–6, 2023). Ed. by Panagiotis Katsaros and Laura Nenzi. Vol. 14245. Lecture Notes in Computer Science. Thessaloniki, Greece: Springer, 2023, pp. 168–190. DOI: 10.1007/978-3-031-44267-4_9 (cit. on p. 14).

[Che+15] Xin Chen, Stefan Schupp, Ibtissem Ben Makhlouf, Erika Ábrahám, Goran Frehse, and Stefan Kowalewski. "A Benchmark Suite for Hybrid Systems Reachability Analysis". In: *NFM* (Apr. 27–29, 2015). Ed. by Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi. Vol. 9058. Lecture Notes in Computer Science. Pasadena, CA, USA: Springer, 2015, pp. 408–414. DOI: 10.1007/978-3-319-17524-9_29 (cit. on p. 23).

[CHSJX22] Aidong Chen, Chen Hong, Xinna Shang, Hongyuan Jing, and Sen Xu. "Timing leakage to break SM2 signature algorithm". In: *Journal of Information Security and*

Applications 67 (2022), p. 103210. DOI: 10.1016/j.jisa.2022.103210 (cit. on p. 4).

[CJL17] Franck Cassez, Peter Gjøl Jensen, and Kim Guldstrand Larsen. "Refinement of Trace Abstraction for Real-Time Programs". In: *RP* (Sept. 7–9, 2017). Ed. by Matthew Hague and Igor Potapov. Vol. 10506. Lecture Notes in Computer Science. London, UK: Springer, 2017, pp. 42–58. DOI: 10.1007/978-3-319-67089-8_4 (cit. on p. 18).

[CJLRR09] Franck Cassez, Jan Jakob Jessen, Kim Guldstrand Larsen, Jean-François Raskin, and Pierre-Alain Reynier. "Automatic Synthesis of Robust and Optimal Controllers - An Industrial Case Study". In: *HSCC 2009* (Apr. 13–15, 2009). Ed. by Rupak Majumdar and Paulo Tabuada. Vol. 5469. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, 2009, pp. 90–104. DOI: 10.1007/978-3-642-00602-9_7 (cit. on p. 21).

[CL00] Franck Cassez and Kim Guldstrand Larsen. "The Impressive Power of Stopwatches". In: *CONCUR* (Aug. 22–25, 2000). Ed. by Catuscia Palamidessi. Vol. 1877. Lecture Notes in Computer Science. University Park, PA, USA: Springer, 2000, pp. 138–152. DOI: 10.1007/3-540-44618-4_12 (cit. on pp. 43, 44).

[Cla+14] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. "Temporal Logics for Hyperproperties". In: *POST 2014* (Apr. 5–13, 2014). Ed. by Martín Abadi and Steve Kremer. Vol. 8414. Lecture Notes in Computer Science. Grenoble, France: Springer, 2014, pp. 265–284. DOI: 10.1007/978-3-642-54792-8_15 (cit. on p. 14).

[CPR08] Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian. "Symbolic Computation of Schedulability Regions Using Parametric Timed Automata". In: *RTSS* (Nov. 30–Dec. 3, 2008). Barcelona, Spain: IEEE Computer Society, 2008, pp. 80–89. DOI: 10.1109/RTSS.2008.36 (cit. on p. 41).

[CSÁ14] Xin Chen, Sriram Sankaranarayanan, and Erika Ábrahám. "Under-approximate flowpipes for non-linear continuous systems". In: *FMCAD* (Oct. 21–24, 2014). Lausanne, Switzerland: IEEE, 2014, pp. 59–66. DOI: 10.1109/FMCAD.2014.6987596 (cit. on p. 22).

[CT13] Franck Cassez and Stavros Tripakis. "Fault Diagnosis of Timed Systems". In: *Communicating Embedded Systems*. Ed. by Claude Jard and Olivier H. Roux. Wiley, 2013, pp. 107–138. DOI: 10.1002/9781118558188.ch4 (cit. on p. 19).

[Dav05] Alexandre David. "Merging DBMs Efficiently". In: *NWPT* (Oct. 19–21, 2005). Copenhagen, Denmark: DIKU, University of Copenhagen, 2005, pp. 54–56 (cit. on pp. 22, 53).

[Doy07] Laurent Doyen. "Robust Parametric Reachability for Timed Automata". In: *Information Processing Letters* 102.5 (2007), pp. 208–213. DOI: 10.1016/j.ipl.2006.11.018 (cit. on pp. 86, 87).

[EDLR16] Yrvann Emzivat, Benoît Delahaye, Didier Lime, and Olivier H. Roux. "Probabilistic Time Petri Nets". In: *PETRI NETS 2016* (June 19–24, 2016). Ed. by Fabrice Kordon and Daniel Moldt. Vol. 9698. Lecture Notes in Computer Science. Toruń, Poland: Springer, 2016, pp. 261–280. DOI: 10.1007/978-3-319-39086-4_16 (cit. on p. 147).

[ELFL21] Mathieu Escouteloup, Ronan Lashermes, Jacques Fournier, and Jean-Louis Lanet. "Under the Dome: Preventing Hardware Timing Information Leakage". In: *CARDIS 2021* (Nov. 11–12, 2021). Ed. by Vincent Grosso and Thomas Pöppelmann. Vol. 13173. Lecture Notes in Computer Science. Lübeck, Germany: Springer, 2021, pp. 233–253. DOI: 10.1007/978-3-030-97348-3_13 (cit. on p. 14).

[FH20] Mohammed Foughali and Pierre-Emmanuel Hladik. "Bridging the gap between formal verification and schedulability analysis: The case of robotics". In: *Journal of Systems Architecture* 111 (2020). ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2020.101817 (cit. on p. 16).

[FHST18] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. "RVHyper: A Runtime Verification Tool for Temporal Hyperproperties". In: *TACAS 2018* (Apr. 14–20, 2018). Ed. by Dirk Beyer and Marieke Huisman. Vol. 10806. Lecture Notes in Computer Science. Thessaloniki, Greece: Springer, 2018, pp. 194–200. DOI: 10.1007/978-3-319-89963-3_11 (cit. on p. 14).

[FHST19] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. "Monitoring hyperproperties". In: *Formal Methods in System Design* 54.3 (2019), pp. 336–363. DOI: 10.1007/S10703-019-00334-Z (cit. on p. 14).

[FHST20] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. "Efficient monitoring of hyperproperties using prefix trees". In: *International Journal on Software Tools for Technology Transfer* 22.6 (2020), pp. 729–740. DOI: 10.1007/S10009-020-00552-5 (cit. on p. 14).

[FI04] Ansgar Fehnker and Franjo Ivancic. "Benchmarks for Hybrid Systems Verification". In: *HSCC* (Mar. 25–27, 2004). Ed. by Rajeev Alur and George J. Pappas.

Vol. 2993. Lecture Notes in Computer Science. Philadelphia, PA, USA: Springer, 2004, pp. 326–341. DOI: 10.1007/978-3-540-24743-2_22 (cit. on p. 23).

[FJ13] Léa Fanchon and Florent Jacquemard. "Formal Timing Analysis Of Mixed Music Scores". In: *ICMC* (Aug. 12–16, 2013). Perth, Australia: Michigan Publishing, Aug. 2013 (cit. on p. 18).

[FK13] Laurent Fribourg and Ulrich Kühne. "Parametric Verification and Test Coverage for Hybrid Automata Using the Inverse Method". In: *International Journal of Foundations of Computer Science* 24.2 (2013), pp. 233–249. DOI: 10.1142/S0129054113400091 (cit. on p. 41).

[FM15] Yliès Falcone and Hervé Marchand. "Enforcement and validation (at runtime) of various notions of opacity". In: *Discrete Event Dynamic Systems* 25.4 (2015), pp. 531–570. DOI: 10.1007/s10626-014-0196-4 (cit. on pp. 21, 141).

[Fre+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. "SpaceEx: Scalable Verification of Hybrid Systems". In: *CAV* (July 14–20, 2011). Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Vol. 6806. Lecture Notes in Computer Science. Snowbird, UT, USA: Springer, 2011, pp. 379–395. DOI: 10.1007/978-3-642-22110-1_30 (cit. on pp. 8, 17).

[Fre+19] Goran Frehse, Alessandro Abate, Dieky Adzkiya, Anna Becchi, Lei Bu, Alessandro Cimatti, Mirco Giacobbe, Alberto Griggio, Sergio Mover, Muhammad Syifa'ul Mufid, Idriss Riouak, Stefano Tonetta, and Enea Zaffanella. "ARCH-COMP19 Category Report: Hybrid Systems with Piecewise Constant Dynamics". In: *ARCH@CPSIoTWeek* (Apr. 15, 2019). Ed. by Goran Frehse and Matthias Althoff. Vol. 61. EPiC Series in Computing. Montréal, QC, Canada: EasyChair, 2019, pp. 1–13. DOI: 10.29007/rjwn (cit. on p. 24).

[FRS15] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. "Algorithms for Model Checking HyperLTL and HyperCTL ^*". In: *CAV 2015* (July 18–24, 2015). Ed. by Daniel Kroening and Corina S. Pasareanu. Vol. 9206. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, 2015, pp. 30–48. DOI: 10.1007/978-3-319-21690-4_3 (cit. on p. 14).

[FS00] Edward W. Felten and Michael A. Schneider. "Timing attacks on Web privacy". In: *CCS* (Nov. 1–4, 2000). Ed. by Dimitris Gritzalis, Sushil Jajodia, and Pierangela

Samarati. Athens, Greece: ACM, 2000, pp. 25–32. DOI: 10.1145/352600.352606 (cit. on p. 19).

[FV19] Azadeh Farzan and Anthony Vandikas. “Automated Hypersafety Verification”. In: *CAV 2019*. Ed. by Isil Dillig and Serdar Tasiran. Vol. 11561. Lecture Notes in Computer Science. New York City, NY, USA: Springer, 2019, pp. 200–218. DOI: 10.1007/978-3-030-25540-4_11 (cit. on p. 14).

[GH21] Stefan Göller and Mathieu Hilaire. “Reachability in Two-Parametric Timed Automata with One Parameter Is EXPSPACE-Complete”. In: *STACS* (Mar. 16–19, 2021). Ed. by Markus Bläser and Benjamin Monmege. Vol. 187. LIPIcs. Saarbrücken, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 36:1–36:18. DOI: 10.4230/LIPIcs.STACS.2021.36 (cit. on p. 108).

[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 pp. 19, 103, 104, 107).

[Gol21] Ebru Aydin Gol. “Control Synthesis for Parametric Timed Automata under Unavoidability Specifications”. In: *ECC* (June 29–July 2, 2021). Virtual Event / Delft, The Netherlands: IEEE, 2021, pp. 740–745. DOI: 10.23919/ECC54610.2021.9655222 (cit. on p. 141).

[GR21] Damas P. Gruska and M. Carmen Ruiz. “Process Opacity and Insertion Functions”. In: *CS&P* (Sept. 27–28, 2021). Ed. by Holger Schlingloff and Thomas Vogel. Vol. 2951. CEUR Workshop Proceedings. Berlin, Germany: CEUR-WS.org, 2021, pp. 83–92 (cit. on p. 14).

[Gri] *Grid’5000*. URL: https://www.grid5000.fr (cit. on pp. 50, 64).

[GSB18] Christopher Gerking, David Schubert, and Eric Bodden. “Model Checking the Information Flow Security of Real-Time Systems”. In: *ESSoS* (June 26–27, 2018). Ed. by Mathias Payer, Awais Rashid, and Jose M. Such. Vol. 10953. Lecture Notes in Computer Science. Paris, France: Springer, 2018, pp. 27–43. DOI: 10.1007/978-3-319-94496-8_3 (cit. on p. 20).

[HAF15] Bardh Hoxha, Houssam Abbas, and Georgios E. Fainekos. “Benchmarks for Temporal Logic Requirements for Automotive Systems”. In: *ARCH@CPSWeek*. Ed. by Goran Frehse and Matthias Althoff. Vol. 34. EPiC Series in Computing. Berlin, Germany and Seattle, WA, USA: EasyChair, 2015, pp. 25–30. DOI: 10.29007/xwrs (cit. on p. 23).

[Hah19] Christopher Hahn. “Algorithms for Monitoring Hyperproperties”. In: *RV 2019* (Oct. 8–11, 2019). Ed. by Bernd Finkbeiner and Leonardo Mariani. Vol. 11757. Lecture Notes in Computer Science. Porto, Portugal: Springer, 2019, pp. 70–90. DOI: 10.1007/978-3-030-32079-9_5 (cit. on p. 14).

[Hen96] Thomas A. Henzinger. “The Theory of Hybrid Automata”. In: *LiCS*. Ed. by Moshe Y. Vardi and Edmund M. Clarke. New Brunswick, New Jersey, USA: IEEE Computer Society, 1996, pp. 278–292. DOI: 10.1109/LICS.1996.561342 (cit. on p. 22).

[HHWT95] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. “A User Guide to HyTech”. In: *TACAS* (May 19–20, 1995). Ed. by Ed Brinksma, Rance Cleaveland, Kim Guldstrand Larsen, Tiziana Margaria, and Bernhard Steffen. Vol. 1019. Lecture Notes in Computer Science. Aarhus, Denmark: Springer, 1995, pp. 41–71. ISBN: 3-540-60630-0. DOI: 10.1007/3-540-60630-0_3 (cit. on pp. 17, 46).

[HMP91] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. “Temporal Proof Methodologies for Real-time Systems”. In: *POPL* (Jan. 21–23, 1991). Ed. by David S. Wise. Orlando, Florida, USA: ACM Press, 1991, pp. 353–366. DOI: 10.1145/99583.99629 (cit. on pp. 26, 27).

[HPT] Frédéric Herbreteau, Gerald Point, and Thanh-Tung Tran. *The TChecker tool and librairies*. URL: https://github.com/ticktac-project/tchecker (cit. on pp. 6, 15).

[HRSV02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. “Linear parametric model checking of timed automata”. In: *Journal of Logic and Algebraic Programming* 52-53 (2002). DOI: 10.1016/S1567-8326(02)00037-1 (cit. on pp. 22, 34–37, 53, 57, 87, 88, 103).

[HSLL97] Klaus Havelund, Arne Skou, Kim Guldstrand Larsen, and Kristian Lund. “Formal modeling and analysis of an audio/video protocol: an industrial case study using UPPAAL”. In: *RTSS’97* (Dec. 5–3, 1997). San Francisco, CA, USA: IEEE Computer Society, 1997, pp. 2–13. DOI: 10.1109/REAL.1997.641264 (cit. on p. 15).

[HSTW20] Frédéric Herbreteau, B. Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz. “Why Liveness for Timed Automata Is Hard, and What We Can Do About It”. In: *ACM Transactions on Computational Logic* 21.3 (Mar. 2020), 17:1–17:28. DOI: 10.1145/3372310 (cit. on p. 22).

[HSW12] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. “Efficient emptiness check for timed Büchi automata”. In: *Formal Methods in System Design*

40.2 (Dec. 2012), pp. 122–146. DOI: 10.1007/s10703-011-0133-1 (cit. on p. 22).

[HSW13] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. "Lazy Abstractions for Timed Automata". In: *CAV* (July 13–19, 2013). Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Saint Petersburg, Russia: Springer, 2013, pp. 990–1005. DOI: 10.1007/978-3-642-39799-8_71 (cit. on p. 22).

[HSW16] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. "Better abstractions for timed automata". In: *Information and Computation* 251 (Dec. 2016), pp. 67–90. DOI: 10.1016/j.ic.2016.07.004 (cit. on p. 22).

[HT15] Frédéric Herbreteau and Thanh-Tung Tran. "Improving Search Order for Reachability Testing in Timed Automata". In: *FORMATS* (Sept. 2–4, 2015). Ed. by Sriram Sankaranarayanan and Enrico Vicario. Vol. 9268. Lecture Notes in Computer Science. Madrid, Spain: Springer, 2015, pp. 124–139. DOI: 10.1007/978-3-319-22975-1_9 (cit. on p. 22).

[HZJ21] Hsi-Ming Ho, Ruoyu Zhou, and Timothy M. Jones. "Timed hyperproperties". In: *Information and Computation* 280 (2021), p. 104639. DOI: 10.1016/J.IC.2020.104639 (cit. on p. 14).

[Inc17] Wolfram Research, Inc. *Mathematica, Version 11.2*. Champaign, IL, 2017. 2017 (cit. on p. 21).

[JANI17] *JANI specification*. 2017. URL: https://jani-spec.org/ (cit. on p. 43).

[JKNP17] Aleksandra Jovanovic, Marta Kwiatkowska, Gethin Norman, and Quentin Peyras. "Symbolic optimal expected time reachability computation and controller synthesis for probabilistic timed automata". In: *Theoretical Computer Science* 669 (2017), pp. 1–21. DOI: 10.1016/j.tcs.2017.01.015 (cit. on p. 21).

[JLBLP15] Jacques-Henri Jourdan, Vincent Laporte, Sandrine Blazy, Xavier Leroy, and David Pichardie. "A Formally-Verified C Static Analyzer". In: *POPL 2015* (Jan. 15–17, 2015). Ed. by Sriram K. Rajamani and David Walker. Mumbai, India: ACM, 2015, pp. 247–259. DOI: 10.1145/2676726.2676966 (cit. on p. 14).

[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 pp. 17, 34, 36, 37, 86, 87, 89, 94, 125, 141).

[JLR19] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. "A game approach to the parametric control of real-time systems". In: *International Journal of Control* 92.9 (2019), pp. 2025–2036. DOI: 10.1080/00207179.2018.1426883 (cit. on pp. 21, 141).

[JLR22] Aleksandra Jovanovic, Didier Lime, and Olivier H. Roux. "Control of Real-Time Systems With Integer Parameters". In: *IEEE Transactions on Automatic Control* 67.1 (2022), pp. 75–88. DOI: 10.1109/tac.2020.3046578 (cit. on p. 22).

[Jov13] Aleksandra Jovanović. "Parametric verification of timed systems." PhD thesis. École Centrale de Nantes, 2013 (cit. on p. 17).

[JRLD07] Jan Jakob Jessen, Jacob Illum Rasmussen, Kim Guldstrand Larsen, and Alexandre David. "Guided Controller Synthesis for Climate Controller Using Uppaal Tiga". In: *FORMATS 2007* (Oct. 3–5, 2007). Ed. by Jean-François Raskin and P. S. Thiagarajan. Vol. 4763. Lecture Notes in Computer Science. Salzburg, Austria: Springer, 2007, pp. 227–240. DOI: 10.1007/978-3-540-75454-1_17 (cit. on p. 21).

[JWL18] Yiding Ji, Yi-Chin Wu, and Stéphane Lafortune. "Enforcement of opacity by public and private insertion functions". In: *Automatica* 93 (2018), pp. 369–378. DOI: 10.1016/j.automatica.2018.03.041 (cit. on p. 14).

[Kel76] Robert M. Keller. "Formal Verification of Parallel Programs". In: *Communications of the ACM* 19.7 (1976), pp. 371–384. DOI: 10.1145/360248.360251 (cit. on pp. 7, 26).

[KLS20] Nikolai Kosmatov, Delphine Longuet, and Romain Soulat. "Formal Verification of an Industrial Distributed Algorithm: An Experience Report". In: *ISoLA 2020* (Oct. 20–30, 2020). Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 12476. Lecture Notes in Computer Science. Rhodes, Greece: Springer, 2020, pp. 525–542. DOI: 10.1007/978-3-030-61362-4_30 (cit. on p. 18).

[KNP12] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. "The PRISM Benchmark Suite". In: *QEST* (Sept. 17–20, 2012). London, United Kingdom: IEEE Computer Society, 2012, pp. 203–204. DOI: 10.1109/QEST.2012.14 (cit. on p. 23).

[Koc+20] 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. "Spectre attacks: exploiting speculative execution". In: *Communications of the*

ACM 63.7 (2020), pp. 93–101. DOI: 10.1145/3399742 (cit. on p. 4).

[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. 19).

[KP12] Michał Knapik and Wojciech Penczek. "Bounded Model Checking for Parametric Timed Automata". In: *Transactions on Petri Nets and Other Models of Concurrency*. Lecture Notes in Computer Science 5 (2012). Ed. by Kurt Jensen, Susanna Donatelli, and Jetty Kleijn, pp. 141–159. DOI: 10.1007/978-3-642-29072-5_6 (cit. on p. 41).

[KPJJ13] Robert Kotcher, Yutong Pei, Pranjal Jumde, and Collin Jackson. "Cross-origin pixel stealing: timing attacks using CSS filters". In: *CCS* (Nov. 4–8, 2013). Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. Berlin, Germany: ACM, 2013, pp. 1055–1062. DOI: 10.1145/2508859.2516712 (cit. on p. 19).

[KSA22] Aqsa Kashaf, Vyas Sekar, and Yuvraj Agarwal. "Protecting Smart Homes from Unintended Application Actions". In: *ICCPS* (May 4–Apr. 6, 2022). Milano, Italy: IEEE, 2022, pp. 270–281. DOI: 10.1109/ICCPS54341.2022.00031 (cit. on p. 18).

[Ler09] Xavier Leroy. "Formal Verification of a Realistic Compiler". In: *Communications of the ACM* 52.7 (2009), 107–115. ISSN: 0001-0782. DOI: 10.1145/1538788.1538814 (cit. on p. 14).

[LGSBL19] Lars Luthmann, Timo Gerecht, Andreas Stephan, Johannes Bürdek, and Malte Lochau. "Minimum/maximum delay testing of product lines with unbounded parametric real-time constraints". In: *Journal of Systems and Software* 149 (2019), pp. 535–553. DOI: 10.1016/j.jss.2018.12.028 (cit. on pp. 18, 41).

[LKP19] Giovanni Liva, Muhammad Taimoor Khan, and Martin Pinzger. "Semantics-driven extraction of timed automata from Java programs". In: *Empirical Software Engineering* 24.5 (2019), pp. 3114–3150. DOI: 10.1007/s10664-019-09699-5 (cit. on pp. 16, 148).

[LLR23] Loriane Leclercq, Didier Lime, and Olivier H. Roux. "A State Class Based Controller Synthesis Approach for Time Petri Nets". In: *PETRI NETS 2023* (June 25–30, 2023). Ed. by Luís Gomes and Robert Lorenz. Vol. 13929. Lecture Notes in Computer Science. Lisbon, Portugal: Springer, 2023, pp. 393–414. DOI: 10.1007/978-3-031-33620-1_21 (cit. on p. 21).

[LMNS05] Kim Guldstrand Larsen, Marius Mikucionis, Brian Nielsen, and Arne Skou. “Testing real-time embedded software using UPPAAL-TRON: an industrial case study”. In: *EMSOFT 2005* (Sept. 18–22, 2005). Ed. by Wayne H. Wolf. Jersey City, NJ, USA: ACM, 2005, pp. 299–306. DOI: 10.1145/1086228.1086283 (cit. on p. 16).

[LMS04] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. “Model Checking Timed Automata with One or Two Clocks”. In: *CONCUR* (Aug. 31–Sept. 3, 2004). Ed. by Philippa Gardner and Nobuko Yoshida. Vol. 3170. Lecture Notes in Computer Science. London, UK: Springer, 2004, pp. 387–401. DOI: 10.1007/978-3-540-28644-8_25 (cit. on p. 14).

[LODLDP13] Alfons Laarman, Mads Chr. Olesen, Andreas Engelbredt Dalsgaard, Kim Guldstrand Larsen, and Jaco van De Pol. “Multi-Core Emptiness Checking of Timed Büchi Automata using Inclusion Abstraction”. In: *CAV* (July 13–19, 2013). Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Saint Petersburg, Russia: Springer, July 2013, pp. 968–983. DOI: 10.1007/978-3-642-39799-8_69 (cit. on pp. 22, 54, 58).

[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 pp. 6, 15, 20, 46).

[LRS21] Didier Lime, Olivier H. Roux, and Charlotte Seidner. “Cost Problems for Parametric Time Petri Nets”. In: *Fundamenta Informormaticae* 183.1-2 (2021), pp. 97–123. DOI: 10.3233/FI-2021-2083 (cit. on p. 147).

[LRST09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. “Romeo: A Parametric Model-Checker for Petri Nets with Stopwatches”. In: *TACAS* (Mar. 22–29, 2009). Ed. by Stefan Kowalewski and Anna Philippou. Vol. 5505. Lecture Notes in Computer Science. York, UK: Springer, 2009, pp. 54–57. DOI: 10.1007/978-3-642-00768-2_6 (cit. on p. 17).

[LSBL17] Lars Luthmann, Andreas Stephan, Johannes Bürdek, and Malte Lochau. “Modeling and Testing Product Lines with Unbounded Parametric Real-Time Constraints”. In: *SPLC, Volume A* (Sept. 25–29, 2017). Ed. by Myra B. Cohen, Mathieu Acher, Lidia Fuentes, Daniel Schall, Jan Bosch, Rafael Capilla, Ebrahim Bagheri, Yingfei Xiong, Javier Troya, Antonio Ruiz Cortés, and David Benavides. Sevilla, Spain: ACM, 2017, pp. 104–113. DOI: 10.1145/3106195.3106204 (cit. on p. 41).

[LYGY10] Mingsong Lv, Wang Yi, Nan Guan, and Ge Yu. "Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software". In: *RTSS* (Nov. 30–Dec. 3, 2010). San Diego, California, USA: IEEE Computer Society, 2010, pp. 339–349. ISBN: 978-0-7695-4298-0. DOI: 10.1109/RTSS.2010.30 (cit. on p. 101).

[MARS21] *MARS repository*. 2021. URL: http://www.mars-workshop.org/repository.html (cit. on p. 24).

[MCC21] *Model Checking Contest*. 2021. URL: https://mcc.lip6.fr/ (cit. on p. 24).

[Mil00] Joseph S. Miller. "Decidability and Complexity Results for Timed Automata and Semi-linear Hybrid Automata". In: *HSCC* (Mar. 23–25, 2000). Ed. by Nancy A. Lynch and Bruce H. Krogh. Vol. 1790. Lecture Notes in Computer Science. Pittsburgh, PA, USA: Springer, 2000, pp. 296–309. DOI: 10.1007/3-540-46430-1_26 (cit. on pp. 16, 27, 86, 87).

[Min67] Marvin L. Minsky. *Computation: Finite and infinite machines*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1967. ISBN: 0-13-165563-9 (cit. on p. 86).

[Mon10] David Monniaux. "Quantifier Elimination by Lazy Model Enumeration". In: *CAV 2010* (July 15–19, 2010). Ed. by Tayssir Touili, Byron Cook, and Paul B. Jackson. Vol. 6174. Lecture Notes in Computer Science. Edinburgh, UK: Springer, 2010, pp. 585–599. DOI: 10.1007/978-3-642-14295-6_51 (cit. on p. 21).

[MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis. "On the Synthesis of Discrete Controllers for Timed Systems". In: *STACS 95* (Mar. 2–4, 1995). Ed. by Ernst W. Mayr and Claude Puech. Vol. 900. Lecture Notes in Computer Science. Munich, Germany: Springer, 1995, pp. 229–242. DOI: 10.1007/3-540-59042-0_76 (cit. on p. 21).

[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. 20).

[NPP18] Hoang Gia Nguyen, Laure Petrucci, and Jaco van de Pol. "Layered and Collecting NDFS with Subsumption for Parametric Timed Automata". In: *ICECCS* (Dec. 12–14, 2018). Ed. by Anthony Widjaja Lin, Jun Sun, and

Anthony Widjaja Lin. Melbourne, Australia: IEEE Computer Society, Dec. 2018, pp. 1–9. DOI: 10.1109/ICECCS2018.2018.00009 (cit. on pp. 41, 48, 49, 56).

[NSJMM22] Reiya Noguchi, Ocan Sankur, Thierry Jéron, Nicolas Markey, and David Mentré. "Repairing Real-Time Requirements". In: *Proceedings of the 20th International Symposium on Automated Technology for Verification and Analysis, ATVA 2022* (Oct. 25–28, 2022). Ed. by Ahmed Bouajjani, Lukás Holík, and Zhilin Wu. Vol. 13505. Lecture Notes in Computer Science. Virtual Event: Springer, 2022, pp. 371–387. DOI: 10.1007/978-3-031-19992-9_24 (cit. on p. 16).

[OW10] Joël Ouaknine and James Worrell. "Towards a Theory of Time-Bounded Verification". In: *ICALP* (July 7–10, 2010). Ed. by Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis. Lecture Notes in Computer Science. Bordeaux, France: Springer, 2010, pp. 22–37. DOI: 10.1007/978-3-642-14162-1_3 (cit. on p. 17).

[Pet62] Carl Adam Petri. "Kommunikation mit Automaten". PhD thesis. Darmstadt University of Technology, Germany, 1962 (cit. on p. 5).

[PKPS19] Laure Petrucci, Michał Knapik, Wojciech Penczek, and Teofil Sidoruk. "Squeezing State Spaces of (Attack-Defence) Trees". In: *ICECCS* (Nov. 10–13, 2019). Ed. by Jun Pang and Jing Sun. Guangzhou, China: IEEE, 2019, pp. 71–80. DOI: 10.1109/ICECCS.2019.00015 (cit. on p. 18).

[Pnu77] Amir Pnueli. "The Temporal Logic of Programs". In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science* (Oct. 31–Nov. 1, 1977). Providence, Rhode Island, USA: IEEE Computer Society, 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32 (cit. on pp. 58, 181).

[QV15] Sophie Quinton and Tullio Vardanega. *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. 2015. URL: http://waters2015.inria.fr/ (cit. on p. 24).

[RFRJM19] Matthieu Renard, Yliès Falcone, Antoine Rollet, Thierry Jéron, and Hervé Marchand. "Optimal enforcement of (timed) properties with uncontrollable events". In: *Mathematical Structures in Computer Science* 29.1 (2019), pp. 169–214. DOI: 10.1017/S0960129517000123 (cit. on p. 16).

[Ros19] Amnon Rosenmann. "The Timestamp of Timed Automata". In: *FORMATS* (Aug. 27–29, 2019). Ed. by Étienne André and Mariëlle Stoelinga. Vol. 11750. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, 2019, pp. 181–198. DOI: 10.1007/978-3-030-29662-9_11 (cit. on p. 74).

[SAL15] Youcheng Sun, Étienne André, and Giuseppe Lipari. "Verification of Two Real-Time Systems Using Parametric Timed Automata". In: *WATERS* (July 7, 2015). Ed. by Sophie Quinton and Tullio Vardanega. Lund, Sweden, July 2015 (cit. on p. 18).

[SBMR13] Ocan Sankur, Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. "Robust Controller Synthesis in Timed Automata". In: *CONCUR 2013* (Aug. 27–30, 2013). Ed. by Pedro R. D'Argenio and Hernán C. Melgratti. Vol. 8052. Lecture Notes in Computer Science. Buenos Aires, Argentina: Springer, 2013, pp. 546–560. DOI: 10.1007/978-3-642-40184-8_38 (cit. on p. 21).

[SBP19] Nathanaël Sensfelder, Julien Brunel, and Claire Pagetti. "Modeling Cache Coherence to Expose Interference". In: *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Ed. by Sophie Quinton. Vol. 133. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 18:1–18:22. ISBN: 978-3-95977-110-8. DOI: 10.4230/LIPIcs.ECRTS.2019.18 (cit. on p. 16).

[Sch99] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999. ISBN: 978-0-471-98232-6 (cit. on p. 34).

[SGQ16] Lijun Shan, Susanne Graf, and Sophie Quinton. *RTLib: A Library of Timed Automata for Modeling Real-Time Systems*. Research Report. Grenoble 1 UGA - Université Grenoble Alpe; INRIA Grenoble - Rhone-Alpes, Nov. 2016 (cit. on p. 23).

[SLDP09] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. "PAT: Towards Flexible Verification under Fairness". In: Lecture Notes in Computer Science 5643 (2009). Ed. by Ahmed Bouajjani and Oded Maler, pp. 709–714. DOI: 10.1007/978-3-642-02658-4_59 (cit. on pp. 6, 15).

[SM73] Larry J. Stockmeyer and Albert R. Meyer. "Word Problems Requiring Exponential Time: Preliminary Report". In: *Proceedings of the 5th Annual ACM Symposium on Theory of Computing* (Apr. 30–May 2, 1973). Ed. by Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong. Austin, Texas, USA: ACM, 1973, pp. 1–9. DOI: 10.1145/800125.804029 (cit. on p. 119).

[SNÁ17] Stefan Schupp, Johanna Nellen, and Erika Ábrahám. "Divide and Conquer: Variable Set Separation in Hybrid Systems Reachability Analysis". In: *QAPL@ETAPS*. Ed. by Herbert Wiklicky and Erik P. de Vink. Vol. 250. Electronic Proceedings in Theoretical Computer Science. Uppsala, Sweden, 2017, pp. 1–14. DOI: 10.4204/EPTCS.250.1 (cit. on p. 22).

[Tri02] Stavros Tripakis. "Fault Diagnosis for Timed Automata". In: *FTRTFT* (Sept. 9–12, 2002). Ed. by Werner Damm and Ernst-Rüdiger Olderog. Vol. 2469. Lecture Notes in Computer Science. Oldenburg, Germany: Springer, 2002, pp. 205–224. DOI: 10.1007/3-540-45739-9_14 (cit. on p. 19).

[Tri06] Stavros Tripakis. "Folk theorems on the determinization and minimization of timed automata". In: *Information Processing Letters* 99.6 (2006), pp. 222–226. DOI: 10.1016/j.ipl.2006.04.015 (cit. on p. 15).

[Tri98] Stavros Tripakis. "L'analyse formelle des systèmes temporisés en pratique. (The Formal Analysis of Timed Systems in Practice)". PhD thesis. Joseph Fourier University, Grenoble, France, 1998 (cit. on p. 97).

[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 pp. 20, 103, 104, 107).

[VNNK19] Panagiotis Vasilikos, Hanne Riis Nielson, Flemming Nielson, and Boris Köpf. "Timing Leaks and Coarse-Grained Clocks". In: *CSF 2019* (June 25–28, 2019). Hoboken, NJ, USA: IEEE, 2019, pp. 32–47. DOI: 10.1109/CSF.2019.00010 (cit. on p. 16).

[Wei99] Volker Weispfenning. "Mixed Real-Integer Linear Quantifier Elimination". In: *ISSAC* (July 29–31, 1999). Ed. by Keith O. Geddes, Bruno Salvy, and Samuel S. Dooley. Vancouver, BC, Canada: Association for Computing Machinery, 1999, pp. 129–136. ISBN: 1581130732. DOI: 10.1145/309831.309888 (cit. on pp. 74, 84).

[WGSW18] Meng Wu, Shengjian Guo, Patrick Schaumont, and Chao Wang. "Eliminating Timing Side-Channel Leaks Using Program Repair". In: *ISSTA* (July 16–21, 2018). Ed. by Frank Tip and Eric Bodden. Amsterdam, Netherlands: ACM, 2018, pp. 15–26. DOI: 10.1145/3213846.3213851 (cit. on p. 14).

[WZ18] Lingtai Wang and Naijun Zhan. “Decidability of the Initial-State Opacity of Real-Time Automata”. In: *Symposium on Real-Time and Hybrid Systems*. Ed. by Cliff B. Jones, Ji Wang, and Naijun Zhan. Vol. 11180. Lecture Notes in Computer Science. Springer, 2018, pp. 44–60. DOI: 10.1007/978-3-030-01461-2_3 (cit. on pp. 8, 77, 78, 114).

[WZA18] Lingtai Wang, Naijun Zhan, and Jie An. “The Opacity of Real-Time Automata”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2845–2856. DOI: 10.1109/TCAD.2018.2857363 (cit. on pp. 8, 77, 78).

[Yov97] Sergio Yovine. “KRONOS: A Verification Tool for Real-Time Systems”. In: *International Journal on Software Tools for Technology Transfer* 1.1-2 (1997), pp. 123–133. DOI: 10.1007/s100090050009 (cit. on p. 15).

APPENDICES

A CLASSICAL NOTATIONS

In this appendix, we formally define the logic notions and notations used in the Section 5.1.1.

A.1 *Linear temporal logic*

We recall below the syntax and semantics of the linear temporal logic (LTL) [Pnu77].

Syntax

Given a set AP of atomic propositions, LTL formulae over the set AP are formed according to the following grammar:

$$\varphi \quad ::= \quad \text{true} \quad | \quad a \quad | \quad \varphi_1 \wedge \varphi_2 \quad | \quad \neg\varphi \quad | \quad \bigcirc\, \varphi \quad | \quad \varphi_1 \cup \varphi_2$$

with $a \in AP$.

The $\bigcirc$ modality, pronounced "next", is a unary operator where $\bigcirc\varphi$ holds (at the current moment) if φ holds in the next state. The $\cup$ modality is called "until". It is a binary operator where $\varphi_1 \cup \varphi_2$ holds at the current moment if there is some future moment for which φ_2 holds and φ_1 holds at all moment until this future moment.

The classical boolean connectives, such as the disjunction $\vee$, the implication $\rightarrow$ and the equivalence $\leftrightarrow$, can be derived from this grammar and can therefore be used in LTL formulae.

From the $\cup$ operator, it is possible to derive to the temporal modalities $\diamond$ (named "eventually", sometimes in the future) and $\square$ (named "always", from now on forever).

Semantics

An LTL formula can be satisfied by valuations of the variables in AP. The satisfaction relation is defined in the following, where $w[i : \dots] = w_i w_{i+1} \dots$ denotes the suffix of w starting with the symbol in the i position.

Given $w = w_0 w_1 w_2 \dots \in (2^{AP})^\omega$, the satisfaction relation $\models$ is defined as the smallest relation such that:

- $w \models \text{true}$

- $w \models a$ *iff* $a \in w_0$
- $w \models \varphi_1 \wedge \varphi_2$ *iff* $w \models \varphi_1$ and $w \models \varphi_2$
- $w \models \neg\varphi$ *iff* $w \not\models \varphi$
- $w \models \bigcirc\varphi$ *iff* $w[1:\ldots] \models \varphi$
- $w \models \varphi_1 \cup \varphi_2$ *iff* $\exists j \geq 0, w[j:\ldots] \models \varphi_2$ and $\forall i, 0 \leq i < j, w[i:\ldots] \models \varphi_1$.

A.2 *Computation tree logic*

We recall below the syntax and semantics of the computation tree logic (CTL) [CE81].

Syntax

CTL state formulae over the set AP of atomic propositions are formed according to the following grammar, with $a \in AP$ and φ a path formula:

$$\phi \quad ::= \quad \text{true} \quad | \quad a \quad | \quad \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \quad \exists\varphi \quad | \quad \forall\varphi$$

CTL path formulae are formed according to the following grammar, where ϕ, ϕ_1, ϕ_2 are state formulae:

$$\varphi \quad ::= \quad \bigcirc\phi \quad | \quad \phi_1 \cup \phi_2$$

The path operators $\bigcirc$ and $\cup$ are defined with the same meaning than in LTL, $\exists$ and $\forall$ are quantifiers meaning respectively "for some path" (there exists a path where φ holds) and "for all paths" (for all paths, φ holds).

Semantics

CTL formulae are interpreted using LTSs. Given an LTS S, the semantics of a CTL formula is given by two relations (denoted $\models$), one on the states of S (for state formulae) and one on run of S (for path formulae).

Let $a \in AP$ be an atomic proposition, $S = (Q_S, \Sigma_S, \rightarrow_S, q_{0,S})$ be an LTS corresponding to the concrete semantics of a PTA, a state $q_S \in Q_S$, ϕ, ϕ_1, ϕ_2 be CTL state formulae, and φ be a CTL path formula. Since the LTS refers to the semantics of a PTA, we consider as labeling function L the function which assigns to a concrete state of S the corresponding location of the PTA. Formally, given $q_S = \mathfrak{s} = (\ell, \mathbf{C})$, $L(q_S) = \ell$.

We denote by $Runs(q_S)$ the set of runs starting from state q_S in the LTS.

For state formulae, the relation $\models$ is defined as follows:

- $q_S \models a$ *iff* $a = L(q_S)$
- $q_S \models \neg\phi$ *iff* $q_S \not\models \phi$
- $q_S \models \phi_1 \wedge \phi_2$ *iff* $q_S \models \phi_1 \wedge q_S \models \phi_2$
- $q_S \models \exists\varphi$ *iff* $R \models \varphi$ for some run $R \in Runs(q_S)$
- $q_S \models \forall\varphi$ *iff* $R \models \varphi$ for all runs $R \in Runs(q_S)$.

For path formulae, the relation $\models$ is defined as follows, with a run R of the LTS and where $q_S[i]$ refers to the (i+1)th state of R:

- $R \models \bigcirc\phi$ *iff* $R[1] \models \phi$
- $R \models \phi_1 \cup \phi_2$ *iff* $\exists j \geq 0, q_S[j] \models \phi_2$ and $\forall i, 0 \leq i < j, q_S[i] \models \phi_1$

A.3 *Universal fragment of CTL∗*

We now introduce formulae in the universal fragment of CTL∗ (∀CTL*) [BK08] and their semantics.

Syntax

Formulae in ∀CTL* are given by the following grammar, with $a \in AP$, the state formula ϕ and path formula φ:

$$\phi \quad ::= \quad \text{true} \quad | \quad \text{false} \quad | \quad a \quad | \quad \neg a \quad | \quad \phi_1 \wedge \phi_2 \quad | \quad \phi_1 \vee \phi_2 \quad | \quad \forall\varphi$$

$$\varphi \quad ::= \quad \phi \quad | \quad \bigcirc\,\varphi \quad | \quad \varphi_1 \wedge \varphi 2 \quad | \quad \varphi_1 \vee \varphi 2 \quad | \quad \varphi_1 \cup \varphi_2 \quad |$$

Note that, in [BK08], ∀CTL* is defined with a release operator, denoted R and defined by $\exists(\phi_1 \,\mathsf{R}\, \phi_2) = \neg\forall((\neg\phi_1) \cup (\neg\phi_2))$ and $\forall(\phi_1 \,\mathsf{R}\, \phi_2) = \neg\exists((\neg\phi_1) \cup (\neg\phi_2))$.

B CORRESPONDENCE WITH THE NOTIONS DEFINED IN PREVIOUS PAPERS

In Table B.4, we present the correspondence between the notions introduced in this thesis and their original designation.

The changes are due to:

- a clarification of the "opacity" we consider, based only on the execution times;
- the inclusion of ℓ_{priv} and ℓ_f in the definition of a TA;
- a clarification on the problem namings.

Table B.4: Correspondence with the notions defined in [ALMS22]

Notion defined in this thesis	**Corresponding notion defined in the paper**
ET-opacity	(timed) opacity
ET-opaque for a set of execution times	(timed) opaque w.r.t. ℓ_{priv} on the way to ℓ_f for a set of execution times
∃-ET-opaque	(timed) opaque w.r.t. ℓ_{priv} on the way to ℓ_f for a non-empty set of execution times
$DVisit^{priv}(\mathcal{A})$	$DReach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$
$DVisit^{\neg priv}(\mathcal{A})$	$DReach^{\mathcal{A}}_{\neg\ell_{priv}}(\ell_f)$
∃-ET-opacity decision problem	
full ET-opacity decision problem	Full timed opacity decision problem
ET-opacity t-computation problem	Timed opacity computation problem
∃-ET-opacity p-synthesis problem	Timed opacity synthesis problem
full ET-opacity p-synthesis problem	Full timed opacity synthesis problem
∃-ET-opacity p-emptiness problem	Timed opacity emptiness problem
full ET-opacity p-emptiness problem	Full timed opacity emptiness problem

C 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

i) this is the way the program is presented in the DARPA library, and

ii) a (minor) difficulty may come from these loops instead of a simple Thread.sleep(32*32) statement.

NOMENCLATURE

Clocks, parameters and constraints

x	A clock
$\vec{0}$	Clock valuation assigning 0 to all clocks
μ	Clock valuation
p	A parameter
v	Parameter valuation

Mathematical notations

$\lvert\bullet\rvert$	Cardinality of a set
$\dot{\bullet}$	Derivative
$\mathrm{fr}(\bullet)$	Fractionnal part
$\lfloor\bullet\rfloor$	Integral part
$\bullet^{\omega}$	Repetition of zero or more occurrences in a word
$\mathbb{N}$	Set of integers
$\mathbb{N}^{\infty}$	$\mathbb{N} \cup \{+\infty\}$
$\mathbb{Q}_{\geq 0}$	Set of non-negative rationals
$\mathbb{R}_{\geq 0}$	Set of non-negative reals
$\mathbb{R}_{\geq 0}^{\infty}$	$\mathbb{R}_{\geq 0} \cup \{+\infty\}$
$\mathbb{Z}$	Set of (positive and negative) integers

Execution-time opacity

$\mathit{DVisit}^{\mathit{priv}}(\bullet)$ Set of the durations of private runs of $\bullet$

$\mathit{Visit}^{\mathit{priv}}(\bullet)$ Set of private runs of $\bullet$

$\mathit{DVisit}^{\neg\mathit{priv}}(\bullet)$ Set of the durations of public runs of $\bullet$

$\mathit{Visit}^{\neg\mathit{priv}}(\bullet)$ Set of public runs of $\bullet$

$\mathit{DVisit}^{\mathit{priv}}_{\leq\square}(\bullet)$ Set of the durations of secret runs of $\bullet$

$\mathit{Visit}^{\mathit{priv}}_{\leq\square}(\bullet)$ Set of secret runs of $\bullet$

$\mathit{DVisit}^{\mathit{priv}}_{>\square}(\bullet)$ Set of the durations of private and non-secret runs of $\bullet$

$\mathit{Visit}^{\mathit{priv}}_{>\square}(\bullet)$ Set of private and non-secret runs of $\bullet$

Parametric Timed Automata

a	An action
Σ	Set of actions
$\mathbf{C}$	A constraint (over $\mathbb{X}$ or $\mathbb{X} \cup \mathbb{P}$)
e	An edge
E	Set of edges
D	Set of execution times
Δ	Expiring bound considered in Chapter 8 on page 111
ℓ	A location
L	Set of locations
$\mathcal{P}_{\mathcal{LU}}$	Lower/upper parametric timed automaton
$\mathcal{P}$	Parametric timed automaton
$\mathbf{PZG}(\bullet)$	PZG of $\bullet$
r	A region
$\mathcal{RA}_\bullet$	Region automaton of $\bullet$
$\mathcal{RG}_\bullet$	Region graph of $\bullet$
$\mathcal{R}_\bullet$	Set of regions
$dur(\bullet)$	Duration of a run
$\mathfrak{T}_\bullet$	Semantics of $\bullet$ (with a TTS)
$\mathcal{D}$	Set of expiring bounds
$dur_\bullet(\bullet, \bullet)$	Duration between two states (or locations) in a run
σ	A (static, untimed) strategy
$\mathbf{s}$	A symbolic state
$\mathbf{S}$	Set of symbolic states
$\mathcal{A}$	Timed automaton