Contributions to timed systems and transducers

Habilitation Thesis

Mémoire d'Habilitation à Diriger des Recherches

Pierre-Alain Reynier

Laboratoire d'informatique fondamentale de Marseille
Aix-Marseille Université & CNRS, UMR 7279

November 2015

The committee was composed of:

Rajeev Alur	University of Pennsylvania	Reviewer
François Denis	Aix-Marseille Université	
Anca Muscholl	Université de Bordeaux	Reviewer
Jean-François Raskin	Université Libre de Bruxelles	
Philippe Schnoebelen	CNRS & ENS Cachan	
Jean-Marc Talbot	Aix-Marseille Université	
Sophie Tison	Université de Lille 1	
James Worrell	Oxford University	Reviewer

The document is mostly based on joint works with:

S. Akshay	Patricia Bouyer
Peter Bulychev	Mathieu Caralp
Franck Cassez	Luc Dartois
Laure Daviaud	Alexandre David
Emmanuel Filiot	Olivier Gauwin
Loïc Hélouët	Claude Jard
Rémi Jaubert	Jan J. Jessen
Kim G. Larsen	Janusz Malinowski
Sebastian Maneth	Nicolas Markey
Omer Nguena-Timo	Peter Niebert
Youssouf Oualhadj	Jean-François Raskin
Arnaud Sangnier	Ocan Sankur
Frédéric Servais	Jean-Marc Talbot

Abstract. Algorithmic formal verification has proved to be a promising technology to improve reliability and quality of software systems. These works are rooted in the theory of regular languages, that enjoy important automata-logic connections. In order to address complex modern software systems, there is a need for developing new theoretical models, and for improving our understanding of the decidability and complexity of decision problems. This relies in particular on the development of automata-logic connections for these new models. In this habilitation thesis, I present some of my recent contributions in the areas of timed systems and transducers, that allow to faithfully describe embedded real-time systems and programs manipulating data streams, respectively. More precisely, the first part of this document addresses robustness issues in timed systems, and contains results on model checking and controller synthesis problems in presence of timing perturbations. The second part, devoted to transducers, studies fundamental problems for different classes of word and nested word transformations, with applications to streamability problems.

Contents

Introduction

This habilitation thesis summarizes a selection of my research results obtained since my PhD thesis, defended in June 2007.

The growing importance of computer systems

Computer systems are now present in every aspect of our lives. They are present in personal equipments (laptops, smartphones, embedded systems in cars...) as well as in industrial ones (airplanes, plants...), and may in both cases operate as critical systems. With this widespread development, these systems are becoming more and more complex, and it becomes harder and harder to guarantee their reliability. In parallel, with the development of the Web, we have also observed the multiplication of communicating applications exchanging huge volumes of data, requiring the conception of highly scalable applications. Developing software which is both reliable and able to manipulate large inputs with a low memory consumption is a major challenge.

The objective of the works I present in this habilitation thesis is to contribute to the development of fundamental tools to achieve these issues of reliability and efficiency of software. The common loam is the use of automata based models. As I will explain, they are at the heart of the model checking approach, which is a successful formal method for ensuring reliability. At the same time, they are intensively used in database management, allowing to design simple and robust operational models.

Automata based approaches to reliability and efficiency

Software-engineering techniques aim to increase the quality of software through a rigorous development process. This is achieved for instance by systematic testing, which helps discovering bugs but does not guarantee their absence. With a different point of view, *formal methods* aim at relying on mathematical tools to provide rigorous foundations to software development. In the context of software verification, the objective is to come up with a proof of correctness of the software at the end of the development process.

Different formal approaches have been studied, and we will be interested in the first part of this thesis in model checking, an *automata based approach* to software analysis. With a formal specification of the desired behavior of a system in one hand (given for instance as some logical formula), and a model of the system in the other hand (given for instance as some automaton), the model checking approach aims at verifying that the actual behavior of the system corresponds to the specification.

Model checking was initially developed on finite state systems, checked against temporal logics. Amir Pnueli was awarded the Turing prize in 1996 for introducing temporal logics in

computing science, while Edmund M. Clarke, E. Allen Emerson and Joseph Sifakis received this prize in 2007 for making model checking a highly effective verification technology. Since its beginning, model checking has been extended to numerous classes of infinite state systems, including pushdown systems, Petri nets, counter automata, timed automata or hybrid automata. Application domains of model checking are numerous, ranging from industrial applications in circuit design to databases, going through embedded systems and security issues.

Using automata based formal approaches for software verification and design allows to increase their reliability. In addition, automata based modelization can also be fruitful for several other aspects of software development. For instance, timed automata models have been used to model real-time software systems, and to derive sharp evaluation of their worst-case execution times. Similarly, performance evaluation can be realized using Markov decision processes. A last example comes from the context of database-driven systems in which efficient query evaluation schemes have been derived using these techniques.

Verification and synthesis of embedded systems

Embedded systems are highly critical, and their design is very challenging as they often combine real-time computing constraints with a distributed architecture with complex communications. This motivated the development of formal models to describe, analyze and synthesize these systems.

Regarding the real-time constraints, the model of timed automata, introduced at the beginning of the 1990's by Rajeev Alur and David Dill, is now widely accepted as the reference. Timed automata are defined as the extension of finite-state automata with finitely many real-valued variables, called clocks. The languages they recognize are sets of of timed words, which extend well-known words over a finite alphabet by associating to each action a time-stamp recording the absolute time of occurrence of this action.

The model of Petri nets has been introduced by Carl Adam Petri in his PhD thesis in 1962. It constitutes one of the main models to describe distributed and concurrent systems, and has been used in numerous applications, from verification to performance evaluation. This models fits into the class of infinite-state systems as the number of possible configurations is infinite. This makes several interesting problems undecidable, and requires the development of new techniques to make decidable problems tractable.

Studying the theoretical properties of these models by identifying the frontiers between decidability and undecidability, as well as developing new efficient model checking algorithms, is the core of my research work in this domain. I also considered models which combine real-time constraints with concurrency, such as time extensions of Petri nets. Different such extensions have been considered, by associating timing constraints either on places, arcs, or transitions. Depending on the chosen semantics, the decidability status of the most basic problem, namely that of reachability of an (untimed) marking, may change.

The design of embedded systems is a highly non-trivial task, as one has to cope with timing constraints emerging from different components. The automatic synthesis offers the possibility to handle complex timing constraints expressed using networks of timed automata.

Controller synthesis extends model checking as follows: given the model of a system in which some actions are classified into *controllable* and *uncontrollable* ones, and a logic formula representing the specification, the objective is to automatically build a strategy for a controller of the system that achieves the specification. Relying on the formal tool of two-player games,

this framework has been intensively developed during the last decade. In the context of timed systems, efficient algorithms have been developed and successfully implemented in tools such as UppAal-TiGa.

Though I contributed to the different aspects presented before, I decided to focus the first part of this thesis to my contributions related to the robustness analysis of timed systems. Indeed, the semantics of timed automata relies on several mathematical idealizations that are not satisfied by real hardware and as a consequence, the implementability of timed automata is still a difficult task. It was shown that these implementability issues are related to a notion of robustness of timed automata, which can be understood as their tolerance to the introduction of small perturbations in timing measurements. More precisely, Anuj Puri introduced a parameterized semantics of timed automata which relaxes the timing constraints by some parameter δ. It was later shown by Jean-François Raskin and his co-authors that this δ can be understood as the precision of the implementation platform: if the timed automaton enlarged by $\delta > 0$ satisfies some linear-time property, then for some sufficiently fast and precise hardware (with constraints depending on δ), there exists an implementation of this timed automaton that also satisfies the property. As a consequence, this motivated the development of a robustness theory for timed automata, which aims at studying the parametric semantics of timed automata by considering an existential quantification over δ. While parametric timed automata are quickly undecidable, this parametric semantics of timed automata can be analyzed, as we will see in the developments of this thesis.

Analysis of data manipulating programs

Basic databases operations such as updates, views, or requests, are performed routinely by management systems or Web services. Data is central in such applications, and usually comes with an infinite domain, ensuring integrity of database-driven systems thus requires the development of new formal models. Automata on words and trees augmented with data coming from an infinite alphabet (such as register automata working on data words) show how comparison of variables with values in an infinite domain can be handled in the framework of automata theory.

Several operations on databases can also be modeled as functions (or relations) from the database (modeled as a string, or as a tree) to some quantitative domain such as integers, or even strings or trees. Indeed, one often queries a database in order to know *how many* entries satisfy a given property. Updating a tree-structured database by adding a new entry, or removing one, is naturally modeled as a tree-to-tree transformation. Transductions appear thus as a natural formal tool to analyse these systems. Different formalisms exist to model them, including grammars and logics. In this thesis, we will mainly be interested in the extension of automata with outputs, known as transducers.

In the beginning of the 2000's, the XML data model has become widespread, together with the need of adequate automata models. Though XML documents are simply linearizations of tree structures, and different models of tree automata are well known, it is important to study automata models working directly on these linearizations in order to derive memory efficient evaluation algorithms. The model of visibly pushdown automata, introduced in 2004 by Rajeev Alur and Parthasarathy Madhusudan, is a restriction of pushdown automata well-suited to take as input linearizations of unranked trees, such as XML documents. It has been shown that visibly pushdown automata can be used to derive efficient tools for manipulating XML Document Type Definitions (DTDs). This motivated the identification

of a model of transducer based on visibly pushdown automata, allowing to describe complex XML transformations.

Biographical sketch since the PhD

My PhD thesis, defended in June 2007, adresses the verification of distributed timed systems, modeled as networks of timed automata or as timed extensions of Petri nets. I have been interested during my doctoral studies in algorithmic and decidability issues regarding these systems, and in robustness issues.

At the end of my PhD, I decided to develop my works on the robustness analysis of timed systems, and therefore I joined the group of Prof. Jean-François Raskin at the Université Libre de Bruxelles during the academic year 2007/2008. Our initial objective at that time was to design an algorithm to automatically synthesize robust controllers for timed games. We studied this problem during my post-doctoral stay, but did not succeed to solve it. We finally managed to come up with a solution to this problem last year (2014) with my colleagues Youssouf Oualhadj (my post doctoral student at that time) and Ocan Sankur. In the meantime, I worked on different problems related to robust model checking of timed systems.

In parallel to this activity on robustness, I started to work on a different topic when I have been hired as assistant professor in Marseille in september 2008. Indeed, when I moved there, I initiated a new collaboration between Marseille and the group of Prof. Jean-François Raskin in Bruxelles on the topic of transformations. The starting point was the definition of a new model of transducers, based on the model of visibly pushdown automata. One of the objectives was to obtain a decision procedure for determinization of these transducers. This is of particular interest in order to derive efficient evaluation algorithms. Though this problem was harder than expected (it is still open as I am writing this document), it was the beginning of a very fruitful collaboration on the analysis of transformations which is still very active.

Apart from the two topics of robustness of timed systems and transformations analysis, I continued to work on distributed systems (Petri nets for instance) and on other aspects of timed systems analysis. These results are not presented in this thesis, but they are briefly surveyed at the end of this introduction.

Content of this thesis

This thesis is organized in two independent parts, on the two topics of robustness of timed systems, and analysis of transformations respectively.

Part I gathers results I obtained on *robustness analysis of timed systems.*

Chapter 1 contains important definitions and results on model checking of timed automata, and on controller synthesis in timed games. The implementability of timed automata is also discussed and illustrated with a few examples.

In Chapter 2, I present my contributions related to robust model checking. Given a timed automaton $\mathcal{A}$ and some temporal property φ, this problem consists in deciding whether there exists $\delta > 0$ such that the δ-enlarged semantics of $\mathcal{A}$ satisfies φ. When studying timed systems,

it is natural to consider specifications expressing timing requirements. A restriction of the rich timed temporal logic MTL has been introduced in [BMOW07], together with an efficient model checking algorithm, based on completely different tools, namely channel machines. We present an algorithm for the robust model checking of this logic, showing that the analysis of the effect of perturbations can be encoded using channel machines, and using our previous results on LTL robust model checking. This is a joint work with Patricia Bouyer and Nicolas Markey, published in [4].

While robust model checking allows to decide whether a property is satisfied for *some* level of timing perturbations, it is relevant for practical applications to compute the largest admissible perturbations. We consider this problem for safety properties, and present an algorithm for flat timed automata (meaning that there are no nested cycles). To this end, we provide an algorithm for the symbolic computation of the parametric reachability set of a flat timed automaton under its enlarged semantics. This is joint work with my (now former) PhD student Rémi Jaubert published in [18].

A third contribution related to robust model checking concerns the model of time Petri nets. In this work, we show that when considering unbounded time Petri nets, several problems are undecidable. We also manage to identify subclasses for which robust model checking is decidable, thanks to existing results on timed automata. This is joint work with S. Akshay, Loïc Hélouët and Claude Jard, published in [1, 2].

Chapter 3 presents my recent results on robust controller synthesis in timed games. Standard timed games are two-player zero-sum games, where the two players are called Controller and Environment. The objective is to synthesize a strategy for Controller which prescribes delays and actions in order to control the timed automaton, in such a way that some property is satisfied. Environment resolves the non-determinism associated with actions. In such games, Controller has infinite precision, in the sense that it gets exact clock values and can prescribe infinitely precise delays. Robust controller synthesis aims at synthesizing a strategy tolerant to some timing perturbation: there should exist some $\delta > 0$ such that the strategy is still winning, even if its prescribed delays are modified by at most δ time units. These perturbations are chosen by the player Environment.

The difficulty of this problem relies in the analysis of the cycles of the region automaton: indeed, we need to guarantee that Controller is able to repeat some cycle whatever how Environment perturbs his delays. We prove that this property is related to a notion of convergence in cycles, which has recently been characterized using so-called forgetful cycles [BA11]. Using these techniques, we show the decidability of the robust controller synthesis problem for timed automata against Büchi objectives. We prove that the problem is ExpTime-complete, which is also the complexity of the corresponding problem in the non-robust setting.

The framework considered so far is fully antagonist, as the strategy of the controller should be winning against *every* behavior of the environment. In order to address more realistic settings, we consider different stochastic models for the environment, and prove decidability of the associated robust controller synthesis problems. We consider for instance a model in which the perturbation of the delays are chosen randomly. These works have been carried out in two different papers. The first one has been co-authored with Patricia Bouyer, Nicolas Markey and Ocan Sankur [27]. The second paper has been written with Youssouf Oualhadj while he was my post doctoral student and with Ocan Sankur [22].

Part II presents results related to the *analysis of transformations*.

In Chapter 4, I present different models of string-to-string transformations. After introducing (two-way) finite-state transducers, I define the model of streaming string transducers (SST) which has recently been introduced by Rajeev Alur and Pavol Černý. Last, I introduce a logic based formalism to describe transformations. The two important classes of rational and regular string transformations are defined, and characterizations are given by means of the different models of transducers.

Chapter 5 gathers the results I obtained on string-to-string transformations. Most of these results are motivated by the analysis of the streamability of these transformations, *i.e.* decide whether it is possible to realize a transformation using a single left-to-right traversal of the input word, and using a memory that does not depend on the length of the input word. I thus start by formally describe this notion of streamability, and I build a link with the class of deterministic one-way finite state transducers. This class is characterized, among rational functions, by the so-called twinning property. This settles this streamability problem for the class of rational functions, as the twinning property can be decided in polynomial time.

Concerning regular string functions, a necessary condition for streamability is to belong to the class of rational string functions. In terms of transducers, this amounts to decide, given a functional two way finite-state transducer, whether there exists an equivalent one-way transducer. Though very natural, this problem had never been studied before, to the best of our knowledge. Showing that this problem is decidable is rather difficult, and I present a proof relying on a construction proposed by Rabin and Scott for automata. This is a joint work with Emmanuel Filiot, Olivier Gauwin and Frédéric Servais published in [14].

The model of streaming string transducer allows to express any regular string function using a deterministic one-way automaton, equipped with a finite set of string variables. While streamability amounts to checking whether it is possible to realize a transformation using a deterministic one-way transducer, a more general problem is thus to minimize the number of variables required to express a given transformation using an SST. Together with Jean-Marc Talbot and with Laure Daviaud who owns currently a post doctoral researcher position in our group, we address this problem in a particular case, namely that of rational functions (which have a counterpart in terms of SST). In this setting, we introduce a generalization of the twinning property of order k that characterizes precisely transformations realizable using k variables. In addition, we prove that this property is decidable, thus allowing to identify the minimal number of variables required to express a given rational function using this subclass of SST. This recent work is not published yet.

A strength of regular languages is their equivalent representation by automata, logic and algebra. When considering regular string functions, we recover the logical equivalence using MSO string-to-string transformations. The class of first-order definable languages has been thoroughly studied with an equivalent algebraic characterization in terms of aperiodic monoid. Recent results provided characterizations of the class of first-order definable string-to-string transformations by means of aperiodic SST [FKT14], and aperiodic deterministic two-way transducers [CD15]. Together with Luc Dartois while he was a post-doctoral researcher in Marseille, we recently proposed different transformations between deterministic two-way transducers and SST, and proved that they preserve aperiodicity.

Chapter 6 deals with tree-to-string transformations. I start with a presentation of visibly pushdown automata for which I proposed with Jean-Marc Talbot and our PhD student Mathieu Caralp a trimming procedure [8, 9]. Then, I present the model of visibly pushdown transducers we introduced in [16] with Emmanuel Filiot, Jean-François Raskin, Jean-Marc Talbot and Frédéric Servais. This model takes as input the linearization of an unranked tree, and produces as output a string that does not necessarily represent a tree. We prove the decidability of important properties such as functionality and k-valuedness, and study closure properties. We also show that the problem of type-checking is undecidable, and that visibly pushdown transducers are not closed under composition.

The two previous negative results motivate the study of subclasses of visibly pushdown transducers which define tree-to-tree transformations, *i.e.* such that the output string is the linearization of an unranked tree. We study two different such subclasses: a syntactic one and a semantical one. We show that the second class is more general, decidable in polynomial time, closed under composition, and owns a decidable type checking problem. Last, we compare the tree-to-tree class of transductions defined by the syntactic subclass with known models of hedge-to-hedge transductions. These results are joint works with my PhD student Mathieu Caralp, and my colleagues Emmanuel Filiot, Frédéric Servais and Jean-Marc Talbot, published in [26] and [6]. A journal version of these results is under revision.

The original motivation for studying visibly pushdown transducers is to use an automaton model close from actual implementations handling XML streams. In particular, our objective is to characterize streamable transformations. The notion of streamability needs to be defined, and we consider here different levels of streamability. As a first step, we require that the memory used to evaluate the transformation may depend on the height of the input (nested) word, but should not depend on its length. This is reasonable as common XML documents have very low depth. We manage to characterize this class of transformations using an adaptation of the twinning property, and prove that this class is decidable in coNP-time. However, the memory may depend *exponentially* in the height of the input word, which may be too much for some applications. As a second step, we thus introduce a stronger twinning property, and prove the following important facts: this twinning property defines a class of transductions (it does not depend on the transducer, but only on the transduction it defines); it subsumes the class of subsequentializable visibly pushdown transducers; it can be decided in coNP-time; and it can be evaluated using a memory that depends *quadratically* on the current height of the input word. This work has been carried with Emmanuel Filiot, Olivier Gauwin and Frédéric Servais, and has been published in [13]. A journal version of these results is under revision.

Conclusion summarizes the main results presented in this thesis, and highlights the common tools on which rely their proofs. The major perspectives are also described and illustrated with mid-term and long-term goals.

Other contributions not in this thesis

Several papers are not mentioned in this thesis, because I decided to focus on the two topics of robustness of timed systems, and analysis of transformations. I briefly describe these results below.

Analysis of (time) Petri nets The Karp and Miller approach [KM69] is a very well-known technique to decide the coverability problem, which consists in determining whether there exists a reachable marking dominating some given marking m. The Karp and Miller algorithm actually allows to compute the so-called minimal coverability set of a Petri net, which can be understood as a minimal (finite) over-approximation of the reachability set. Unfortunately, this algorithm is known to be very inefficient, as the same computations may be repeated several times. In order to avoid this, a technique of pruning was introduced before in [Fin93], but it appeared to be erroneous as demonstrated in a recent work [GRVB10]. With Frédéric Servais, who was a PhD student at ULB at that time, we managed to propose an involved pruning technique which allows to safely apply Karp and Miller acceleration, yielding a very efficient procedure to compute the minimal coverability set of a Petri net. These results appeared in the communications [24, 25]. Recently, a rather simple algorithm has been proposed in [VH14], together with involved techniques for the representation of intermediate data structures, as well as heuristics for the exploration of the Petri net. The resulting implementation shows very good performances. It seems very interesting to develop a new implementation of our procedure in order to test whether it can also benefit from these technical improvements.

The most widely used time extension of Petri nets is that of *time Petri nets*, which are usually considered under a urgent semantics, in the sense that a transition can never miss its deadline due to time elapsing. Reachability is undecidable for this class, and the undecidability proof heavily relies on this urgency condition. With Arnaud Sangnier, who was a PhD student at ENS Cachan at that time, we decided to study the decidability status of the reachability problem when this urgency condition is relaxed. This non-urgent semantics is usually named the *weak* semantics. Surprisingly, we managed to prove that under the weak semantics, the set of reachable markings is exactly the reachability set of the underlying untimed Petri net. As a consequence, reachability is decidable for weak time Petri nets. These results have been published in [23].

Controller synthesis for timed systems During my post-doctoral stay in Brussels, I was involved in a European project with industrial partners. One of them provided us the description of a pump system associated with an oil tanker and asked us to automatically synthesize a controller for the pump respecting some safety constraints regarding the oil level inside the tank, as well as some optimization constraints in terms of energy consumption. I derived from this description a modelization of the controller synthesis problem fitting the input syntax of the tool UppAal-TiGa, and we managed to automatically synthesize such a timed controller. This work, done in collaboration with Franck Cassez, Jan J. Jessen, Kim G. Larsen and Jean-François Raskin, has been published in [10].

In some real complex systems, it is not reasonable to assume that the controller has access to the full description of the current configuration of the system. The model of timed games with partial observation allows to take this difficulty into account. Allowing the controller to know the value of some internal variable corresponds to adding some monitor/sensor to the system, which may come with a cost. In order to minimize the cost of the synthesized controllers, we addressed the problem of identifying the minimal set of observations under which a given system is controllable. We proposed an algorithm based on refinement techniques, as well as some heuristics to identify quickly the right set of observations. A prototype implementation has been realized in order to validate our algorithms. These results, obtained with

Peter Bulychev, Franck Cassez, Alexandre David, Kim G. Larsen and Jean-François Raskin, have been published in [5].

Timed systems are a particular case of hybrid systems, which combine discrete steps, modeled by a finite number of states, and continuous evolution, modeled by a finite set of real-valued variables governed by differential equations. This constitutes a very expressive setting, for which numerous problems are undecidable. The work reported in [19] has been done with my colleague Peter Niebert and his PhD student Janusz Malinowski. This work is an attempt to make controller synthesis feasible for a class of hybrid systems. In order to avoid the state space explosion, we proposed a hierarchical algorithm which considers increasing subsets of variables. Using an implementation of our approach, we have been able to automatically synthesize a controller for an inverted pendulum. This work has been carried out in the context of a collaboration with the start-up Novadem whose aim is to develop unmanned quadricopters.

Characteristic formulae for event-recording automata In this work, we considered with Omer-Landry Nguena-Timo who was a PhD student at LaBRI at that time a subclass of timed automata called event-recording automata. We have studied a fixpoint logic tailored to express timed properties corresponding to this class of timed automata. We have shown that we can build in this logic a formula which characterizes a given event-recording timed automaton up to timed bisimilarity. We also proved that a previously introduced timed logic is not expressive enough for this objective, contrarily to what was claimed in the literature. This work has been published in [20, 21].

Weighted visibly pushdown automata Last, in addition to works presented in this thesis on visibly pushdown transducers, I have also worked on weighted visibly pushdown automata with Jean-Marc Talbot and our PhD student Mathieu Caralp [7]. Our work concerns the boundedness problem for visibly pushdown automata equipped with weights considered in the semi-ring $(\mathbb{N}, +, \times)$. We provided a characterization of automata $\mathcal{A}$ whose evaluation is bounded, *i.e.* such that there exists $k \in \mathbb{N}$ such that for every input word w, the value associated by $\mathcal{A}$ to w is at most k. Our characterization is based on patterns, which generalize the patterns known for automata on finite words. This allows as a particular case to characterize finitely ambiguous visibly pushdown automata. Using transformations between (weighted) visibly pushdown automata and (weighted) tree automata, we also derived a characterization of bounded weighted tree automata over the semi-ring $(\mathbb{N}, +, \times)$.

Tree transducers with origin semantics I started recently a new collaboration with Sebastian Maneth in order to study decision problems inspired by databases motivations and that can be addressed using tree transducers techniques. Such a problem is the determinacy of a query by a view: we want to determine whether a view of a database (that can be understood as an extraction of it) determines a query (the view is sufficient to answer the query). Though undecidable in general, we have studied this problem using a new semantics for tree transducers (the so-called origin semantics that has recently been introduced in [Boj14]), and we have proven several decidability results. This joint work with Emmanuel Filiot, Sebastian Maneth and Jean-Marc Talbot has been published in [15].

A note on references

References to my publications use numbers, and refer to my list of publications, which can be found as Appendix A. References to other works use alphanumeric keys, and refer to the bibliography which can be found beginning on page 115.

Part I

Robustness of timed systems

Chapter 1

Preliminaries on timed automata

Contents

1.1 Timed automata and model checking

Model checking is a formal method for automatically checking whether a system satisfies a given property. It is usually defined for labelled transition systems and temporal logics. We will instantiate here the definition for timed automata, which will be the central object of our further developments.

Timed automata Given a finite set of clocks $\mathcal{C}$, we call *valuations* the elements of $\mathbb{R}_{\geq 0}^{\mathcal{C}}$. For a subset $R \subseteq \mathcal{C}$ and a valuation ν, $\nu[R \leftarrow 0]$ is the valuation defined by $\nu[R \leftarrow 0](x) = \nu(x)$ for $x \in \mathcal{C} \setminus R$ and $\nu[R \leftarrow 0](x) = 0$ for $x \in R$. Given $d \in \mathbb{R}_{\geq 0}$ and a valuation ν, the valuation $\nu + d$ is defined by $(\nu + d)(x) = \nu(x) + d$ for all $x \in \mathcal{C}$. We extend these operations to sets of valuations in the obvious way. We write $\vec{0}$ for the valuation that assigns 0 to every clock.

An atomic clock constraint is a formula of the form $k \preceq x \preceq' l$ or $k \preceq x - y \preceq' l$ where $x, y \in \mathcal{C}$, $k, l \in \mathbb{Z} \cup \{-\infty, \infty\}$ and $\preceq, \preceq' \in \{<, \leq\}$. A *guard* is a conjunction of atomic clock constraints. A valuation ν satisfies a guard g, denoted $\nu \models g$, if all constraints are satisfied when each $x \in \mathcal{C}$ is replaced with $\nu(x)$. We denote by $[\![g]\!]$ the set of clock valuations that satisfy g. We write $\Phi_{\mathcal{C}}$ for the set of guards built on $\mathcal{C}$.

Let Σ be a finite alphabet. A *timed automaton* $\mathcal{A}$ over Σ is a tuple $(\mathcal{L}, \mathcal{C}, \ell_0, F, E)$, where $\mathcal{L}$ is a finite set of locations, $\mathcal{C}$ is a finite set of clocks, $E \subseteq \mathcal{L} \times \Phi_{\mathcal{C}} \times \Sigma \times 2^{\mathcal{C}} \times \mathcal{L}$ is a set of edges, $\ell_0 \in \mathcal{L}$ is the initial location, and $F \subseteq \mathcal{L}$ is the set of final locations. An edge $e = (\ell, g, a, R, \ell')$ is also written as $\ell \xrightarrow{g,a,R} \ell'$. A state of $\mathcal{A}$ is a pair $q = (\ell, \nu)$ composed of a location ℓ and a clock valuation ν over $\mathcal{C}$.

The set of possible behaviors of a timed automaton can be described by the set of its runs, as follows. A *run* of $\mathcal{A}$ is a sequence $q_1 e_1 q_2 e_2 \ldots$ where $q_i \in \mathcal{L} \times \mathbb{R}_{\geq 0}^{\mathcal{C}}$, and writing $q_i = (\ell, \nu)$, either $e_i \in \mathbb{R}_{\geq 0}$, in which case $q_{i+1} = (\ell, \nu + e_i)$, or $e_i = (\ell, g, a, R, \ell') \in E$, in which case

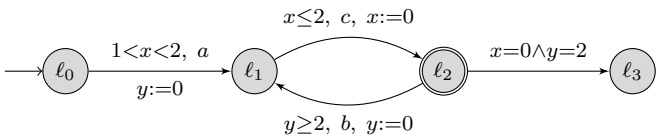


Figure 1.1: A timed automaton $\mathcal{A}$ from [Pur00]

$\nu \models g$ and $q_{i+1} = (\ell', \nu[R \leftarrow 0])$. We say that e_i is a delay action when $e_i \in \mathbb{R}_{\geq 0}$, and that e_i is a discrete action with label a when $e_i = (\ell, g, a, R, \ell') \in E$. We may write $q \xrightarrow{t} q'$ to denote a transition of delay t, and $q \xrightarrow{a} q'$ to denote a discrete transition labelled by a. W.l.o.g., we assume that discrete and delay actions alternate. We denote by $\mathsf{state}_i(\rho)$ the i-th state of any run ρ, by $\mathsf{first}(\rho)$ its first state, and, if ρ is finite, $\mathsf{last}(\rho)$ denotes the last state of ρ. The *label* of a run $\rho = q_1 e_1 q_2 e_2 \dots e_{2n} q_{2n+1}$ with $e_1 \in E$ is the timed word $w = (a_i, d_i)_{1 \leq i \leq n}$ where a_i is the label of e_{2i-1} and d_i is the delay e_{2i}. The set of runs in $\mathcal{A}$ starting in state q is denoted $\mathsf{Runs}(\mathcal{A}, q)$.

A run ρ is accepting if it starts in the initial state $(\ell_0, \vec{0})$ and ends in some final location. A timed word $w \in (\Sigma \times \mathbb{R}_{\geq 0})^*$ is accepted by a timed automaton $\mathcal{A}$ if there exists an accepting run ρ whose label is w. The language of $\mathcal{A}$ is the set of the timed words it accepts, and is denoted $L(\mathcal{A})$. Given a timed word w, the untimed word associated with w (*i.e.* its projection on Σ^*) is denoted by $\mathsf{untimed}(w)$. This notation is lifted to languages, and we denote by $\mathsf{untimed}(L(\mathcal{A}))$ the untimed language of $\mathcal{A}$, defined as the projection of $L(\mathcal{A})$ on Σ^*.

Standard definitions of timed automata also allow invariants on locations which restrict time elapsing. For the sake of simplicity, we do not consider this technical addition here, however our results hold in presence of invariants.

We say that a state q is reachable in $\mathcal{A}$ if there exists a run ρ such that $\mathsf{first}(\rho) = (\ell_0, \vec{0})$ and $\mathsf{last}(\rho) = q$. We denote by $\mathsf{reach}(\mathcal{A})$ the set of reachable states in $\mathcal{A}$.

We say that $\mathcal{A}$ is *deterministic* if for every pair of edges $(\ell, g_1, a, R_1, \ell_1)$ and $(\ell, g_2, a, R_2, \ell_2)$ starting in the same location ℓ and over the same letter a, we have $[\![g_1]\!] \cap [\![g_2]\!] = \emptyset$.

An example of timed automaton is given on Figure 1.1. This timed automaton is deterministic, and one can verify that its untimed language can be described by the regular expression $ac(bc)^*$.

Timed transition system The semantics of timed systems is often given as a timed transition system. Formally, a *timed transition system* (TTS for short) over Σ is a transition system $S = (Q, q_0, Q_f, \rightarrow)$, where Q is the set of states, $q_0 \in Q$ is the initial state, $Q_f \subseteq Q$ is the set of final states and the transition relation $\rightarrow$ consists of delay moves $q \xrightarrow{d} q'$ (with $d \in \mathbb{R}_{\geq 0}$), and discrete moves $q \xrightarrow{a} q'$ (with $a \in \Sigma_\varepsilon$). Moreover, we require standard properties of time-determinism, additivity and continuity for the transition relation $\rightarrow$.

The semantics of $\mathcal{A}$ defined above can be understood using the (infinite state) timed transition system $[\![\mathcal{A}]\!] = (Q, q_0, Q_f, \rightarrow)$, where Q is the set of states of $\mathcal{A}$, $q_0 = (\ell_0, \vec{0})$ is the initial state, $Q_f = \{(\ell, \nu) \mid \ell \in F\}$ and $\rightarrow \subseteq Q \times (\mathbb{R}_{\geq 0} \cup \Sigma) \times Q$ has been defined above.

Timed (bi)-simulation: Let $S = (Q, q_0, Q_f, \rightarrow)$ and $S' = (Q', q'_0, Q'_f, \rightarrow')$ be two TTS. A relation $\mathcal{R} \subseteq Q \times Q'$ is a *timed simulation* if and only if, $(q_0, q'_0) \in \mathcal{R}$ and for every $\sigma \in \Sigma_\varepsilon \cup \mathbb{R}_{\geq 0}$, $q_1 \in Q$, $q'_1 \in Q'$ such that $(q_1, q'_1) \in \mathcal{R}$, if $q_1 \xrightarrow{\sigma} q_2$, then there exists q'_2 such

that $q_1' \xrightarrow{\sigma} q_2'$ and $(q_2, q_2') \in \mathcal{R}$. We will say that S' *simulates* S and write $S \preceq S'$ when such a relation $\mathcal{R}$ among states of S and S' exists. If in addition $\mathcal{R}^{-1}$ is a timed simulation relation from S' to S, then we say that $\mathcal{R}$ is a timed bisimulation. We say that S and S' are *timed bisimilar* when such a relation $\mathcal{R}$ among states of S and S' exists, and write $S \approx S'$.

Region automaton. In order to symbolically reason about the infinite state space of timed automata, [AD94] defines an equivalence relation (of finite index) as follows. Let $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, F, E)$ be a timed automaton, and M be the largest integer occurring in $\mathcal{A}$. Two valuations ν and ν' are equivalent iff the following conditions hold [1]:

- for all $c \in \mathcal{C}$, either $\nu(c)$ and $\nu'(c)$ are greater than M, or $\lfloor \nu(c) \rfloor = \lfloor \nu'(c) \rfloor$;
- for all $c, c' \in \mathcal{C}$, if both $\nu(c)$ and $\nu(c')$ are lower than M, then
  - $\langle \nu(c) \rangle \leq \langle \nu(c') \rangle$ iff $\langle \nu'(c) \rangle \leq \langle \nu'(c') \rangle$;
  - $\langle \nu(c) \rangle = 0$ iff $\langle \nu'(c) \rangle = 0$.

This defines an equivalence relation, whose equivalence classes are referred to as *regions*. We write $[\nu]$ for the region containing ν, and $\overline{r}$ for the topological closure of the region r (w.r.t. the usual topology on $\mathbb{R}^{\mathcal{C}}_{\geq 0}$). The set of regions is finite and exponential in the size of the timed automaton. We define the *region automaton* as the finite automaton $\mathcal{R}(\mathcal{A}) = (\Gamma, \gamma_0, \rightarrow_a)$ where

- Γ is the set $\{(\ell, r) \mid \ell \in \mathcal{L},\ r \text{ region}\}$,
- γ_0 is the initial state (ℓ_0, r_0) where r_0 is the region which contains the valuation ν_0 with $\nu_0(c) = 0$ for every $c \in \mathcal{C}$,
- $\rightarrow_a \subseteq \Gamma \times (\Sigma \cup \{\tau\}) \times \Gamma$ and $((\ell, r), \sigma, (\ell', r')) \in \rightarrow$ iff the following holds:
  - either $\sigma \in \Sigma$ and $(\ell, \nu) \xrightarrow{\sigma} (\ell', \nu')$ is a transition of $[\![\mathcal{A}]\!]$ for some $\nu \in r$ and $\nu' \in r'$,
  - or σ is the symbol τ, and there exists $t \in \mathbb{R}_{\geq 0}$ s.t. $(\ell, \nu) \xrightarrow{t} (\ell', \nu')$ is a transition of $[\![\mathcal{A}]\!]$ for some $\nu \in r$ and $\nu' \in r'$.

The notions of path in the region automaton, trace of a path, ... are defined in the usual way. It is well known that this automaton is *time-abstract bisimilar* to the original timed automaton, which implies that, under the standard semantics, all reachability and more generally ω-regular properties can be checked equivalently on the original timed automaton or on the region automaton. We assume that classical properties of region automata are known, and refer to [AD94] for more details. Let us observe that the size of the region automaton is exponential in the size of the timed automaton, but that it can be explored on-the-fly yielding PSPACE algorithms (for checking emptiness for instance).

Symbolic representation of valuations Although the region automaton is a very important tool to obtain decidability results, it is not amenable to implementation. To this end, zones constitute a symbolic representation of sets of valuations that is more abstract, and yields to smaller state spaces. Formally, a zone Z is a convex set of valuations that can be

[1] $\lfloor \nu(c) \rfloor$ represents the integer part of $\nu(c)$ and $\langle \nu(c) \rangle$ represents its fractional part.

defined using a guard, *i.e.* such that there exists a guard g satisfying $Z = [\![g]\!]$. We denote by $\mathsf{Zones}(\mathcal{C})$ the set of zones defined on the set of clocks $\mathcal{C}$. The main asset of zones is their representation using the data structure of difference bound matrices (DBM for short) allowing to implement efficiently every basic operation (time elapsing, reset of clocks, intersection ...).

A *symbolic state* is a pair $(\ell, Z) \in \mathcal{L} \times \mathsf{Zones}(\mathcal{C})$. Consider a transition $e = (\ell, g, a, R, \ell') \in E$ of a TA $\mathcal{A}$. We define the operator Post^e computing the symbolic successors over e starting from the zone Z, with $Z \in \mathsf{Zones}(\mathcal{C})$, by $\mathsf{Post}^e(Z) = \{v' \in \mathbb{R}^{\mathcal{C}}_{\geq 0} \mid \exists v \in Z, \exists d \in \mathbb{R}_{>0} : v \models g \wedge v' = v[R \leftarrow 0] + d\}$. It is well known that $\mathsf{Post}^e(Z)$ is still a zone. We define similarly the operator Pre^e for the set of predecessors by e. Given a sequence of transitions ρ, we define the operators Post^ρ and Pre^ρ as the compositions of these operators for each transition of ρ. We define the set of successors from a symbolic state by $\mathsf{Succ}(\ell, Z) = \{(\ell', \mathsf{Post}^e(Z)) \in L \times \mathsf{Zones}(\mathcal{C}) \mid e = (\ell, g, a, R, \ell') \in E\}$.

Temporal logics One needs tools to specify the properties that the systems should fulfill. Temporal logics, introduced by Pnueli in the 70's, are central in model checking approach because there exist efficient model checking algorithms based on automata translations. We present below the Linear Temporal Logic, LTL for short. We will mainly consider linear-time temporal logics in this thesis.

Definition 1 (Logic LTL)**.** *The logic* LTL *over finite set of actions* Σ *is defined by the following grammar: (a ranges over the set of actions* Σ*)*

$$\mathsf{LTL} \ni \varphi \ ::= \ a \ \mid \ \varphi \vee \varphi \ \mid \ \neg\varphi \ \mid \ \mathbf{X}\,\varphi \ \mid \ \varphi\,\mathbf{U}\,\varphi$$

Semantics of LTL. We define the semantics of LTL over (finite or infinite) words over Σ. We thus assume we are given a word $u = (a_i)_i \in \Sigma^* \cup \Sigma^\omega$. Given a natural number $j \in \mathbb{N}$, we denote by u^j the word $(a_i)_{i \geq j}$. The satisfaction relation for LTL is denoted $\models$ and is defined inductively as follows (we omit the semantics of standard boolean operators):

$$\begin{array}{lcl} u \models a & \iff & a_0 = a \\ u \models \mathbf{X}\,\varphi & \iff & u^1 \models \varphi \\ u \models \varphi_1\,\mathbf{U}\,\varphi_2 & \iff & \exists\, i \geq 0 \text{ s.t. } u^i \models \varphi_2 \text{ and } \forall\, 0 \leq j < i, u^j \models \varphi_1 \end{array}$$

The set of words satisfying an LTL formula φ is denoted $[\![\varphi]\!]$.

In the following, we use classical shortcuts like $\mathbf{F}\,\varphi$ (which holds for $\texttt{true}\,\mathbf{U}\,\varphi$ where `true` denotes the "true" formula) or $\mathbf{G}\,\varphi$ (which holds for $\neg(\mathbf{F}\,(\neg\varphi))$).

In particular, LTL properties allow to express basic properties such as safety (something bad never happens, expressed by the $\mathbf{G}$ operator).

Last, given a timed automaton $\mathcal{A}$ and an LTL formula φ, we say that $\mathcal{A}$ satisfies φ iff for all $w \in L(\mathcal{A})$, we have $\mathsf{untimed}(w) \models \varphi$.

Model checking Given a timed automaton $\mathcal{A}$ and a property φ expressed in some formal language, the model checking problem consists in determining whether $\mathcal{A}$ satisfies φ. This problem can of course be instantiated with several classes of models and properties. We review some of them here, together with complexity and decidability results.

The model of the system is often described using an automata-based formalism, while the model of the property is usually given as some logical formula. The automata-based approach

to model checking, introduced in [VW86, Var95, KVW00] consists in translating the property into an automaton, and in using automata algorithms to solve the problem.

The problem of model checking of timed automata against *safety properties* is defined as follows:

Input: A timed automaton $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, F, E)$ and a subset $\mathsf{Bad} \subseteq \mathcal{L}$

Problem: Does $\mathsf{reach}(\mathcal{A}) \cap (\mathsf{Bad} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}) = \emptyset$ hold?

This problem can be reduced to a reachability problem in the region automaton of $\mathcal{A}$, and we obtain:

Theorem 1 ([AD94])**.** *Model checking of timed automata against safety properties is* PSpace-c.

The problem of LTL model checking for timed automata is defined as follows:

Input: A timed automaton $\mathcal{A}$ and an LTL formula φ

Problem: Does $\mathcal{A} \models \varphi$ hold?

Given a timed automaton, the satisfaction of an LTL formula only depends on its untimed language. As a consequence, we can also use the region automaton to decide this problem. Using well-known translations of LTL formula into Büchi automata, one obtains:

Theorem 2 ([AD94])**.** LTL *model checking for timed automata is* PSpace-c.

1.2 Timed games and controller synthesis

While model-checking aims at proving the correction of an existing system w.r.t. a specification, synthesis aims at building from a specification a correct-by-construction system. We will not consider reactive synthesis as introduced by Pnueli and Rosner in [PR89], but rather supervisory control as introduced in the seminal work of Ramadge and Wonham [RW87]. We refer the reader to [ELTV14] for a comparison of these two approaches.

Assume that we have the description of a system in which the set of transitions is partitioned into *controllable* and *uncontrollable* ones. The controller synthesis is formalized as a two-player zero-sum game (*i.e.* the two players are antagonist), played between Controller and Environment. While Controller may choose which controllable transition he wants to execute, Environment may play an uncontrollable transition at any time. The objective is then to decide the existence of a winning strategy for Controller.

This approach was first lifted to timed systems in [MPS95]. Let $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, F, E)$ be a timed automaton. In this setting, a strategy for Controller maps each finite run ending in some state (ℓ, ν) to a controllable transition (that can be executed from state (ℓ, ν)), or to wait (meaning that this player wants to let time elapse). Player Environment's strategies are similar, and if both players want to execute a transition, the priority is given to Environment. A strategy of Controller is declared winning iff the set of runs of the timed game it induces all satisfy Controller's objective, which is often given as an ω-regular property. Using a fixpoint computation on the set of regions of the timed automaton, one can prove the decidability of the controller synthesis in timed game automata for reachability and safety objectives:

Theorem 3 ([MPS95])**.** *The existence of a winning strategy for Controller in a timed game with reachability or safety objectives is decidable in* ExpTime.

This work led to numerous further developments, involving other semantics for timed games ([AMPS98, dAFH$^+$03]), symbolic approaches [CDF$^+$05] and the development of the too UppAal-TIGA. Recent works used this tool to solve case studies [10].

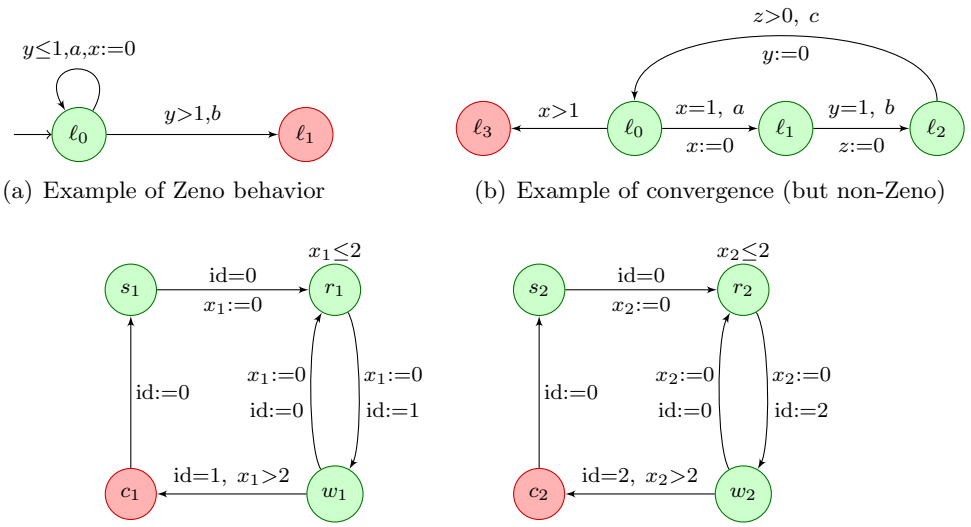
Already in [MPS95], but also in more recent works such as [dAFH$^+$03], a focus has been put on unrealistic strategies for Controller. For instance, it was admitted that a player should not win the game by following a Zeno behavior, *i.e.* performing infinitely many transitions in a finite amount of time. In [MPS95], this property was ensured using the so-called restriction of strictly non-Zeno timed automata. In [dAFH$^+$03], the authors proposed an encoding of this property in the abstract game used to solve the (timed) game, leading to the decision of the existence of non-Zeno winning strategies. As we will see in the following developments, Zeno behaviors are not the only pathological runs of timed automata. In some sense, our results on robust controller synthesis generalize the ones of [dAFH$^+$03].

1.3 Implementability of timed automata

Altough timed automata are a model widely used for representing real-time systems, there exists a gap between the models analyzed and their implementations that may cause the lost of correctness properties checked on the models. As we will see in the forthcoming examples, the semantics of timed automata relies indeed on several mathematical idealizations that cannot be guaranteed on realistic physical devices.

The line of works we present in the first part of this thesis builds on the work of Anuj Puri [Pur00] in which he proposed a parametric semantics for timed automata: given a timed automaton $\mathcal{A}$ and a rational number $\varepsilon > 0$, Puri defines the hybrid automaton $H(\mathcal{A}, \varepsilon)$ obtained by replacing clock variables by hybrid variables whose derivative belongs to the interval $[1 - \varepsilon, 1 + \varepsilon]$. Formally, this means that after a time elapsing of d time units, each clock variable is incremented of some d', with $d' \in [d - d\varepsilon, d + d\varepsilon]$ (with possibly a different d' for each clock). This enlarges the set of runs of $\mathcal{A}$, and Puri proved that the limit of the reachability set of these hybrid automata $H(\mathcal{A}, \varepsilon)$ when ε converges towards 0 does exist, and can be computed using the region automaton. The parametric semantics we will consider is strongly related to that of Puri.

Examples of timed automata The semantics of timed automata suffers from several mathematical idealizations that cannot be faithfully respected by implementations. For instance, there is no lower bound between two consecutive actions, clock values are considered with infinite precision, and communication between processes is instantaneous. We illustrate these assumptions with few examples of timed automata. First, the example depicted on Figure 1.2.(a) illustrates the well-known issue of Zeno behaviors that perform an infinite number of actions in a finite amount of time. In order to execute the sequence a^ω, one needs to perform an infinite number of a's in only one time unit. Second, the timed automaton implementing a timed version of Fischer's protocol for mutual exclusion is depicted on Figure 1.2.(c) for two participants. The two processes communicate through the shared variable id. Here, the correction of the system, *i.e.* the mutual exclusion of locations c_1 and c_2, is lost if the strict inequality $x_i > 2$ is replaced by a non-strict inequality. This shows the high dependance of the correction of the system to the perfect precision of its implementation concerning clock measures. It shows also that if the two processes use clocks that are not perfectly synchronized, then the same error may occur.



(a) Example of Zeno behavior

(b) Example of convergence (but non-Zeno)

(c) Fischer's protocol for mutual exclusion

Figure 1.2: Timed automata exhibiting timing anomalies.

Last, we consider the timed automaton depicted on Figure 1.2.(b). This automaton has been introduced in [CHR02] in order to exhibit a convergence phenomenon which is not a Zeno behavior. More precisely, one can prove that in order to loop forever in green locations along the word $(abc)^\omega$, the delay between the consecutive actions c and a will necessarily converge towards 0, but the global time is diverging.

A model of implementation In [DDR05], the authors introduce a *program semantics* for timed automata. The platform of implementation is characterized by two parameters $\delta_{\mathsf{precision}}$ and δ_{CPU} which correspond intuitively to the clock precision and to the processor speed. Analyzing this semantics with two parameters is not easy, and they show that one can instead study a rather simple parametric semantics depending on a single parameter δ. This semantics, called the *enlarged semantics*, enlarges every guard by the parameter δ. Given a timed automaton $\mathcal{A}$ and a rational number $\delta \geq 0$, we denote by $\mathcal{A}_\delta$ the timed automaton obtained by enlarging every guard with δ. For instance, the constraint $3 \leq x < 5$ becomes $3 - \delta \leq x < 5 + \delta$. They manage to prove that whenever the inequality $4\delta_{\mathsf{precision}} + 3\delta_{\mathsf{CPU}} \leq \delta$ holds, then the program semantics of a timed automaton $\mathcal{A}$ (with parameters $\delta_{\mathsf{precision}}$ and δ_{CPU}) is simulated by the δ enlarged semantics of $\mathcal{A}$. As a consequence, this intuitively implies that if one has proven the correction of $\mathcal{A}$ under its enlarged semantics *for some* positive δ, then using a sufficiently fast and precise implementation platform, one obtains the *correction* of the program semantics, as desired. This motivates the study of the enlarged semantics with an existential quantification on the parameter δ. The line of works we present in the next section follows this idea.

Related work In [AT05], the authors propose to build an explicit model of the execution platform of the timed automaton, in order to get a precise model of its implementation. This approach is expensive in terms of verification as it increases the size of the model, and

in addition it does not fulfill natural requirements. For instance, if one wants to change the platform execution (take a faster processor...), one has to redo the whole verification process.

In our context, imprecisions on clock values can accumulate along infinite runs. A different approach for implementability of timed automata has been presented in [ACS13] but it relies on the opposite assumption that clocks are periodically synchronized, and thus the drift of clocks is always bounded.

Another model of program implementation corresponds to the sampled semantics of timed systems. In this context, a fixed granularity of time τ is chosen, and every delay is a multiple of τ. Several works have studied the semantics of timed automata under such a sampled semantics (also called digitization), for finite and infinite words (see [OW03, KP05, AKY10]). In this context, the existence of a sampling rate under which some property is satisfied (existence of an infinite run, controllability w.r.t. some objective) has been studied for instance in [CHR02, KP05]. The formulation of the problem is thus similar to that of the problems we consider, and as we will discuss in the perspectives of Chapter 3, it may be possible to identify links between these two approaches.

Last, our robustness analysis of timed automata has also a semantical interpretation as we consider the behavior of the timed automaton obtained for every positive perturbation, whatever how small it is. It is thus related to other works that have studied alternative semantics for timed automata.

For instance, robust timed automata [GHJ97] are a semantical modification of timed automata based on a topological acceptance condition. This semantics is incomparable with the standard semantics as it may both add and remove behaviors. The aim of this semantics was to obtain new decidability results (such as language inclusion), and no link with implementability has been shown.

A probabilistic semantics has been introduced in [BBB+07]. It is based on a measure of the runs satisfying a given property, amongst the whole set of runs. The author study model-checking problems under this new semantics, and obtain decidability results in the one-clock case. However, this study is more language-based than related to implementability.

1.4 Robustness of timed automata

Based on the link established in [DDR05] between the implementability of some timed automaton $\mathcal{A}$ and its enlarged semantics $\mathcal{A}_\delta$, we will now be interested in studying this new semantics. Let us recall that given a non-negative rational number δ, $\mathcal{A}_\delta$ denotes the timed automaton obtained from $\mathcal{A}$ by enlarging every timing constraint by δ, *i.e.* replacing any constraint of the form $a \leq x \leq b$ by the enlarged constraint $a - \delta \leq x \leq b + \delta$.

Instead of fixing the value of δ under consideration, we consider δ as a parameter. It is well known that timed automata enriched with parameters are in general undecidable [AHV93] (see [BO14] for a recent improvement of the decidability results). We thus consider model checking problems with an existential quantification over $\delta > 0$: does there exist a *positive* value of δ under which the specification is met? This intuitively amounts to ask whether there exists a sufficiently fast and precise platform on which the timed automaton can be implemented safely.

It is worth observing that as we consider an enlarged semantics, our parameter only helps in relaxing the constraints. This way, using the terminology of [HRSV02], it corresponds to an 'upper' parameter. However, our results do not follow from those of [HRSV02] as in this

paper they allow the extremal value $\delta = 0$.

A linear-time property is defined as a subset of the set of timed words. For instance, LTL properties are linear-time properties. We say that a timed automaton $\mathcal{A}$ *robustly satisfies a linear-time property* P iff there exists $\delta > 0$ such that every timed word associated with an accepting run of $\mathcal{A}_\delta$ belongs to the set P. If this holds, then we write $\mathcal{A} \models P$.

Recall that the set of discrete behaviors of $\mathcal{A}_\delta$ increases with δ. We present now how infinitesimally small perturbations may induce new discrete behaviors in timed automata, and review existing results on robust model checking of timed automata.

Perturbations in timed automata When enlarging guards of a timed automaton, new behaviors may appear due to strict constraints, as it is the case for the timed automaton of Figure 1.2.(a). These new behaviors can easily be detected using the region graph construction.

More subtle behaviors are due to accumulation of perturbations. Consider the timed automaton $\mathcal{A}$ depicted on Figure 1.1. Location ℓ_3 is not reachable in the standard semantics of $\mathcal{A}$ while it is reachable in $\mathcal{A}_\delta$ for every positive δ. Indeed, let us consider state (ℓ_2, ν) with $\nu(x) = 0$ and $\nu(y) = 2$. We denote by t_1 (resp. t_2) the transition from ℓ_1 to ℓ_2 (resp. from ℓ_2 to ℓ_1), and let $\rho = t_1 t_2$. One can easily verify that in $\mathcal{A}_\delta$, the state (ℓ_2, ν) is reachable after $\lceil \frac{1}{2\delta} \rceil$ iterations of the cycle ρ (see Figure 1.3), hence it is reachable in $\mathcal{A}_\delta$ for every positive δ. Observe also that the smaller δ is, the longer is the witness of the reachability of this state in $\mathcal{A}_\delta$. This accumulation of perturbations will be at the heart of our further developments.

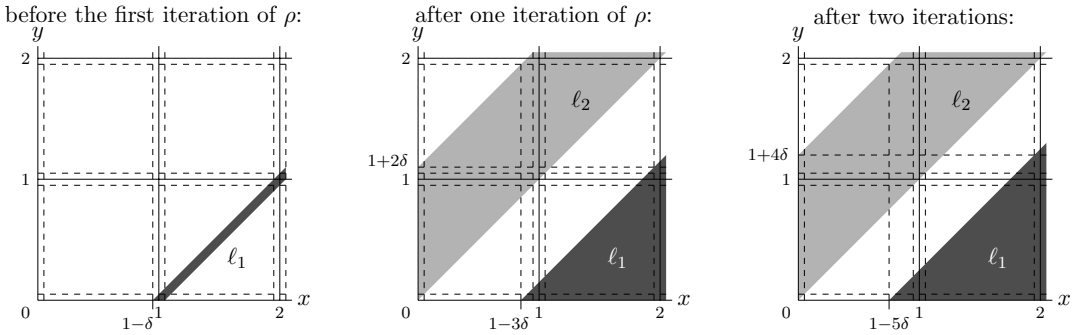


Figure 1.3: Reachable states during the forward analysis of $\mathcal{A}_\delta$.

Safety properties The *robust model checking problem of timed automata against safety properties* asks, given a TA $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, F, E)$ and a subset $\mathsf{Bad} \subseteq \mathcal{L}$, whether or not there exists $\delta > 0$ such that $\mathsf{reach}(\mathcal{A}_\delta) \cap (\mathsf{Bad} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}) = \emptyset$.

As an illustration, let us consider the behaviors of the timed automaton $\mathcal{A}$ of Figure 1.1 described above. Letting $\mathsf{Bad} = \{\ell_3\}$, $\mathcal{A}$ satisfies the safety property "never Bad", but it does not satisfy this safety property *robustly*.

This problem has been solved in [DDMR04], under two restrictions on timed automata that we describe now. A *progress cycle* in the region automaton of $\mathcal{A}$ is a cyclic path along which all the clocks are reset, and that does not only contain the initial region (*i.e.* the region where all the clocks are set to 0). We say that a timed automaton $\mathcal{A}$ is *progressive* if all the cycles in the region automaton $\mathcal{R}(\mathcal{A})$ are progress cycles.

We say that a timed automaton has bounded clocks if there exists a constant M such that for every reachable state, all clocks are bounded by M.

The bounded clocks hypothesis is not really restrictive since bounded timed automata are as expressive as standard timed automata (see for example [BFH+01]). The progress cycle hypothesis has first been considered in [Pur98], and in the framework of bounded timed automata, it is less restrictive than classical strong non-*Zenoness* assumptions.

Theorem 4 ([DDMR04])**.** *The robust model checking of progressive timed automata with bounded clocks against safety properties is* PSPACE-C.

In order to prove this result, the set $\mathsf{reach}^*(\mathcal{A}) = \bigcap_{\delta>0} \mathsf{reach}(\mathcal{A}_\delta)$ is considered, and the two following facts are proven:

1. checking whether $\exists \delta > 0$ s.t. $\mathsf{reach}_\delta(\mathcal{A}) \cap \mathsf{Bad} = \emptyset$ is equivalent to check whether $\mathsf{reach}^*(\mathcal{A}) \cap \mathsf{Bad} = \emptyset$,

2. checking whether $\mathsf{reach}^*(\mathcal{A}) \cap \mathsf{Bad} = \emptyset$ is decidable, and PSPACE-C.

While the first point has a topological nature, the second point relies on the analysis of the strongly connected components (SCC for short) of the classical region automaton construction. Intuitively, the main impact of the enlarged semantics can be summarized as follows: given a progress cycle ρ of a timed automaton $\mathcal{A}$, the reachability relation in $\mathcal{A}_\delta$ along the iterations of ρ is complete for any $\delta > 0$. As a consequence, in order to compute $\mathsf{reach}^*(\mathcal{A})$, as soon as the reachability set intersects the frontier of an SCC (in the topological sense), then the whole SCC can be added to the reachability set. As an illustration, the sets $\mathsf{reach}(\mathcal{A})$ and $\mathsf{reach}^*(\mathcal{A})$ are described on Figure 1.4, for the timed automaton $\mathcal{A}$ depicted on Figure 1.1. The difference is due to the iteration of the cycle around ℓ_1 and ℓ_2.

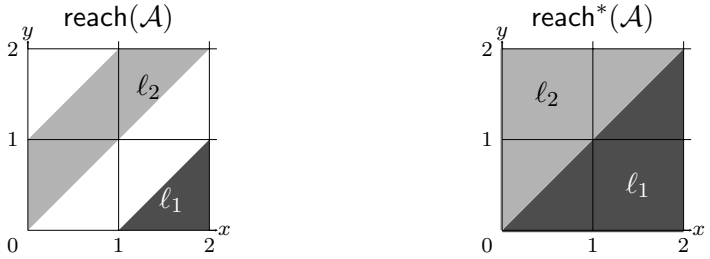


Figure 1.4: Differences between $\mathsf{reach}(\mathcal{A})$ and $\mathsf{reach}^*(\mathcal{A})$.

The drawback of the previous approach is that it is based on the region automaton construction, which is well known to not be amenable to implementation. As a remedy, zones are often considered as an adequate symbolic representation to provide efficient algorithms. In [DK06], an algorithm has been proposed to compute the set $\mathsf{reach}^*(\mathcal{A})$ using zones, but this algorithm only applies to flat timed automata, *i.e.* with no nested cycles. Intuitively, each cycle ρ is accelerated in isolation, using a greatest fixpoint computation of the so-called stable zone W_ρ, defined as the set of valuations having infinitely many successors and predecessors through cycle ρ. The stable zone represents the acceleration of the cycle ρ. Informally, the algorithm of [DK06], presented as Algorithm 1, proceeds in a forward exploration of the timed automaton using zones, and 'accelerates' cycles using their stable zones. We will present in Section 2.2 a parametric extension of this algorithm.

Algorithm 1 Computation of $\mathsf{reach}^*(\mathcal{A})$ [DK06].

Require: a progressive flat timed automaton $\mathcal{A}$ with bounded clocks.
Ensure: the set $\mathsf{reach}^*(\mathcal{A})$.

Compute $W_\rho = \nu Y.\mathsf{Pre}^\rho(Y) \cap \nu Y.\mathsf{Post}^\rho(Y)$, for each cycle ρ in $\mathcal{A}$.
Wait $= \{(\ell_0, Z_0)\}$; *// Initial states*
Passed $= \emptyset$;
while Wait $\neq \emptyset$ **do**
  pop (ℓ, Z) from Wait ;
  if $\forall (\ell, Z') \in$ Passed, $Z \not\subseteq Z'$ **then**
    if there exists a cycle ρ around location ℓ **then**
      if $Z \cap W_\rho \neq \emptyset$ **then**
        Wait $=$ Wait $\cup\, \mathsf{Succ}(\ell, W_\rho))$;
        Passed $=$ Passed $\cup \{(\ell, W_\rho)$;
    Wait $=$ Wait $\cup\, \mathsf{Succ}(\ell, Z)$;
    Passed $=$ Passed $\cup \{(\ell, Z)\}$;
return Passed ;

Last, we mention a more recent result, whose proof relies on an encoding of robustness into channel machines that we describe in Section 2.1. This result solves the robust model checking of safety properties for general timed automata.

Proposition 1 ([BMS11])**.** *The robust model checking of timed automata against safety properties is* PSPACE-C*.*

LTL properties A natural extension of these positive results for robust model checking of safety properties is to consider more general temporal logics. During my PhD, together with Patricia Bouyer and Nicolas Markey, we studied the logic LTL. We followed the standard approach for LTL model checking, relying on the conversion of LTL formula into Büchi automata.

We observed that the robust model checking of LTL properties can be reduced this way to robust model checking of co-Büchi properties. We proposed an algorithm solving the former problem by considering an extended region graph, based on an analysis of SCCs.

Theorem 5 ([3])**.** LTL *robust model checking is* PSPACE-C*.*

Untimed language preservation A timed automaton may be analyzed by means of its untimed language. A natural notion of robustness is then to require that this untimed language is preserved when small enough timing perturbations are introduced. The *untimed language preservation problem of timed automata* asks, given a TA $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, F, E)$, whether there exists $\delta > 0$ such that $\mathsf{untimed}(L(\mathcal{A}_\delta)) = \mathsf{untimed}(L(\mathcal{A}))$. If this is the case, we say that $\mathcal{A}$ is language-robust.

This problem has been studied in [San11] for progressive timed automata with bounded clocks. Further, the decidability of this problem also follows from Proposition 1 for general timed automata as we prove below.

Proposition 2. *Checking if a timed automaton $\mathcal{A}$ is language-robust is decidable.*

Proof. Given a finite timed automaton $\mathcal{A}$, the untimed language of $\mathcal{A}$ is a regular language. We can build a finite state automaton $\mathcal{C}$ accepting the complement of this language, equipped with final states. Let $\mathcal{B}$ be another timed automaton, and denote by $\mathcal{B} \otimes \mathcal{C}$ the product of $\mathcal{B}$ with $\mathcal{C}$. It is easy to verify that $\mathcal{B} \otimes \mathcal{C}$ never enters a final state iff $\mathsf{untimed}(L(\mathcal{B})) \subseteq \mathsf{untimed}(L(\mathcal{A}))$. As for any non-negative δ we have $L(\mathcal{A}) \subseteq L(\mathcal{A}_\delta)$, we obtain that $\mathcal{A}_\delta \otimes \mathcal{C}$ does not enter its final states iff $L(\mathcal{A}) = L(\mathcal{A}_\delta)$. As $\mathcal{C}$ is untimed, the two timed automata $\mathcal{A}_\delta \otimes \mathcal{C}$ and $(\mathcal{A} \otimes \mathcal{C})_\delta$ are equal. Our problem thus reduces to a robust safety problem for the automaton $\mathcal{A} \otimes \mathcal{C}$ which is decidable thanks to Proposition 1. One can verify that the algorithm obtained this way runs in 2-ExpSpace (see [San13]). □

Chapter 2

Robust model checking

Contents

In this chapter, we first consider a class of timed properties, defined as a large and expressive fragment of the Metric Temporal Logic, and prove that its robust model checking problem is ExpSpace-c. Then, we consider the quantitative robust model checking problem that aims at computing the maximal admissible perturbation. Last, we consider robust model checking problems for the model of time Petri nets.

2.1 Timed temporal logics

The temporal properties we have studied previously do not allow to express *timed* properties, as they only depend on the untimed words associated with accepting runs. When studying timed systems, it is however natural to consider real-time specifications.

Metric Temporal Logic (MTL [Koy90]) is a widely studied real-time extension of the logic LTL. While undecidable in general, several restrictions on the problem under consideration (finite *vs* infinite words, model-checking *vs* satisfiability), or fragments of the logic, allow to obtain decidability. A survey of these (un)decidability results can be found in [OW08].

An important decidable fragment of MTL is the one forbidding punctuality constraints, considered in [AFH96], and named Metric Interval Temporal Logic (MITL for short). However, the decidability proof relies on a translation of MITL formula into timed automata and on a product construction. This approach is not suitable to address robustness problems, as our techniques do enlarge *all* guards of the timed automaton, and that would implicitly mean that we do not consider the exact formula, but only an approximation of it.

Another decidable fragment has been identified in [BMOW07], with a completely different decision procedure relying on one-clock alternating timed automata and on channel automata. The model-checking problem for this fragment, called co-flat Metric Temporal Logic (coFlat-MTL), is ExpSpace-c. In this Section, we explain informally how to extend

the decision procedure presented in [BMOW07] to robust model-checking. These results have been published in [4].

Fragments of metric temporal logic

Linear-time properties are often defined *via* temporal logic formulae. In this paper, we focus on subclasses of Metric Temporal Logic.

Fix a finite, non-empty alphabet Σ. The syntax of MTL over Σ is defined by the following grammar:

$$\mathsf{MTL} \ni \phi ::= \sigma \mid \neg\sigma \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \,\mathbf{U}_I\, \phi \mid \phi \,\widetilde{\mathbf{U}}_I\, \phi$$

where $\sigma \in \Sigma$, and I is an interval of $\mathbb{R}_{\geq 0}$ with bounds in $\mathbb{N} \cup \{\infty\}$. MTL formulae are interpreted over timed words. Let $w = (a_i, d_i)_{i \geq 0}$ be a timed word, and $p \in \mathbb{N}$. The (pointwise) semantics of MTL is defined recursively as follows (we omit Boolean operations):

$$\begin{array}{rcl} w, p \models a & \iff & a_p = a \\ w, p \models \phi \,\mathbf{U}_I\, \psi & \iff & \exists i > 0 \text{ s.t. } w, p+i \models \psi, \ \displaystyle\sum_{p < j \leq i} d_j \in I \\ & & \text{and } \ \forall 0 < j < i, \ w, p+j \models \phi \\ w, p \models \phi \,\widetilde{\mathbf{U}}_I\, \psi & \iff & w, p \models \neg\Big((\neg\phi) \,\mathbf{U}_I\, (\neg\psi)\Big). \end{array}$$

If $w, 0 \models \phi$, we write $w \models \phi$.

Additional operators, such as `true` (true), `false` (false), $\Longrightarrow$, $\Longleftrightarrow$, $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{X}$, are defined in the usual way: $\mathbf{F}_I\, \phi \equiv \mathtt{true}\, \mathbf{U}_I\, \phi$, $\mathbf{G}_I\, \phi \equiv \mathtt{false}\, \widetilde{\mathbf{U}}_I\, \phi$, and $\mathbf{X}_I\, \phi \equiv \mathtt{false}\, \mathbf{U}_I\, \phi$. We also use pseudo-arithmetic expressions to denote intervals. For example, '$= 1$' denotes the singleton $\{1\}$.

Following [BMOW07], we identify the following syntactic fragments of MTL: LTL can be considered as the fragment of MTL in which modalities are not constrained (*i.e.* where $\mathbb{R}_{\geq 0}$ is the only constraining interval); Bounded-MTL is the fragment of MTL in which all interval constraints are bounded: observe that Bounded-MTL disallows unconstrained modalities, and is in particular not suitable to express invariance properties (the most basic type of temporal specifications). The fragment coFlat-MTL[1] has then been defined to remedy this deficiency:

$$\mathsf{coFlat\text{-}MTL} \ni \phi ::= \sigma \mid \neg\sigma \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \,\mathbf{U}_J\, \phi \mid \phi \,\mathbf{U}_I\, \underline{\psi} \mid \phi \,\widetilde{\mathbf{U}}_J\, \phi \mid \underline{\psi} \,\widetilde{\mathbf{U}}_I\, \phi$$

where J ranges over the set of bounded intervals, I over the set of all intervals, and the underlined formula $\underline{\psi}$ ranges over LTL.

One immediately sees that coFlat-MTL subsumes both LTL and Bounded-MTL, but it is not closed under negation. However, it is closed under invariance, and can then express *e.g.* bounded response-time or even richer formulae such as $\mathbf{G}\,(\mathit{request} \Rightarrow \mathbf{F}_{\leq 5}\,(\mathit{acquire} \wedge \mathbf{F}_{=1}\,\mathit{release}))$.

Channel automata

Channel automata are a formalism for reasoning about alternating timed automata that has been used in [BMOW07] to prove the ExpSpace-membership of the model-checking problem

[1] We do not explain this terminology here, and refer to [BMOW07] for deeper considerations.

for coFlat-MTL. Intuitively, they consist of a finite state automaton equipped with a FIFO channel with some operations on this channel.

A *channel automaton with renaming and occurrence testing* (CAROT for short) is a tuple $\mathcal{C} = (S, s_0, \Sigma, \delta, F)$, where S is a finite set of control states, $s_0 \in S$ is the initial control state, $F \subseteq S$ is a set of accepting control states, Σ is a finite alphabet, $\delta \subseteq S \times Op \times S$ is the set of transition rules, where

$$Op = \{\sigma!, \sigma? \mid \sigma \in \Sigma\} \cup \{zero?(\sigma) \mid \sigma \in \Sigma\} \cup \{R \mid R \subseteq \Sigma \times \Sigma\}$$

is the set of operations. Given a rule $\tau = (s, \alpha, s') \in \delta$, we define $op(\tau) = \alpha$.

Intuitively, operations $\sigma!$ and $\sigma?$ are the classical write and read operations, $zero?(\sigma)$ is a guard for testing the occurrence of σ in the channel, and $R \subseteq \Sigma \times \Sigma$ is interpreted as a global renaming. An *end-of-channel* marker can be used to count the number of times the whole channel is read: it suffices to add, from each state of the CAROT, an outgoing transition reading $\triangleright$, immediately followed by a transition writing $\triangleright$. That way, there is always a unique copy of $\triangleright$ on the channel (except when it has just been read). The number of transitions writing $\triangleright$ along a computation ρ is the number of *cycles* of ρ, denoted by $cycles(\rho)$. In the sequel, the CAROT are assumed to contain the end-of-channel marker $\triangleright$, and to have all their states equipped with a loop reading and writing that symbol.

We consider in the sequel a restricted version of the reachability problem for CAROTs, where we impose a bound on the "time" (measured here as the number of cycles of the channel): the *cycle-bounded reachability problem* for CAROTs is defined as follows: given a CAROT $\mathcal{C}$, an input word $w \in \Sigma^*$, and a cycle bound h, does $\mathcal{C}$ have an accepting computation ρ on w with $cycles(\rho) \leqslant h$?

In [BMOW07], a non-deterministic procedure is presented to check the existence of an accepting cycle-bounded computation using only polynomial space in the *value* of the cycle bound and in the size of the CAROT:

Theorem 6 ([BMOW07])**.** *The cycle-bounded reachability problem for CAROT is solvable in polynomial space in the size of the channel automaton, the size of the input word, and the value of the cycle bound.*

Encoding robustness problems using channel automata

This encoding is rather technical, hence we will only give an informal presentation of it. We will proceed in three steps:

- first, we explain how to simulate a timed automaton,
- second, we take into account the enlarged semantics, by adding special symbols on the channel,
- third, we encode the MTL formula by considering its translation into a one-clock alternating timed automaton.

Simulating a timed automaton Let $\mathcal{A}$ be a timed automaton, we describe a CAROT $\mathcal{C}_\mathcal{A}$. We explain how a state $(\ell, \nu) \in \mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}$ of a timed automaton is encoded as a configuration of $\mathcal{C}_\mathcal{A}$, by means of its discrete state and of its channel. First, the valuation ν is represented by a sequence of disjoint subsets C_0, C_1, ..., C_p of $\mathcal{C}$ obtained using standard region techniques.

More precisely, C_0 contains elements whose fractional part is 0 and the others C_i gather elements with the same fractional part and are sorted according to the increasing order of fractional parts.

The intended encoding of a state (ℓ, ν) in the CAROT $\mathcal{C}_\mathcal{A}$ is the following: the integral values of the clocks of $\mathcal{A}$ are stored in the discrete state of the CAROT, as well as the set C_0 and the current location ℓ. The other C_i's are stored on the channel, from C_1 (recently written on the channel) to C_p (the next item to be read from the channel).

Example 1. *Consider state (ℓ, ν) with $\mathcal{C} = \{x_1, x_2, x_3, x_4\}$ and $\nu(x_1) = 1.4$, $\nu(x_2) = 1.0$, $\nu(x_3) = 3.4$ and $\nu(x_4) = 2.1$. The discrete state of the CAROT stores the location ℓ, the mapping $\lfloor \nu \rfloor = (1, 1, 3, 2)$, and the set $C_0 = \{x_2\}$. The content of the channel is depicted on the left of Figure 2.1 (the channel is read from the right, and written from the left).*

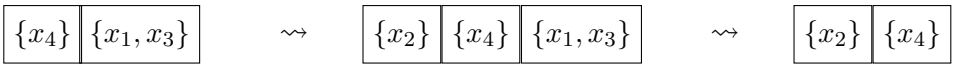


Figure 2.1: Evolution of channel content during time-elapsing

We explain how transitions can be executed by the CAROT. We start with time-elapsing transitions. There are two cases: if the set C_0 stored in the discrete state is non-empty, then this set if pushed onto the channel, and the discrete state is updated by letting C_0 be the empty set. Otherwise ($C_0 = \emptyset$ in the discrete state), the channel is read and thus the set C_p is removed from the channel. Then, the discrete state is updated by letting $C_0 = C_p$, and by incrementing the integral values associated with clocks belonging to C_p. In particular, observe that the channel represents one time unit, and that a cycle in this channel machine exactly corresponds to one time unit elapsing.

Example 2. *We illustrate time-elapsing on state (ℓ, ν) given in Example 1. When time-elapsing is performed, as $C_0 \neq \emptyset$, the set C_0 is pushed onto the channel, and the discrete state is updated with $C_0 = \emptyset$.*

A second time-elapsing removes the set $\{x_1, x_3\}$ from the channel (as we have now $C_0 = \emptyset$), and the discrete state is then composed of location ℓ, of the mapping $\lfloor \nu \rfloor = (2, 1, 4, 2)$, and of the set $C_0 = \{x_1, x_3\}$.

The evolution of the channel content during these two time-elapsing transitions is depicted on Figure 2.1.

For an edge $e = (\ell, g, a, R, \ell')$ of $\mathcal{A}$, whether it can be executed can easily be checked using the discrete state of the CAROT. If this is the case, then one updates the location in the discrete state. The reset of clocks belonging to R can be performed using rewriting operations on the channel and an update of the integral part in the discrete state.

Consider as initial configuration the discrete state representing the state $(\ell_0, \vec{0})$, and the initial channel content $w = \varepsilon$. Denoting by $[\![\mathcal{C}_\mathcal{A}]\!]$ the transition system associated to $\mathcal{C}_\mathcal{A}$, one can prove the following lemma stating the correctness of the construction:

Lemma 1. *The transition systems $[\![\mathcal{C}_\mathcal{A}]\!]$ and $[\![\mathcal{A}]\!]$ are time-abstract bisimilar.*

Enlarged semantics Our objective is to simulate the behavior of the automaton $\mathcal{A}_\delta$ for arbitrarily small values of δ. To this end, we consider values of δ of the form $\frac{1}{n}$ with $n \in \mathbb{N}$

arbitrarily large. We consider a new symbol Δ belonging to the channel alphabet and we introduce n copies of this symbol onto the channel. These symbols are (almost) always present on the channel (as soon as a Δ symbol is removed from the channel, a new one is pushed onto it). The intended meaning of these symbols is a partitioning of one time unit. Intuitively, if n symbols Δ are present on the channel, they should represent the n values $\frac{1}{n+1}, \ldots, \frac{n}{n+1}$. In particular, clocks that are before the first symbol Δ (resp. after the last symbol Δ) have a fractional value smaller than $\frac{1}{n+1}$ (resp. greater than $\frac{n}{n+1}$). As a consequence, in order to simulate the enlarged semantics, we consider that clocks that are before the first symbol Δ (resp. after the last symbol Δ) have a small fractional value (resp. a large fractional value).

Example 3. *We illustrate the role of symbols Δ with the state (ℓ, ν) considered in Example 1. Suppose that there are 3 symbols Δ on the channel, and that they indeed correspond to real numbers $\frac{1}{4}$, $\frac{2}{4}$ and $\frac{3}{4}$ respectively. The new channel content is depicted on Figure 2.2. In particular, observe that as clock x_4 is before the first symbol Δ, the guard $x_4 \leq 2$ can be fired in the enlarged semantics, although we have $\nu(x_4) = 2.1$.*

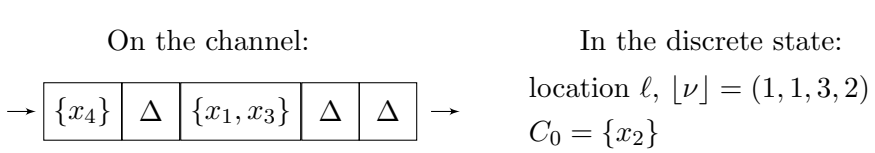


Figure 2.2: Encoding of the state of Example 1 in a CAROT $\mathcal{C}_{\mathcal{A}}^{\Delta}$ with initial content Δ^3

We explain how to simulate the enlarged semantics. Assuming the previous property on clock values, we thus have to store in the control part of the CAROT which clocks are "before" (resp. "after") the first (resp. last) symbol Δ. Whereas this can easily be done for the clocks that have been recently written on the channel, this is not possible for the clocks lying at the head of the channel (this would require to store the position of each clock w.r.t. each symbol Δ). Instead, we use non-determinism to allow the CAROT make predictions about the content of the head of the channel, and then we verify when reading clocks from the channel that these predictions were correct.

Let us denote by $\mathcal{C}_{\mathcal{A}}^{\Delta}$ the CAROT just described, and denote by (d, c_n) its initial configuration composed of the discrete state d representing the state $(\ell_0, \vec{0})$, and the initial channel content $c_n = \Delta^n$. Though we cannot ensure precisely the property introduced above about time values associated with symbols Δ, we manage to prove the following property:

Lemma 2. *For any $n \geq 3$, we have:*

- *if $L(\mathcal{A}_{\frac{1}{n}}) \neq \emptyset$, then $\mathcal{C}_{\mathcal{A}}^{\Delta}$ has a time-divergent accepting computation starting in (d, c_n)*
- *if $\mathcal{C}_{\mathcal{A}}^{\Delta}$ has a time-divergent accepting computation starting in (d, c_n), then $L(\mathcal{A}_{\frac{2}{n}}) \neq \emptyset$.*

Encoding the MTL formula The last step of the construction of the CAROT for encoding the robust model-checking problem consists in adding the part related to the temporal formula φ (in MTL). This part will be handled in a very similar way as in [BMOW07]. First, we build the one-clock alternating timed automaton $\mathcal{B}_{\neg\varphi}$ corresponding to $\neg\varphi$ (see [OW05]). Then, we build the product of the CAROT $\mathcal{C}_{\mathcal{A}}^{\Delta}$ with a CAROT simulating the behavior of $\mathcal{B}_{\neg\varphi}$. It is worth noticing here that the model of CAROT allows to enlarge only a subset of guards,

contrary to the enlarged semantics of timed automata. Indeed, the special symbols Δ are used to simulate enlarged guards in $\mathcal{A}$, while guards of $\mathcal{B}_{\neg\varphi}$ are not enlarged. The resulting CAROT, say $\mathcal{C}^{\Delta}_{\mathcal{A},\neg\varphi}$, running from the initial configuration (d', c'_n) corresponding to the initial configuration (d, c_n) of $\mathcal{C}^{\Delta}_{\mathcal{A}}$ and to the initial state of $\mathcal{B}_{\neg\varphi}$, simulates joint behaviors of $\mathcal{C}^{\Delta}_{\mathcal{A}}$ and $\mathcal{B}_{\neg\varphi}$. The accepting condition for $\mathcal{C}^{\Delta}_{\mathcal{A},\neg\phi}$ is the Büchi condition given by 'flattening' $\mathcal{B}_{\neg\varphi}$. The results of [BMOW07] combined with our above results yield:

Theorem 7 ([4]). *Let $\mathcal{A}$ be a timed automaton and φ be an* MTL *property. Then $\mathcal{A} \not\models \varphi$ iff for all $n \geq 3$, $\mathcal{C}^{\Delta}_{\mathcal{A},\neg\varphi}$ has a time-divergent accepting computation starting in (d', c'_n).*

Decidability results

We present now the decidability results we obtain for timed temporal logics. Thanks to Theorem 7, solving a robust model-checking problem amounts to decide a kind of "universality" checking of the CAROT, where we universally quantify on the initial number of Δ's on the channel.

The algorithm to decide the robust model-checking problem for Bounded-MTL formula relies on the fact that the truth value of a Bounded-MTL formula ϕ along a run ϱ only depends on the first h time units of ϱ, where h is the sum of the constants appearing in ϕ [BMOW07]. This allows to pump in the computations of the CAROT, reducing the universality checking to checking the property for a large enough value of n. We obtain:

Theorem 8 ([4]). *The robust model-checking problem for* Bounded-MTL *is* ExpSpace-c *(and* PSpace-c *if constants of the formula are given in unary).*

The case of coFlat-MTL is more involved than that of Bounded-MTL. The reason is that, unlike Bounded-MTL, the truth value of a coFlat-MTL formula ϕ along a run ϱ does not only depend on a prefix of ϱ of bounded duration. Instead, we use a decomposition result for infinite accepting runs proven in [BMOW07] (Theorem 12), involving three different cases:

- finite parts, of bounded length
- finite parts, of unbounded length, corresponding to an untimed subformula of ϕ
- an infinite suffix, verifying some LTL formula

In addition, the two first types can appear a bounded number of times.

Each of the three cases can be solved using existing techniques: the first case corresponds to Bounded-MTL formula, the second case amounts to verify that a configuration is reachable from another one for every positive δ, *i.e.* a robust safety problem, and the third case corresponds to the robust model-checking of LTL formula.

Gathering these results, we prove:

Theorem 9 ([4]). *The robust model-checking problem for* coFlat-MTL *is* ExpSpace-c.

2.2 Quantitative robust model checking

The results presented before allow to decide the existence of a positive perturbation δ under which a given property is satisfied. When the answer is positive, some of the algorithms presented also allow to identify a witness of the form $1/2^{|\mathcal{A}|}$. While this allows to deduce a

correct value for the parameter δ, computing the largest value of δ for which the enlarged semantics meets the specification was still an open problem. We are interested here in this last problem, which we call the *quantitative problem of robustness* for safety properties.

In this section, we present results published in [18]. We will prove that the quantitative robustness problem for safety properties is decidable for flat timed automata (*i.e.* where each location belongs to at most one cycle). In addition, we show that the maximal safe value of δ is a rational number. To this end, we solve a more general problem: we prove that it is possible to compute, for a flat timed automaton $\mathcal{A}$, the reachability set of $\mathcal{A}_\delta$ parameterized by δ. We call it the parametric reachability set. For this computation, we present a forward algorithm based on parametric zones (recall that a zone is a constraint on clocks). This algorithm can be understood as a parametric version of the symbolic algorithm proposed in [DK06] for flat timed automata. We then tackle two issues: the termination of our procedure and its correctness. For the first aspect, as we are in a parametric setting, we need completely new arguments of termination (the number of parametric zones we compute cannot be bounded as it is the case for zones). Considering a graph representation of zones introduced in [CJ99], we obtain proofs of termination depending only on the number of clocks, and not on the constants appearing in the automaton. To our knowledge, this constitutes an original approach in the context of timed automata. Regarding correctness, we identify under which conditions the enlarged reachability set coincides with the standard reachability set. This allows us to propose an algorithm computing the parametric reachability set of a flat timed automaton.

Graph representation of constraints

In the sequel, we will use a representation of clock constraints as a weighted directed graph introduced in [CJ99]. We recall here only succinctly its definition. Intuitively, the value of a clock can be recovered from its date of reset and the current time. The vertices of the graph represent these values, with one duplicate for each fired transition. Constraints on clock values are expressed as weights on arcs. Given a sequence of edges ρ, we will only consider the operator Post^ρ, but all our results also apply to the operator Pre^ρ.

More formally, we consider a set of clocks $\mathcal{C} = \{x_1, \ldots, x_n\}$ with $n = |\mathcal{C}|$. We let x_0 be a fictive clock whose value is always zero and define $\mathcal{C}_0 = \mathcal{C} \cup \{x_0\}$. We introduce a new clock τ that is never reset and thus represents the total elapsed time. In addition, for each clock $x_i \in \mathcal{C}$ we let variable X_i denote $X_i = \tau - x_i$; X_i thus represents last date of reset of clock x_i. We denote $\overrightarrow{V}$ the vector of variables defined as $(\tau, X_1, \ldots, X_n)$.

For an edge $e = (\ell, g, a, R, \ell')$, we define the formula $T^e(\overrightarrow{V}, \overrightarrow{V'})$ which expresses the relationship between values of the variables before (represented by $\overrightarrow{V}$) and after the firing of the edge (represented by $\overrightarrow{V'} = (\tau', X'_1, \ldots, X'_n)$):

$$T^e(\overrightarrow{V}, \overrightarrow{V'}) := (\tau \leq \tau') \wedge \bigwedge_{i=1}^{n} \left(X_i \leq \tau \wedge X'_i \leq \tau' \wedge \overline{g} \wedge \bigwedge_{x_i \in R} \tau = X'_i \wedge \bigwedge_{x_i \notin R} X_i = X'_i \right)$$

where $\overline{g}$ is the constraint g where for any i, clock x_i is replaced by $\tau - X_i$. In this formula, τ represents the time at which the edge e has been executed, while τ' represents the current time.

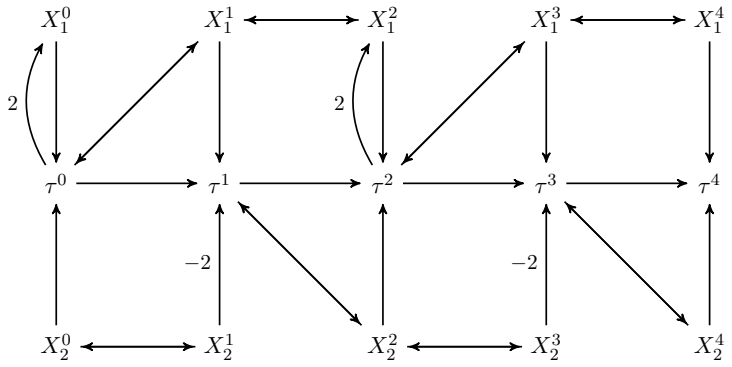


Figure 2.3: An example of graph $G(\rho)$ associated with a sequence of edges ρ

Let $\rho = e_1 \dots e_m$ be a sequence of edges. For $j \in \{0, \dots, m\}$, we denote by $\overrightarrow{V^j}$ the vector $(\tau^j, X_1^j, \dots, X_n^j)$. Then we define formula $T^\rho(\overrightarrow{V^0}, \overrightarrow{V^m})$ expressing the constraints between variables before and after the firing of the sequence ρ as follows:

$$T^\rho(\overrightarrow{V^0}, \overrightarrow{V^m}) := \exists \overrightarrow{V^1}, \dots, \overrightarrow{V^{m-1}}. \bigwedge_{j=0}^{m-1} T^{t_{j+1}}(\overrightarrow{V^j}, \overrightarrow{V^{j+1}})$$

We associate with formula $T^\rho(\overrightarrow{V^0}, \overrightarrow{V^m})$ a weighted directed graph whose vertices are variables used to define the formula, and arcs represent constraints of the formula:

Definition 2 (Graph $G(\rho)$)**.** *Let $\rho = e_1 \dots e_m$ be a sequence of edges of $\mathcal{A}$. The weighted directed graph $G(\rho)$ has a set of vertices $\mathcal{S} = \bigcup_{j=0}^{m} V^j$, where $V^j = \{\tau^j, X_1^j, \dots, X_n^j\}$. Given two vertices $X, X' \in \mathcal{S}$ and a weight $c \in \mathbb{Q}$, there is an arc from X to X' labelled by c if and only if the formula $T^\rho(\overrightarrow{V^0}, \overrightarrow{V^m})$ contains the constraint $X - X' \leq c$.*

Example 4. *Consider the timed automaton of Figure 1.1, page 19, with $x_1 = x$ and $x_2 = y$. Two iterations of the cycle around location ℓ_1 correspond to the following sequence of edges:*

$$\rho \;=\; \xrightarrow{x_1 \leq 2,\ x_1 := 0} \quad \xrightarrow{x_2 \geq 2,\ x_2 := 0} \quad \xrightarrow{x_1 \leq 2,\ x_1 := 0} \quad \xrightarrow{x_2 \geq 2,\ x_2 := 0}$$

The graph depicted on Figure 2.3 represents $G(\rho)$ (arcs without label have weight 0). Let us illustrate the construction of $G(\rho)$ with few examples:

- *clock x_1 is reset by the first edge, hence there are arrows in both directions labeled by 0 between nodes τ^0 and X_1^1 (the last date of reset for x_1 is τ^0),*
- *clock x_2 is* not *reset by the first edge, hence there are arrows in both directions labeled by 0 between nodes X_2^0 and X_2^1 (the last date of reset for x_2 is unchanged),*
- *the upper bound $x_1 \leq 2$ of the first edge is represented by the arrow from τ^0 to X_1^0 labeled by 2.*

For any path p in some graph $G(\rho)$, we write $w(p)$ the total weight of the path. Suppose now that there is no cycle of negative weight in graph $G(\rho)$. Let P_{end}^ρ denote the set of

minimal weighted paths from a vertex in V^0 to another vertex in V^0. Given a path $p \in P^\rho_{end}$ starting in vertex X^0_i and ending in vertex X^0_l, we define the following clock constraint: $C(p) = x_l - x_i \leq w(p)$.

The following correctness properties follow from results of [CJ99]:

Proposition 3. *Let ρ be a sequence of transitions. We have:*

- *there exists a cycle γ with $w(\gamma) < 0$ in $G(\rho)$ $\iff$ $\mathsf{Post}^\rho(\top) = \emptyset$*
- *if there is no cycle of negative weight, then $[\![\bigwedge_{p \in P^\rho_{end}} C(p)]\!] = \mathsf{Post}^\rho(\top)$*

More generally, given a zone Z, one can modify the graph $G(\rho)$ so as to include constraints of Z on vertices in V^0. This way, the mapping C applied on paths in P^ρ_{end} then defines the zone $\mathsf{Post}^\rho(Z)$. Similarly, the zone $\mathsf{Pre}^\rho(Z)$ can be represented by adding constraints of Z on vertices in $V^{|\rho|}$.

Given a zone defined as the result of the firing of a sequence of transitions, this representation allows to recover how the constraints are obtained. Thus, the graph stores the complete history of the constraints.

Accelerating computations of greatest fixpoints

In the sequel, we use this construction in the particular case of the iteration of a cycle ρ, given as a sequence of edges of a TA. We consider the sequence of zones $(Z_k)_{k\geq 0}$ defined by $Z_0 = \top$ and $Z_{k+1} = \mathsf{Post}^\rho(Z_k)$. Note that by monotonicity of Post^ρ, the sequence $(Z_k)_{k\geq 0}$ is decreasing and converges towards $Z_\infty = \nu Y.\mathsf{Post}^\rho(Y)$. According to Proposition 3, note that constraints defining zone Z_k can be obtained from shortest paths in graph $G(\rho^k)$.

Moreover, we will only be interested in vertices at the frontier between the different copies of the graph of ρ. Then, given a clock $x_i \in \mathcal{C}_0$ and an index $j \leq k$, *vertex X^j_i now denotes the date of reset of clock x_i after the j-th execution of ρ* (this notation is a shorthand for the notation $X^{j\times|\rho|}_i$, as this last notation will never be used anymore).

We prove the following lemma that provides a bound for termination only dependent on the number of clocks. Note that this result does not require the cycle ρ to be progressive neither the clocks to be bounded. This lemma relies on a simple pumping in the graph $G(\rho^k)$. In our work, we use this result to compute the greatest fixpoints $\nu Y.\mathsf{Post}^\rho(Y)$ and $\nu Y.\mathsf{Pre}^\rho(Y)$ in a parametric setting. More recently, this technique has been used in [DHS+14] to check whether a cycle can be executed infinitely many times.

Lemma 3. *Let $N = |\mathcal{C}_0|^2$, and $k \geq N$. If $Z_{k+1} \subsetneq Z_k$, then we have $Z_\infty = \emptyset$.*

Sketch of proof. We deduce from $Z_{k+1} \subsetneq Z_k$ that there exists a constraint $x_p - x_q \leq \cdot$ that is strictly smaller in Z_{k+1} than in Z_k. This constraint is obtained as a shortest path c between vertices X^{k+1}_q and X^{k+1}_p in the graph $G(\rho^{k+1})$ (see Figure 2.4). This path c necessarily contains arcs in $G(\rho)$ (otherwise the constraint $x_p - x_q \leq \cdot$ would be the same in Z_{k+1} and in Z_k). As $k \geq |\mathcal{C}_0|^2$, we can identify a pair (u, v) of vertices occurring twice, as depicted on Figure 2.4, say after iterations i and j of ρ. The weight of the part of c between these iterations (depicted by dashed lines on Figure 2.4) is necessarily negative (otherwise we could remove it). By iterating this part of c, we obtain the emptiness of the limit Z_∞. □

A consequence of Lemma 3 is that the greatest fixpoint $Z_\infty = \nu Y.\mathsf{Post}^\rho(Y)$ can be computed using at most $|\mathcal{C}_0|^2 + 1$ iterations of the operator Post^ρ, and this number does not depend on the constraints used in the timed automaton.

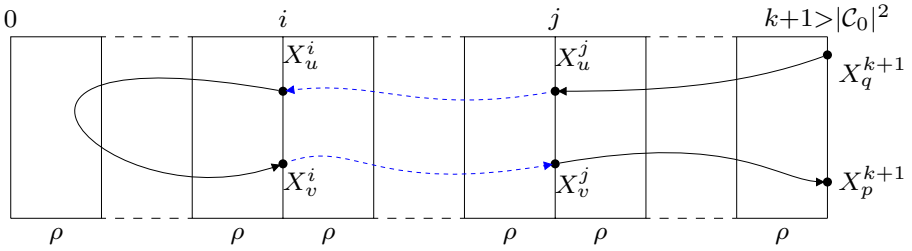


Figure 2.4: Pumping lemma : a path from X_q^{k+1} to X_p^{k+1} using arcs in $G_\rho^\top$ exhibits a return path between pairs of vertices (X_u^i, X_v^i) and (X_u^j, X_v^j).

Parametric zones

In order to be able to describe the parametric reachability set, we introduce a parametric extension of zones. Intuitively, we allow the use of the parameter Δ in the equalities. for instance, the constraint $x_1 \leq 2 + 3\Delta \wedge x_2 - x_1 \leq 3 + \Delta$ is a parametric clock constraint representing a parametric zone.

Formally, we denote by $\mathcal{PG}(\mathcal{C})$ the set of *parametric (clock) constraints* generated by the grammar $g ::= g \wedge g \mid x - y \leq k + b\Delta$, where $x \neq y \in \mathcal{C}_0$, $k \in \mathbb{Q}$ and $b \in \mathbb{N}$. Δ is a parameter representing a delay and b represents the accumulation of this delay.

Example 5. *Consider the zones depicted on Figure 1.3, page 1.3. On the first picture, the zone associated with location ℓ_1 can be described by the following parametric clock constraint:*

$$-1 - \Delta \leq y - x \leq -1 + \Delta \wedge x \leq 2 + \Delta \wedge y \leq 1 + 2\Delta$$

The zone associated with location ℓ_1 on the second picture can be described by the following constraint:

$$y - x \leq -1 + 3\Delta \wedge x \leq 2 + \Delta \wedge y \leq 1 + 4\Delta$$

Given a parametric constraint g and $\delta \in \mathbb{Q}_{\geq 0}$, we denote by $g(\delta)$ the constraint obtained by evaluating the parameter Δ in δ. As the parameter helps in "relaxing" the clock constraint, we have that $\delta \leq \delta'$ implies $[\![g(\delta)]\!] \subseteq [\![g(\delta')]\!]$.

Definition 3 (Parametric Zone)**.** *A* parametric zone $\mathcal{Z}$ *over* $\mathcal{C}$ *is a partial mapping from* $\mathbb{Q}_{\geq 0}$ *to zones over* $\mathcal{C}$*, which satisfies the following properties: (i) its domain* $\mathrm{dom}(\mathcal{Z})$ *is an interval with rational bounds, and (ii) it can be defined as the parametric satisfiability set of a parametric clock constraint,* i.e. *there exists* $g \in \mathcal{PG}(\mathcal{C})$ *such that for all* $\delta \in \mathrm{dom}(\mathcal{Z}), \mathcal{Z}(\delta) = [\![g(\delta)]\!]$*. We denote by* $\mathsf{PZones}(\mathcal{C})$ *the set of parametric zones on* $\mathcal{C}$*.* [2]

The data structure of DBMs for zones can be extended to parametric zones. The resulting data structure, called piecewise linear DBMs (PLDBM for short) is presented in [Jau09]. A PLDBM is a square matrix of size $|\mathcal{C}|+1$ whose entries are continuous piecewise linear functions of δ. Intuitively, when computing intersections of parametric zones we obtain conjunctions of the form $x \leq 2 + 3\Delta \wedge x \leq 3 + \Delta$. We handle them by splitting the set of possible values of Δ so as to exhibit a unique strongest constraint (for the previous example, we obtain $x \leq 2 + 3\Delta$ if $\Delta \leq \frac{1}{2}$, and $x \leq 3 + \Delta$ otherwise). We obtain this was constraints represented by piecewise linear functions. We can prove that elementary operations (time elapsing, reset, intersection with an atomic constraint...) can be performed in quadratic time. Observe that our parametric constraints are less expressive than those considered in [AAB00] as they only

[2] In the sequel, Z and Y denote a zone, while $\mathcal{Z}$ and $\mathcal{Y}$ denote a parametric zone.

involve conjunctions. Thus, we do not have to split DBMs and our data structure has lower complexities than parametric DBMs considered in [AAB00].

By default the considered domain for a parametric zone is $\mathbb{Q}_{\geq 0}$. Given a rational interval I, we denote $\mathcal{Z}_{|I}$ the parametric zone whose domain is restricted to I *i.e.*, $\mathrm{dom}(\mathcal{Z}_{|I}) = \mathrm{dom}(\mathcal{Z}) \cap I$, and which coincides with $\mathcal{Z}$ on $\mathrm{dom}(\mathcal{Z}_{|I})$. Given $\mathcal{Z}, \mathcal{Z}' \in \mathsf{PZones}(\mathcal{C})$, we define $\mathcal{Z} \subseteq \mathcal{Z}'$ if, and only if, we have $\mathrm{dom}(\mathcal{Z}) \subseteq \mathrm{dom}(\mathcal{Z}')$, and for any $\delta \in \mathrm{dom}(\mathcal{Z})$, $\mathcal{Z}(\delta) \subseteq \mathcal{Z}'(\delta)$. We say that a parametric zone $\mathcal{Z}$ is non-empty if there exists $\delta \in \mathrm{dom}(\mathcal{Z})$ such that $\mathcal{Z}(\delta) \neq \emptyset$. Let $\mathcal{Z}$ be a non-empty parametric zone. As the mapping represented by $\mathcal{Z}$ is monotone, we define $\delta_{\neg\emptyset}(\mathcal{Z}) = \inf\{\delta \geq 0 \mid \mathcal{Z}(\delta) \neq \emptyset\}$ the minimal value of the parameter for the zone it denotes to be nonempty. As $\mathcal{Z}$ only involves non-strict linear inequalities, $\delta_{\neg\emptyset}(\mathcal{Z})$ is a rational number and we have $\mathcal{Z}(\delta_{\neg\emptyset}(\mathcal{Z})) \neq \emptyset$ (provided that $\delta_{\neg\emptyset}(\mathcal{Z}) \in \mathrm{dom}(\mathcal{Z})$).

In order to perform parametric computations, we extend operators Post^ρ and Pre^ρ to a parametric setting. We denote these extensions by PPost^ρ and PPre^ρ. We also define the operator $\mathsf{Succ}(\ell, \mathcal{Z})$, where $\mathcal{Z} \in \mathsf{PZones}(\mathcal{C})$, using the PPost operator. All these operations can be made effective thanks to PLDBMs.

Last, we explain how to compute, in the parametric setting, the greatest fixpoint of PPost^ρ for every cycle ρ of the automaton. This relies on Lemma 3. Letting $N = |\mathcal{C}_0|$, we first evaluate the parametric zones $\mathcal{Z} = \mathsf{PPost}^{\rho^N}(\top)$ and $\mathcal{Z}' = \mathsf{PPost}^\rho(\mathcal{Z})$. Then, we determine the minimal value $\delta_0 = \min\{\delta \geq 0 \mid \mathcal{Z}(\delta) = \mathcal{Z}'(\delta)\}$. This definition is correct as $\mathcal{Z}' \subseteq \mathcal{Z}$ and, for large enough values of δ, all parametric constraints are equivalent to $\top$. Thus, we have $\nu\mathcal{Y}.\mathsf{PPost}^\rho(\mathcal{Y})(\delta) \neq \emptyset$ which implies by Lemma 3 that $\mathcal{Z}(\delta) = \mathcal{Z}'(\delta)$. Finally the greatest fixpoint can be represented by $\mathcal{Z}_{|[\delta_0;+\infty[}$ as Lemma 3 ensures that the fixpoint is empty for all $\delta < \delta_0$.

Parametric forward analysis with acceleration

We present Algorithm 2 for the parametric computation of $\mathsf{reach}(\mathcal{A}(\delta))$, which intuitively extends Algorithm 1 of [DK06] to a parametric setting. We first observe that we can modify Algorithm 1 as follows: in the test of Line 8, the stable zone W_ρ can be replaced by the fixpoint $\nu Y.\mathsf{Pre}^\rho(Y)$, and in Lines 9 and 10, the stable zone can be replaced by the fixpoint $\nu Y.\mathsf{Post}^\rho(Y)$.

We give now some explanations on Algorithm 2. First, at Line 1 we perform parametric computation of greatest fixpoints using the procedure proposed previously. Second, the test of intersection between the current zone and the greatest fixpoint of Pre^ρ is realized in a parametric setting by the computation at Line 8 of $\delta_{\min} = \delta_{\neg\emptyset}(\mathcal{Z} \cap \nu\mathcal{Y}.\mathsf{PPre}^\rho(\mathcal{Y}))$. Finally, we split the domain of the current parametric zone into intervals I_1 and I_2. In interval I_1, no acceleration is done for cycles and thus the set $\mathsf{reach}(\mathcal{A}(\delta))$ is computed. Acceleration techniques are used only for interval I_2, and for these values the algorithm computes the set $\mathsf{reach}^*(\mathcal{A}(\delta))$. We can prove that in this case, the equality $\mathsf{reach}(\mathcal{A}(\delta)) = \mathsf{reach}^*(\mathcal{A}(\delta))$ holds. Note that the test at Line 9 allows to handle differently the particular case of value $\delta_{\min}$ which does not always require to apply acceleration.

Theorem 10 ([18])**.** *Let $\mathcal{A}$ be a progressive timed automaton with bounded clocks. Algorithm 2 terminates and returns a finite set of parametric zones representing the parametric reachability set of $\mathcal{A}$.*

Algorithm 2 Parametric Computation of the Reachability Set.

Require: a progressive flat timed automaton $\mathcal{A}$ with bounded clocks.
Ensure: the set $\mathsf{reach}(\mathcal{A}(\delta))$ for all $\delta \in \mathbb{R}_{\geq 0}$.

Compute $\nu\mathcal{Y}.\mathsf{PPre}^\rho(\mathcal{Y})$ and $\nu\mathcal{Y}.\mathsf{PPost}^\rho(\mathcal{Y})$ for each cycle ρ of $\mathcal{A}$.
Wait $= \{(\ell_0, \mathcal{Z}_0)\}$; *// Initial States*
Passed $= \emptyset$;
while Wait $\neq \emptyset$ **do**
  pop $(\ell, \mathcal{Z})$ from Wait ;
  if $\forall(\ell, \mathcal{Z}') \in$ Passed, $\mathcal{Z} \not\subseteq \mathcal{Z}'$ **then**
    if there exists a cycle ρ around location ℓ **then**
      $\delta_{\min} = \delta_{\neg\emptyset}(\mathcal{Z} \cap \nu\mathcal{Y}.\mathsf{PPre}^\rho(\mathcal{Y}))$;
      if $\delta_{\min} = \delta_{\neg\emptyset}(\nu\mathcal{Y}.\mathsf{PPre}^\rho(\mathcal{Y}))$ **then**
        $I_1 = [0; \delta_{\min}]$; $I_2 =]\delta_{\min}; +\infty[$;
      else
        $I_1 = [0; \delta_{\min}[$; $I_2 = [\delta_{\min}; +\infty[$;
      Wait $=$ Wait $\cup\, \mathsf{Succ}(\ell, \mathcal{Z}_{|I_1}) \cup \mathsf{Succ}(\ell, \mathcal{Z}_{|I_2}) \cup \mathsf{Succ}(\ell, \nu\mathcal{Y}.\mathsf{PPost}^\rho(\mathcal{Y})_{|I_2})$;
      Passed $=$ Passed $\cup\, (\ell, \mathcal{Z}_{|I_1}) \cup (\ell, \mathcal{Z}_{|I_2}) \cup (\ell, \nu\mathcal{Y}.\mathsf{PPost}^\rho(\mathcal{Y})_{|I_2})$;
    else
      Wait $=$ Wait $\cup\, \mathsf{Succ}(\ell, \mathcal{Z})$;
      Passed $=$ Passed $\cup\, (\ell, \mathcal{Z})$;
return Passed ;

Quantitative safety Given a timed automaton $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, F, E)$ and a subset $\mathsf{Bad} \subseteq \mathcal{L}$, the quantitative robustness problem aims at computing the value $\delta_{\max} = \sup\{\delta > 0 \mid \mathsf{reach}(\mathcal{A}_\delta) \cap \mathsf{Bad} = \emptyset\}$. Once the reachable state space of the automaton is computed by Algorithm 2, it is easy to compute the value of $\delta_{\max}$. Simply compute the value $\delta_{\neg\emptyset}$ on each parametric zone associated with a bad location and keep the lower one: $\delta_{\max} = \min\{\delta_{\neg\emptyset}(\mathcal{Z}) \mid \exists \ell \in \mathsf{Bad}$ such that $(\ell, \mathcal{Z}) \in$ Passed$\}$. We thus obtain:

Theorem 11 ([18])**.** *The quantitative robustness problem for safety properties is decidable for flat progressive timed automata with bounded clocks. In addition, the value $\delta_{\max}$ is a rational number.*

2.3 Robustness issues in time Petri nets

In this section, we tackle the problem of robustness under small perturbations in the distributed and timed setting of time Petri nets [Mer74]. This model extends the well-known model of Petri nets and thus, unlike timed automata, the discrete control structure of time Petri nets is unbounded. Real-time constraints are introduced by associating time intervals with transitions representing "guards" within which the transition must fire once it is enabled. Our aim is to study the effect of small enlargement of those intervals.

We will present in this section how some of the presented results on timed automata can be applied to the study of robustness for time Petri nets. However, due to their unbounded discrete part, time Petri nets exhibit some new difficulties. The results presented in this section have been published in [1, 2].

Time Petri nets

An interval I of $\mathbb{R}_{\geq 0}$ is a $\mathbb{Q}_{\geq 0}$-*interval* iff its left endpoint belongs to $\mathbb{Q}_{\geq 0}$ and its right endpoint belongs to $\mathbb{Q}_{\geq 0} \cup \{\infty\}$. We denote by $\mathcal{I}(\mathbb{Q}_{\geq 0})$ the set of $\mathbb{Q}_{\geq 0}$-intervals of $\mathbb{R}_{\geq 0}$. Given $I \in \mathcal{I}(\mathbb{Q}_{\geq 0})$, we set $I^{\downarrow} = \{x \in \mathbb{R}_{\geq 0} \mid x \leq y \text{ for some } y \in I\}$, the *downward closure* of I.

A time Petri net $\mathcal{N}$ over Σ is a tuple $(P, T, {}^\bullet(.), (.)^\bullet, m_0, \Lambda, I)$ where P is a finite set of *places*, T is a finite set of *transitions* with $P \cap T = \emptyset$, ${}^\bullet(.) \in (\mathbb{N}^P)^T$ is the *backward* incidence mapping, $(.)^\bullet \in (\mathbb{N}^P)^T$ is the *forward* incidence mapping, $m_0 \in \mathbb{N}^P$ is the *initial* marking, $\Lambda : T \to \Sigma_\varepsilon$ is the *labeling function* and $I : T \mapsto \mathcal{I}(\mathbb{Q}_{\geq 0})$ associates with each transition a *firing interval*. We denote by $\alpha(t)$ (resp. $\beta(t)$) the lower bound (resp. the upper bound) of interval $I(t)$. An example is depicted on Figure 2.5 with usual graphical conventions.

A *configuration* of a TPN is a pair (m, ν), where m is a *marking* in the usual sense, *i.e.* a mapping in $\mathbb{N}^P$, with $m(p)$ the number of tokens in place p. A transition t is *enabled* in a marking m if $m \geq {}^\bullet t$. We denote by $En(m)$ the set of enabled transitions in m. The second component of the pair (m, ν) is a valuation over $En(m)$ which associates with each enabled transition its age, *i.e.* the amount of time that has elapsed since this transition was last enabled. An enabled transition t can be fired if $\nu(t)$ belongs to the interval $I(t)$. The result of this firing is as usual the new marking $m' = m - {}^\bullet t + t^\bullet$. Moreover, we define the predicate $\uparrow enabled(t', m, t)$ as follows: we say [3] that transition t' is *newly enabled* by firing of t from marking m (written $\uparrow enabled(t', m, t)$) iff $t' \in En(m - {}^\bullet t + t^\bullet) \wedge ((t' \notin En(m - {}^\bullet t)) \vee t = t')$. Clocks being reset by the firing of t from marking m are exactly clocks associated with transitions t' that are newly enabled by this firing.

The set $ADM(\mathcal{N})$ of *(admissible) configurations* consists of the pairs (m, ν) such that $\nu(t) \in I(t)^{\downarrow}$ for every transition $t \in En(m)$. Thus time can progress in a marking only when it does not leave the firing interval of any enabled transition.

Formally, the semantics of a TPN $\mathcal{N}$ is defined as the timed transition system $[\![\mathcal{N}]\!] = (Q, q_0, Q_f, \to)$ where $Q = Q_f = ADM(\mathcal{N})$, $q_0 = (m_0, \mathbf{0})$, and $\to$ is defined by:

- **delay moves:** $\forall d \in \mathbb{R}_{\geq 0}, (m, \nu) \xrightarrow{d} (m, \nu + d)$ iff $\forall t \in En(m)$, $\nu(t) + d \in I(t)^{\downarrow}$,
- **discrete moves:** $(m, \nu) \xrightarrow{\Lambda(t)} (m - {}^\bullet t + t^\bullet, \nu')$ iff $t \in En(m)$ is s.t. $\nu(t) \in I(t)$, and $\forall t' \in En(m - {}^\bullet t + t^\bullet)$, $\nu'(t') = 0$ if $\uparrow enabled(t', m, t)$ and $\nu'(t') = \nu(t)$ otherwise.

One may immediately notice that the semantics of TPN integrates a notion of urgency: as in delay moves, time can elapse only up to a point that does not violate any upper bound of time constraints attached to enabled transitions, some transitions have to fire before other ones.

Example 6. *Consider the TPN of Figure 2.5. Transition t_1 has to fire at a date d that lays between 1 and 2 time units after its enabling, and transition t_2 at a date d that is strictly greater than 2 time units after its enabling. According to the semantics, one cannot let 2 time units elapse without firing t_1. Hence, within this setting, only transition t_1 can fire.*

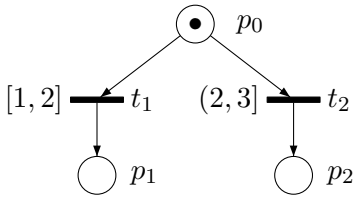


Figure 2.5: The TPN $\mathcal{N}_0$.

[3] We follow the standard definition of $\uparrow enabled(t', m, t)$, but alternative semantics exist (see [BCH+05a] for a comparison). One can prove that the undecidability results as well as the decidability results presented in this work also hold for TPN equipped with these alternative semantics.

The language of $\mathcal{N}$ is defined as the language of $[\![\mathcal{N}]\!]$ and is denoted by $L(\mathcal{N})$. We will also consider the untimed language $\mathsf{untimed}(L(\mathcal{N}))$. The reachability set of $\mathcal{N}$, denoted $\mathsf{reach}(\mathcal{N})$, is the set of markings $m \in \mathbb{N}^P$ such that there exists a reachable configuration (m, ν). A *bounded TPN* is a TPN $\mathcal{N}$ such that $\mathsf{reach}(\mathcal{N})$ is finite.

Perturbations in time Petri nets

Our goal is to consider a model of perturbation for time Petri nets similar to that considered for timed automata, in order to be able to transfer results from timed automata to time Petri nets. Given an interval $I \in \mathcal{I}(\mathbb{Q}_{\geq 0})$ and a rational number $\delta \geq 0$, we denote by I_δ the interval obtained by replacing its lower bound α by the bound $\max(0, \alpha - \delta)$, and its upper bound β by the bound $\beta + \delta$. Given a TPN $\mathcal{N}$, we denote by $\mathcal{N}_\delta$ the TPN obtained by replacing every interval I by the interval I_δ. We can then easily prove that Petri nets enjoy a monotony property similar to that of timed automata. This entails that if the system verifies a safety property for some perturbation δ_0, it will also verify this property for any $\delta \leq \delta_0$:

Lemma 4. *Let $\mathcal{N}$ be a TPN and $\delta \leq \delta' \in \mathbb{Q}_{\geq 0}$. We have $[\![\mathcal{N}_\delta]\!] \preceq [\![\mathcal{N}_{\delta'}]\!]$.*

Problems considered We now define robustness problems on TPN in a way which is consistent with the monotony property stated above.

Robust boundedness: Given a bounded TPN $\mathcal{N}$, does there exist $\delta \in \mathbb{Q}_{>0}$ such that $\mathcal{N}_\delta$ is bounded?

Robust untimed language preservation: Given a bounded TPN $\mathcal{N}$, does there exist $\delta \in \mathbb{Q}_{>0}$ such that $\mathsf{untimed}(L(\mathcal{N}_\delta)) = \mathsf{untimed}(L(\mathcal{N}))$?

We call a TPN $\mathcal{N}$ *robustly bounded* if there exists $\delta \in \mathbb{Q}_{>0}$ such that $\mathcal{N}_\delta$ is bounded. This problem is strongly related to the problem of ***robust safety*** asking, given a bounded TPN $\mathcal{N}$ with set of places P, and a marking $m \in \mathbb{N}^P$, whether there exists $\delta \in \mathbb{Q}_{>0}$ such that $\mathcal{N}_\delta$ does not cover [4] m. In fact, our undecidability and decidability results for robust boundedness will easily extend to this problem. However, the situation differs for robust untimed language preservation and so we treat this problem separately.

The above properties are considered in the setting of bounded TPN. However, from the undecidability results presented in this section, we can infer undecidability in the unbounded setting as well.

Sequential time Petri nets We identify the class of *sequential TPNs* that corresponds to a strict subclass of timed automata. We state their properties in detail here as they will be useful in later proofs. Also this exhibits a clear way to distinguish the relative power of TPNs and TA.

A TPN $\mathcal{N}$ is *sequential* if it satisfies the following property: for any reachable configuration (m, ν), and for any transitions $t, t' \in T$ that are fireable from (m, ν) (*i.e.* such that $t, t' \in En(m)$, $\nu(t) \geq \alpha(t)$ and $\nu(t') \geq \alpha(t')$), t and t' are in conflict, *i.e.* there exists a place p such that $m(p) < {}^\bullet t(p) + {}^\bullet t'(p)$.

The following lemma states robustness properties of sequential TPNs and their relation to timed automata.

[4] A TPN $\mathcal{N}$ over a set of places P covers a marking $m \in \mathbb{N}^P$ iff there exists $m' \in \mathsf{reach}(\mathcal{N})$ such that $m' \geq m$.

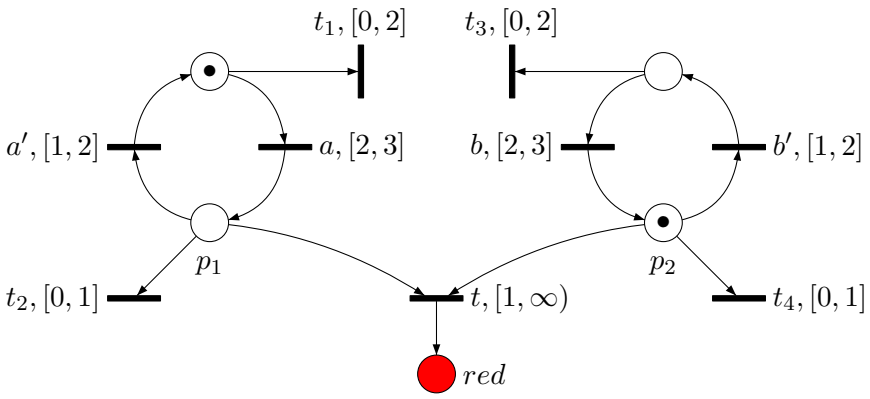


Figure 2.6: The TPN $\mathcal{N}_1$ exhibiting new discrete behaviors under infinitesimal perturbations.

Lemma 5. *We have the following properties:*

(*i*) *Checking whether a bounded TPN $\mathcal{N}$ is sequential is decidable.*

(*ii*) *If $\mathcal{N}$ is a sequential bounded TPN, then it can be translated into a timed automaton which resets every clock on each transition.*

(*iii*) *If $\mathcal{N}$ is sequential, then there exists $\delta \in \mathbb{Q}_{>0}$ such that* $\mathsf{reach}(\mathcal{N}_\delta) = \mathsf{reach}(\mathcal{N})$ *and* $\mathsf{untimed}(L(\mathcal{N}_\delta)) = \mathsf{untimed}(L(\mathcal{N}))$.

Accumulations of perturbations in time Petri nets As for timed automata, different sources of non robustness may be distinguished in time Petri nets. First, new behaviors due to neighbor configurations can appear (see for instance the TPN $\mathcal{N}_0$: for every $\delta > 0$, intervals $[1 - \delta, 2 + \delta]$ and $(2 - \delta, 3 + \delta]$ have a non-empty intersection). Such behaviors can easily be detected using the state class graph construction. In addition, if we consider a bounded horizon, one can compute an upper bound on δ so as to ensure that no new discrete behavior appears. The remaining interesting case is the accumulation of perturbations along infinite executions. For such situations, as for timed automata, new discrete behaviors exist due to the repetition of some cycle, with the property that the smaller δ is, the larger will be the number of iterations of the cycle required.

We have presented an example of a TA, provided by Puri [Pur00], exhibiting such an accumulation of perturbations (see $\mathcal{A}_0$ on Figure 1.1). Though translations from TA to TPN do exist for the exact semantics (see for instance [BCH+05b]), it is not guaranteed that they also preserve the enlarged semantics. In particular, the TPN resulting from the application of such translations to $\mathcal{A}_0$ do not exhibit such accumulations. Instead, we directly build a TPN which exploits the concurrency of Petri nets while using ideas similar to those of Puri.

This TPN, denoted $\mathcal{N}_1$, is depicted on Figure 2.6. Informally, it is composed of two components performing periodic behaviors around places p_1 and p_2 respectively. These components can be synchronized using transition t if places p_1 and p_2 are marked simultaneously. Intuitively, these components simulate the behavior of clocks x and y of the TA $\mathcal{A}_0$ in the sense that in the exact semantics, places p_1 and p_2 are marked respectively for the k-th time at timestamps τ_k and τ'_k such that $\tau'_k - \tau_k = 1$, and thus t is never fired. On the other hand, in the enlarged semantics, the value of $\tau_k - \tau'_k$ may 'drift' and thus t may be fired. Observe

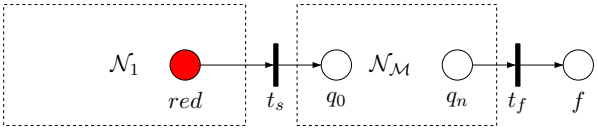


Figure 2.7: TPN $\mathcal{N}_2$ obtained by combining $\mathcal{N}_1$ and $\mathcal{N}_\mathcal{M}$.

that this example can be simplified using singleton intervals, but we avoid this to show that problems due to accumulation may arise even without singletons.

It is easy to prove that, for every positive δ, it is possible in the net $(\mathcal{N}_1)_\delta$ to fire simultaneously transitions a and b. As a consequence, places p_1 and p_2 can remain marked for 1 time unit, and the red place is reachable in $(\mathcal{N}_1)_\delta$.

Undecidability results

We use the TPN $\mathcal{N}_1$ of Figure 2.6 to prove undecidability of all the robustness problems considered for bounded TPN.

Theorem 12 ([1, 2])**.** *The problems of (i) robust boundedness, (ii) robust untimed language preservation and (iii) robust safety are undecidable for bounded TPN.*

Sketch. Let us first recall that given a Minsky machine $\mathcal{M}$ whose target control state is state q_n, it is possible to build a TPN $\mathcal{N}_\mathcal{M}$ satisfying the following properties (see [2] for details). First, as $\mathcal{N}_\mathcal{M}$ simulates exactly executions of $\mathcal{M}$, $\mathcal{N}_\mathcal{M}$ is bounded iff $\mathcal{M}$ is, and $\mathcal{N}_\mathcal{M}$ covers marking $\{q_n\}$ iff $\mathcal{M}$ reaches state q_n. Second, one can easily verify that the net $\mathcal{N}_\mathcal{M}$ is sequential.

We then combine the TPNs $\mathcal{N}_1$ from Figure 2.6 and $\mathcal{N}_\mathcal{M}$ as depicted on Figure 2.7 to obtain the TPN $\mathcal{N}_2$. It it then easy to observe the two following facts:

- for $\delta = 0$, the red place of $\mathcal{N}_1$ is never marked, and thus no token enters the net $\mathcal{N}_\mathcal{M}$
- for every positive δ, transition t_s becomes fireable, and thus the $\mathcal{N}_\mathcal{M}$ is "executed". In addition, as it is sequential, its execution is not altered by the presence of small enough perturbations (Lemma 5)

The results easily follow. □

Subclasses with decidability results

A robust translation from TPN to TA As robustness issues were first studied for timed automata, and several translations of TPN into TA exist in literature, it is natural to study which of these translations are compatible with robustness. A way to reduce robustness problems for TPNs to robustness problems for TA is to show that an existing timed bisimulation between TPN and its TA translation is preserved under perturbation.

Let us study the marking class timed automaton construction of [DDSS07]. The construction of [DDSS07] was only stated for TPN whose underlying Petri net (*i.e.* the Petri net obtained by ignoring the timing information in the given TPN) is bounded. We adapt the construction to a more general framework: we consider as input a TPN $\mathcal{N}$ which is not necessarily bounded and a finite set of markings M. The construction is then restricted to

the set M, and we can prove that it is correct for the set of behaviors of $\mathcal{N}$ which always remain within M. In the following, we will instantiate M depending on the context.

Theorem 13 ([1, 2])**.** *Let $\mathcal{N}$ be a TPN, M be a finite set of markings containing the initial marking of $\mathcal{N}$, and $\mathcal{A}_M$ be the marking timed automaton of $\mathcal{N}$ over M. We denote by $[\![\mathcal{N}_\delta]\!]_{|M}$ the restriction of $[\![\mathcal{N}_\delta]\!]$ to states (m, ν) such that $m \in M$. Then for all $\delta \in \mathbb{Q}_{\geq 0}$, we have $[\![\mathcal{N}_\delta]\!]_{|M} \approx [\![(\mathcal{A}_M)_\delta]\!]$.*

Robustly bounded TPNs By Theorem 12, we know that checking membership in the class of robustly bounded TPNs is undecidable. We present two decidable subclasses, as well as a semi-decision procedure for the whole class. We first consider the subclass of TPNs whose *underlying Petri net* is bounded:

Proposition 4. *The set of TPNs whose underlying net is bounded is a decidable subclass of robustly bounded TPNs. Further, for each net $\mathcal{N}$ of this class, one can construct a finite timed automaton $\mathcal{A}$ such that $[\![\mathcal{N}_\delta]\!] \approx [\![\mathcal{A}_\delta]\!]$ for all $\delta \geq 0$.*

The decidability follows from that of boundedness for (untimed) Petri nets [KM69]. The second part of the above proposition follows from Theorem 13.

We now exhibit another decidable subclass of robustly bounded TPNs whose underlying Petri nets can be unbounded. We first state the following important technical result:

Lemma 6. *Let $\mathcal{N}$ be a TPN, and M be a finite set of markings. Determining whether there exists $\delta > 0$ such that $\mathsf{reach}(\mathcal{N}_\delta) \subseteq M$ is decidable.*

Proof. Call $\widetilde{M} = M \cup \{m' \mid \exists m \in M, t \in T, m' = m - {}^\bullet t + t^\bullet\}$ the (finite) set of markings reachable from M in at most one-step in the underlying Petri net. Let $\mathcal{A}_{\widetilde{M}}$ be the marking timed automaton of $\mathcal{N}$ over $\widetilde{M}$, and let $\delta \geq 0$. We claim:

$$\mathsf{reach}(\mathcal{N}_\delta) \subseteq M \iff \mathsf{reach}((\mathcal{A}_{\widetilde{M}})_\delta) \subseteq M$$

The proof of this equivalence relies on Theorem 13.

Now, determining whether there exists $\delta > 0$ such that the right hand side of the previous equivalence holds is decidable thanks to Proposition 1. □

We consider the following subclass of bounded TPNs:

Definition 4. *A bounded TPN $\mathcal{N}$ is called Reach-Robust if $\mathsf{reach}(\mathcal{N}_\delta) = \mathsf{reach}(\mathcal{N})$ for some $\delta > 0$. We denote by* RR *the class of Reach-Robust TPNs.*

Checking membership in this class is decidable, *i.e.* given a bounded TPN $\mathcal{N}$ we can decide if there is a positive guard enlargement under which the set of reachable markings remains unchanged. This follows from Lemma 6 by taking $M = \mathsf{reach}(\mathcal{N})$:

Theorem 14 ([1, 2])**.** *RR is a decidable subclass of robustly bounded TPNs.*

We can now address properties of the general class of robustly bounded TPN. Again, this result follows from Lemma 6 and Theorem 13.

Lemma 7. *The set of robustly bounded TPNs is recursively enumerable. Moreover, given a robustly bounded TPN $\mathcal{N}$, we can build effectively a timed automaton $\mathcal{A}$ such that there exists $\delta_0 > 0$ for which, $\forall 0 \leq \delta \leq \delta_0$, $[\![\mathcal{N}_\delta]\!] \approx [\![\mathcal{A}_\delta]\!]$.*

This result allows us to transfer existing robustness results for timed automata to TPNs. We will illustrate the use of this property in the following section.

Untimed language robustness in TPNs We now consider the robust untimed language preservation problem, which was shown undecidable in general in Theorem 12. We show that for the subclass of *distinctly labeled bounded TPNs* (*i.e.* labels on transitions are all distinct, and different from ε) this problem becomes decidable.

Definition 5. *A bounded TPN $\mathcal{N}$ is called Language-Robust if $L(\mathcal{N}_\delta) = L(\mathcal{N})$ for some $\delta > 0$. We denote by LR the class of Language-Robust nets and by $LR_{\neq}$ (resp. $RR_{\neq}$) the subclass of LR (resp. RR) with distinct labeling.*

We first compare the class RR (for which checking membership is decidable by Theorem 14) with the class LR (where, as already noted, checking membership is undecidable by Theorem 12). We can then observe that:

Proposition 5. *(1) The classes RR and LR are incomparable w.r.t. set inclusion. (2) Further, the class $LR_{\neq}$ is strictly contained in the class $RR_{\neq}$.*

Finally, we show that the problem of robust untimed language preservation becomes decidable under the assumption of distinct labeling. This result uses Lemma 7 and Proposition 2 on timed automata.

Theorem 15 ([1, 2])**.** *The class $LR_{\neq}$ is decidable, i.e. checking if a distinctly labeled bounded TPN is in LR is decidable.*

Summary

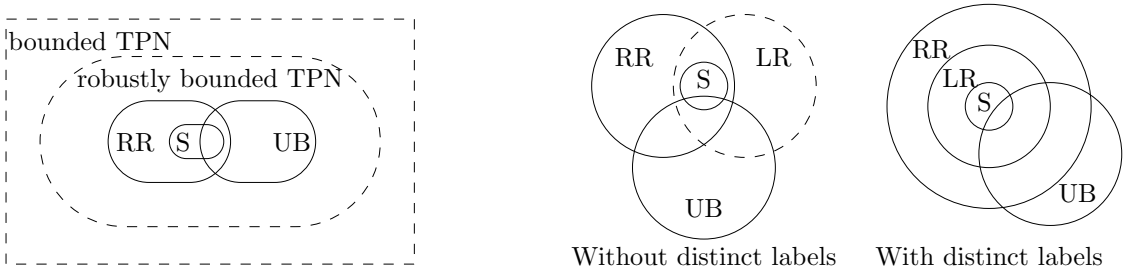


Figure 2.8: RR stands for reach-robust, LR for language-robust, UB for bounded underlying PNs, S for sequential bounded TPNs. Dotted lines represent that checking membership in the class is undecidable, while solid lines represent that it is decidable.

We have launched an investigation into robustness in time Petri nets with respect to guard enlargements. We transferred several positive results from the TA setting to TPNs and showed that some other problems become undecidable in TPNs due to unboundedness. We summarize our results in the diagram in Figure 2.8 which shows the relative expressiveness of the various classes and their decidability. In the table below, we give a more detailed picture of the decidability results for each problem and each class of TPNs considered. The leftmost column lists the problems addressed while the top row lists the subclasses of nets. A "yes" entry in a cell means that the problem considered at that row is decidable for the class considered in the column, "no" means that it is undecidable, and "g" means that a positive answer to the problem is always guaranteed for the considered class. For instance, robustness of subset-markings is guaranteed for Reach-robust nets.

Problems \ Classes	TPNs	Bounded	Rob. Bounded	RR	UB	S
Robust Boundedness	no	no	g	g	g	g
Robust language preservation (with distinct labels)	no	yes	yes	yes	yes	g
Robust language preservation (without distinct labels)	no	no	yes	yes	yes	g
Robustness of $\mathsf{reach}(\mathcal{N})$	no	yes	yes	g	yes	g

2.4 Perspectives

The different results we have presented offer natural immediate perspectives. For instance, regarding our results on MTL, one could be interested in extending the results of [BMS11] to our setting in order to derive a proof of our decidability results fully based on channel machines and valid for general timed automata.

The algorithm of [DK06] that we extended to a parametric setting has recently been lifted from flat timed automata to general timed automata in [KLMP14]. It seems thus promising to try to adapt this new algorithm to a parametric setting.

Last, our results on time Petri nets are mainly consequences of results proven on timed automata. A challenging research topic would thus be to identify classes of unbounded time Petri nets with decidable robust model checking problems, and also, in another direction, to consider these robustness issues directly on the non-interleaved semantics of time Petri nets.

In addition to these research leads, the main challenge for robustness analysis is probably the one of implementation. There is indeed a lack of tool allowing to check the robustness of a timed automaton, and this is actually due to the fact that most decidability results published so far rely on the region automaton construction. The only exceptions are the symbolic algorithms of [DK06, KLMP14], and they have led to the development of two tools. First, an implementation of the algorithm of [KLMP14] is presented in the same paper. Second, another tool has been published this year in [San15]. This tool implements a modification of the same algorithm: it performs a symbolic forward exploration of the timed automaton and accelerates less cycles than [KLMP14]. This results in a semi-decision procedure as no guarantee of termination is provided. The implementation relies on a simplification of the data structure of PLDBMs that considers small values of δ (hence, only the first affine map of a piecewise linear function is preserved), and thus allows addition to compute a positive admissible value of δ.

Chapter 3

Robust controller synthesis

Contents

We consider timed games played on a timed automaton involving two players, namely Controller and Environment. In such games, Controller usually suggests an action a and a delay $d \geq 0$, and Environment answers by choosing an edge enabled after d time units whose label is a. Using this definition, a winning strategy for Controller may require to execute an action after an *exact* delay of one time unit, or to execute actions faster and faster. In this chapter, we are interested in synthesizing robust strategies for Controller, *i.e.* tolerant to some imprecisions in delays, and thus amenable to implementation.

3.1 Robust timed games

Definitions

In order to define perturbations, and to capture the reactivity of a controller to these, we define the following parameterized timed game semantics. Intuitively, the parameterized timed game semantics of a timed automaton is a two-player game parameterized by $\delta > 0$, where Player 1, also called Controller chooses a delay $d > \delta$ and an action $a \in \Sigma$ such that every a-labeled enabled edge is such that its guard is satisfied after any delay in the set $d + [-\delta, \delta]$ (and there exists at least one such edge). Then, Player 2, also called Perturbator chooses an actual delay $d' \in d + [-\delta, \delta]$ after which the edge is taken, and chooses one of the enabled a-labeled edges. Hence, Controller is required to always suggest delays that satisfy the guards whatever the perturbations are.

Formally, given a timed automaton $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, \Sigma, E)$ and $\delta > 0$, we define the *parameterized timed game* of $\mathcal{A}$ w.r.t. δ as a two-player turn-based game $\mathcal{G}_\delta(\mathcal{A})$ between players Controller and Perturbator. The state space of $\mathcal{G}_\delta(\mathcal{A})$ is partitioned into $V_C \cup V_P$ where $V_C = \mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}$ belong to Controller, and $V_P = \mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0} \times \mathbb{R}_{\geq 0} \times \Sigma$ belong to Perturbator. The initial state is $(\ell_0, \vec{0}) \in V_C$. The transitions are defined in the natural way according to the intuition given above.

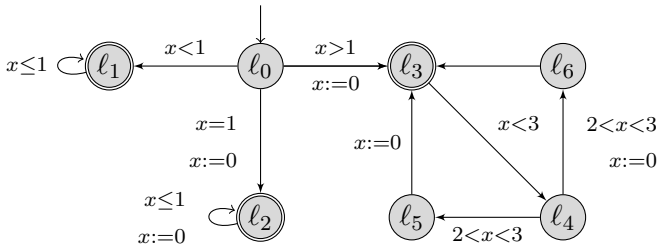


Figure 3.1: A robustly controllable timed automaton

A *play* of $\mathcal{G}_\delta(\mathcal{A})$ is a finite or infinite sequence $q_1 e_1 q_2 e_2 \ldots$ of states and transitions of $\mathcal{G}_\delta(\mathcal{A})$, with $q_1 = (\ell_0, \vec{0})$, where e_i is a transition from q_i to q_{i+1}. It is said to be *maximal* if it is infinite or cannot be extended. A *strategy* for Controller is a function that assigns to every non-maximal play ending in some $(\ell, \nu) \in V_C$, a pair (d, a) where $d > \delta$ and a is an action such that there is a transition from (ℓ, ν) to (ℓ, ν, d, a). A strategy for Perturbator is a function that assigns, to every play ending in (ℓ, ν, d, a), a state (ℓ', ν') such that there is a transition from the former to the latter state. A play ρ is *compatible* with a strategy f for Controller if for every prefix ρ' of ρ ending in V_C, the next transition along ρ after ρ' is given by f. We define similarly compatibility for Perturbator's strategies. A play naturally gives rise to a unique run, where the states are in V_C, and the delays and the edges are those chosen by Perturbator.

Robust timed game problem. Given $\delta > 0$, and a pair of strategies f, g, respectively for Controller and Perturbator, we denote ρ the unique maximal run that is compatible with both f and g. A *Büchi objective* is a subset of the locations of $\mathcal{A}$. A Controller's strategy f is winning for a Büchi objective B if for any Perturbator's strategy g the run ρ that is compatible with f and g is infinite and visits infinitely often a location of B. The *robust timed game problem* asks, for a timed automaton $\mathcal{A}$ and a Büchi objective B, if there exists $\delta > 0$ such that Controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$ for the objective B. When this holds, we say that Controller wins the robust timed game for $\mathcal{A}$, and otherwise that Perturbator does. Note that these games are determined since for each $\delta > 0$, the semantics is a timed game.

Example 7. *Consider first the (deterministic) timed automaton depicted on Figure 1.1, equipped with the Büchi objective $\{\ell_2\}$. This timed automaton is not robustly controllable. In fact,* Perturbator *can enforce in the game $\mathcal{G}_\delta(\mathcal{A})$ that the value of x be increased by δ at each arrival at ℓ_1, thus blocking the run eventually.*

Consider now the timed automaton depicted on Figure 3.1, this automaton is robustly controllable for the Büchi objective $\{\ell_1, \ell_2, \ell_3\}$. We assume that all transitions have the same label, hence the automaton is non-deterministic. The cycle around ℓ_1 cannot be taken forever, as value of x increases due to perturbations. The cycle around ℓ_2 can be taken forever, but Controller *cannot reach ℓ_2 due to the equality $x = 1$.* Controller*'s strategy is thus to loop forever around ℓ_3. This is possible as for both choices of* Perturbator *in location ℓ_4 to resolve non-determinism, clock x will be reset, and thus perturbations do not accumulate. If one of the two resets were absent,* Perturbator *could force the run to always take that branch, and would win the game.*

The main result of this section is the following theorem:

Theorem 16 ([22]). *The robust timed game problem is* ExpTime-c.

It is worth observing that the corresponding (non-robust) timed game problem is also ExpTime-c.

Related work This problem has been addressed in [CHP11] but for a fixed valued value of δ. In this case, it is possible to reduce the problem to standard timed games, and the parametric problem we solve here was left as an open challenging problem. More recently, a similar problem has been formalized in the theory of interfaces in [LLTW14]. Again, the problem is solved for a fixed valued of δ, avoiding thus the main difficulty.

The case of deterministic automata

If the timed automaton $\mathcal{A}$ defining the timed game is deterministic, Perturbator only perturbs the delays prescribed by Controller. This setting has been considered in [27], and can be illustrated for instance by the first example considered in Example 7, depicted on Figure 1.1. As we will see, solving this problem amounts to determining which cycles of the region automaton are safe for Controller, *i.e.* such that Controller is able to loop forever in this cycle whatever the perturbations of Perturbator. This is directly associated to the notion of convergence in cycles, which generalizes the Zeno behaviors, and that is illustrated with the timed automaton of Figure 1.2.(b).

Orbit graphs While standard timed games can be solved using the region automaton as abstraction, this is not the case for robust timed games, as illustrated with the timed automaton depicted on Figure 1.1. We introduce a refinement of the region automaton, based on the notion of folded orbit graph, that allows to identify more precisely the possible timed behaviors along an abstract run described by the region automaton. Although this is technical, we believe that this may be an important tool for several problems in timed automata.

A *vertex* of a region r is any point of $\bar{r} \cap \mathbb{N}^{\mathcal{C}}$. The set of vertices of r is denoted $\mathcal{V}(r)$. With any path π of the region automaton, we associate a $|\pi|$-partite labelled graph called the *orbit graph of* π [Pur00]. Intuitively, the orbit graph of a path gives the reachability relation between the vertices of the regions visited along the path. We define the *folded orbit graph* (FOG for short) $\Gamma(\pi)$ for any path π as a bipartite graph on node set $\{1\} \times \mathcal{V}(\mathsf{first}(\pi)) \cup \{2\} \times \mathcal{V}(\mathsf{last}(\pi))$ (if π is a cycle around $\mathsf{first}(\pi)$, then we only consider vertices $\mathcal{V}(\mathsf{first}(\pi))$). Intuitively, this graph is obtained as a compression of the previous one. As proven in [Pur00], it represents precisely the reachability relation between valuations along the path π. For instance, this relation is complete iff the FOG is.

A strongly connected component (SCC) of a graph is *initial* if it is not reachable from any other SCC. A *forgetful cycle* of $\mathcal{R}(\mathcal{A})$ is a cycle whose FOG is strongly connected. A cycle π is *aperiodic* if for all $k \geq 1$, π^k is forgetful. Note that there exist forgetful cycles that are not aperiodic [Sta12]. An example of a non-forgetful cycle is given in Figure 3.3. Figure 3.4 gives an example of forgetful (and aperiodic) cycle.

Forgetfulness was recently introduced in [BA11] for studying the entropy of timed languages. This was further studied in [Sta12] in the context of frequencies in timed automata, where aperiodicity was considered. Already in these works, forgetful cycles are used to discard

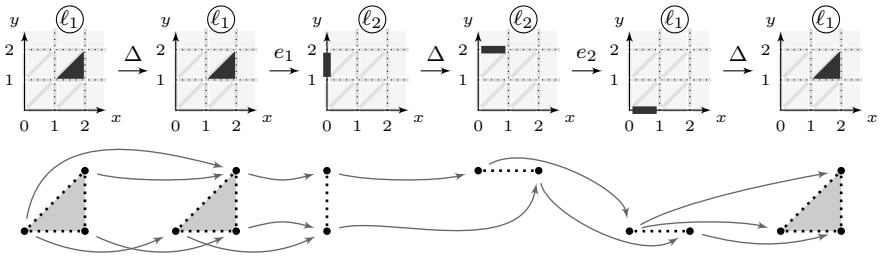


Figure 3.2: The orbit graph of a (cyclic) path in the region automaton of the automaton of Fig. 1.1.

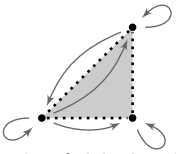


Figure 3.3: The folded orbit graph of the (non-forgetful) cycle of Fig. 3.2.

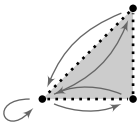


Figure 3.4: The folded orbit graph of a forgetful cycle.

convergent runs. As we show in the next lemma, these notions can be used to characterize the existence of robust strategies along infinite executions in the game semantics.

Lemma 8 ([27])**.** *Let $\mathcal{A}$ be a deterministic timed automaton, and π be a cycle of $\mathcal{R}(\mathcal{A})$. Then* Controller *has a winning strategy along this path iff π is aperiodic.*

π not aperiodic implies no winning strategy for Controller**.** Let us first suppose that π is not aperiodic and explain how to build a winning strategy σ_p for Perturbator in the game $\mathcal{G}_\delta(\mathcal{A})$ for every $\delta > 0$. We will formalize the intuition presented for the timed automaton of Figure 1.1 in Example 7.

Let us represent a valuation ν in some region r by its barycentric coordinates w.r.t. vertices of r: we write $\nu = \lambda_\nu \cdot \mathcal{V}(r)$. As proved by Puri, the folded orbit graph of a cycle π describes the reachability relation along π as follows: let $\nu, \nu' \in r$, we have $\nu \xrightarrow{\pi} \nu'$ iff there exists a stochastic matrix $P \leq M(\Gamma(\pi))$ such that $\lambda_{\nu'} = \lambda_\nu \cdot P$ (where $M(\Gamma(\pi))$ denotes the adjacency matrix of $\Gamma(\pi)$). Intuitively, the edges of $\Gamma(\pi)$ represent how the barycentric coordinates may flow when cycle π is followed.

Given a region r with set of vertices V, a subset $I \subseteq V$ and a valuation $\nu = \lambda_\nu \in r$, we define $L_I(\nu) = \sum_{v \in I} \lambda_\nu(v)$. The next lemma is established in [BA11] (I corresponds to an initial strongly connected component of the orbit graph):

Lemma 9. *Let π be a non-forgetful cycle with vertices V. There exists a subset $I \subsetneq V$ such that for every $\nu \xrightarrow{\pi} \nu'$, we have $L_I(\nu') \leq L_I(\nu)$.*

This lemma states that the sum of the coefficients associated with the subset I decreases when cycle π is followed. We show in [27] that in the enlarged semantics, it is possible to strengthen this lemma by proving that this sum (strictly) decreases by a fixed amount (depending on δ).

Lemma 10. *Let π be a non-forgetful cycle with vertices V and $\delta > 0$. There exists a subset $I \subsetneq V$, and a strategy σ_p for* Perturbator *in the game $\mathcal{G}_\delta(\mathcal{A})$ such that for every play $\nu \xrightarrow{\pi} \nu'$ compatible with σ_p, we have $L_I(\nu') \leq L_I(\nu) - \varepsilon^2/2$ where $\varepsilon = \frac{\delta}{2(|\mathcal{C}|+1)}$.*

An immediate consequence of this lemma is that this cycle cannot be iterated forever, and thus the run is eventually blocking, or leaves the cycle, hence Controller does not have a winning strategy along this cycle.

π aperiodic implies the existence of a winning strategy for Controller. Suppose now that π is aperiodic. We can prove that for n large enough, the FOG of π^n is complete. We then show that along a cycle whose FOG is complete, Controller has a winning strategy. We use the standard tool of so-called controllable predecessors of a set of states X along a path π, denoted $\mathsf{CPre}_\pi(X)$, defined as those states from which Controller has a strategy to reach X by following the path π (*i.e.* a sequence of delays along path π). It is well-known that if X is given as a DBM, then this operator can be implemented by means of DBMs.

We lift this operator to the game $\mathcal{G}_\delta(\mathcal{A})$, and denote it by $\mathsf{CPre}^\delta_\pi(X)$. Here, Controller has a strategy to reach X by following the path π against any strategy of Perturbator. In order to study this parametric operator, we use shrunk DBMs (see [SBM14]) that allow us to study parametric restrictions of a DBM. Indeed, unlike Section 2.2 in which, during a forward analysis in the enlarged semantics, we studied parametric enlargements of DBMs, we consider now a parametric shrinking of DBMs. Using the fact that the FOG of the cycle is complete, we manage to exhibit a non-empty set of states s such that $s \subseteq \mathsf{CPre}^\delta_\pi(s)$, for small enough δ. This yields a winning strategy for Controller in the game $\mathcal{G}_\delta(\mathcal{A})$ for small enough δ, from states in s.

Observe also that this approach allows to build a concrete value of δ and an actual winning strategy for Controller in $\mathcal{G}_\delta(\mathcal{A})$.

Combining Lemma 8 with the fact that looking for a reachable cycle in the region automaton whose FOG is aperiodic can be done in PSPACE, we obtain:

Theorem 17 ([27])**.** *The robust timed game problem for deterministic timed automata is* PSPACE-C.

General case

Thanks to the results presented in the previous subsection, we know precisely which cycles are winning for Controller. In the general case Perturbator may also derive from a given cycle when he resolves non-determinism.

A game in the region automaton We define an appropriate abstraction (as a finite turn-based two-player game) based on region automata in order to characterize winning in the robust timed game. We therefore define, on top of the usual region construction, a complex winning condition $\mathcal{W}$ characterizing accepting runs *along aperiodic cycles*.

We fix a timed automaton $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, \Sigma, E)$ and a Büchi condition ϕ. We define a two-player turn-based game played on the region automaton $\mathcal{R}(\mathcal{A})$. In this game, Controller's strategy consists in choosing actions, while Perturbator's strategy consists in resolving non-determinism. Given a finite-memory strategy σ, we denote by $\mathcal{R}(\mathcal{A})[\sigma]$ the automaton obtained under strategy σ.

Winning condition on $\mathcal{R}(\mathcal{A})$ We define set $\mathcal{W}$ of winning plays in the game $\mathcal{R}(\mathcal{A})$: an infinite play is winning iff the following two conditions are satisfied: 1) an accepting state in ϕ is visited infinitely often 2) disjoint finite factors with complete folded orbit graphs are visited infinitely often.

Proposition 6. *The game $(\mathcal{R}(\mathcal{A}), \mathcal{W})$ is determined, admits finite-memory strategies for both players, and wining strategies can be computed in* ExpTime.

The above proposition is proved by showing that condition 2) of $\mathcal{W}$ can be rewritten as a Büchi condition: the set of folded orbit graphs constitutes a finite monoid (of exponential size) which can be used to build a Büchi automaton encoding condition 2). Using a product construction for Büchi automata, one can define a Büchi game of exponential size where winning for any player is equivalent to winning in $(\mathcal{R}(\mathcal{A}), \mathcal{W})$.

We introduce two conditions for Perturbator and Controller which are used to prove that condition $\mathcal{W}$ characterizes winning in the robust timed game.

$\mathcal{C}_P$: there exists a finite-memory strategy τ for Perturbator such that no cycle in $\mathcal{R}(\mathcal{A})[\tau]$ reachable from the initial state is winning aperiodic.

$\mathcal{C}_C$: there exists a finite-memory strategy σ for Controller such that every cycle in $\mathcal{R}(\mathcal{A})[\sigma]$ reachable from the initial state is winning aperiodic.

Intuitively, determinacy allows us to write that either all cycles are aperiodic, or none is, respectively under each player's winning strategies. We prove that these properties are sufficient and necessary for respective players to win $(\mathcal{R}(\mathcal{A}), \mathcal{W})$:

Lemma 11. *The winning condition $\mathcal{W}$ is equivalent to $\mathcal{C}_P$ and $\mathcal{C}_C$:*

1. Perturbator *wins the game $(\mathcal{R}(\mathcal{A}), \mathcal{W})$ iff property $\mathcal{C}_P$ holds.*

2. Controller *wins the game $(\mathcal{R}(\mathcal{A}), \mathcal{W})$ iff property $\mathcal{C}_C$ holds.*

In both cases, a winning strategy for $\mathcal{W}$ is also a witness for $\mathcal{C}_C$ (resp. $\mathcal{C}_P$).

The proof is obtained by the following facts: finite-memory strategies are sufficient to win the game $(\mathcal{R}(\mathcal{A}), \mathcal{W})$, thanks to the previous proposition; given a folded orbit graph γ, there exists n such that γ^n is complete iff γ is aperiodic; last, the concatenation of a complete FOG with an arbitrary FOG is complete.

Solving the robust timed game We explain now why condition $\mathcal{C}_P$ (resp. $\mathcal{C}_C$) is sufficient to witness the existence of a winning strategy in the robust timed game for Perturbator (resp. Controller). By Lemma 11, the robust timed game problem is then reduced to $(\mathcal{R}(\mathcal{A}), \mathcal{W})$ and we obtain:

Theorem 18 ([22])**.** *Let $\mathcal{A}$ be a timed automaton with a Büchi condition. Then,* Controller *wins the robust timed game for $\mathcal{A}$ iff he wins the game $(\mathcal{R}(\mathcal{A}), \mathcal{W})$.*

Sketch. First, we observe that under condition $\mathcal{C}_P$, Perturbator wins the robust timed game. We use the following observations. Once one fixes a strategy for Perturbator satisfying $\mathcal{C}_P$, intuitively, one obtains a deterministic timed automaton such that all accepting cycles are non-aperiodic. We can thus conclude by Lemma 8.

Suppose now that condition $\mathcal{C}_C$ holds, and let us show that Controller wins the robust timed game. This is the most difficult part of the proof. By hypothesis, we assume now that every reachable cycle in the region automaton is winning aperiodic (the finite memory strategy can be encoded in the automaton). Thanks to Lemma 8, we know that Controller can win if he can follow a *fixed* aperiodic cycle. However, due to non determinism, Perturbator

can switch between different cycles. We consider the unfolding of $\mathcal{R}(\mathcal{A})$ of depth N. We prove that by considering N large enough, one can ensure that every branch has a complete FOG. Second, we prove that winning strategies along fixed branches can be combined in a global winning strategy. This part is technical and uses the notion of shrunk DBMs as a representation of the winning strategy of Controller. □

By Proposition 6, the robust timed game can be solved in ExpTime. In addition, we prove in [22] that when Controller wins the robust timed game, one can also compute $\delta > 0$ and a concrete winning strategy in $\mathcal{G}_\delta(\mathcal{A})$.

3.2 Stochastic adversary

In some systems, considering the environment as a completely adversarial opponent is a too strong assumption. We thus consider stochastic environments by defining two semantics as probabilistic variants of the robust timed games. The first one is the *stochastic game semantics* where Perturbator only resolves the non-determinism in actions, but the perturbations are chosen independently and uniformly at random in the interval $[-\delta, \delta]$. The second semantics is the *Markov decision process (MDP) semantics*, where the non-determinism is also resolved by a uniform distribution on the edges, and there is no player Perturbator.

Stochastic game semantics

Formally, given $\delta > 0$, the state space is partitioned into $V_C \cup V_P$ as previously. At each step, Controller picks a delay $d \geq \delta$, and an action a such that for every edge $e = (\ell, g, a, R, \ell')$ such that $\nu + d \models g$, we have $\nu + d + \varepsilon \models g$ for all $\varepsilon \in [-\delta, \delta]$, and there exists at least one such edge e. Perturbator then chooses an edge e with label a, and a perturbation $\varepsilon \in [-\delta, \delta]$ is chosen independently and uniformly at random. The next state is determined by delaying $d + \varepsilon$ and taking the edge e. To ensure that probability measures exist, we restrict to *measurable strategies*.

In this semantics, we are interested in deciding whether Controller can ensure a given Büchi objective almost surely, for some $\delta > 0$. It turns out that the same characterization as in Theorem 18 holds in the probabilistic case.

Theorem 19 ([22]). *It is* ExpTime-c *to decide whether for some $\delta > 0$,* Controller *has a strategy achieving a given Büchi objective almost surely in the stochastic game semantics. Moreover, if $\mathcal{C}_C$ holds then* Controller *wins almost-surely; if $\mathcal{C}_P$ holds then* Perturbator *wins almost-surely.*

We claim that the characterization of the (non-stochastic) robust timed game presented in the previous section and given by the game $(\mathcal{R}(\mathcal{A}), \mathcal{W})$ also characterizes winning in this stochastic framework. If $\mathcal{C}_C$ holds then Controller wins surely (hence almost-surely). Otherwise, by Lemma 11, $\mathcal{C}_P$ holds. As in the proof of Theorem 18, once one fixes a strategy for Perturbator satisfying $\mathcal{C}_P$, one obtains a deterministic timed automaton such that all accepting cycles are non-aperiodic. To conclude, we adapt Lemma 10 to our stochastic setting by showing that for any $\delta > 0$, there exists an integer n (depending on δ) such that with probability 1, every play along a non-forgetful cycle has length at most n. Intuitively, each time the cycle is iterated, the clock valuation goes strictly closer to the border of the region with positive probability.

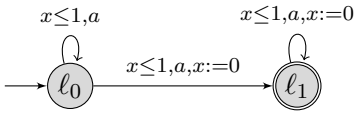


Figure 3.5: This automaton is losing in the MDP semantics for the almost-sure winning but winning under the same semantics for the limit-sure winning. In fact, a blocking state (ℓ_0, x) with $x > 1 - \delta$ is reachable with positive probability for any δ.

The previous theorem is a powerful result showing a strong distinction between robust and non-robust timed games: in the first case, a controller that ensures the specification almost surely can be computed, while in non-robust timed games, *any* controller will fail *almost surely*. Thus, while in previous works on robustness in timed automata (e.g. [Pur00]) the emphasis was on additional behaviors that might appear in the worst-case due to the accumulation of perturbations, we show that in our setting, this is inevitable. Note that this also shows that limit-sure winning (see next section) is equivalent to almost-sure winning.

Markov decision process semantics

The *Markov decision process semantics* consists in choosing both the perturbations, and the edges uniformly at random (and independently). Formally, it consists in restricting Perturbator to choose all possible edges uniformly at random in the stochastic game semantics. We denote by $\mathcal{G}_\delta^{\mathsf{MDP}}(\mathcal{A})$ the resulting game, and $\mathbb{P}^\sigma_{\mathcal{G}_\delta^{\mathsf{MDP}}(\mathcal{A}),s}$ the probability measure on $\mathsf{Runs}(\mathcal{A}, s)$ under strategy σ.

For a given timed Büchi automaton, denote ϕ the set of accepting runs. We are interested in the two following problems: (we let $s_0 = (\ell_0, \vec{0})$)

Almost-sure winning: does there exist $\delta > 0$ and a strategy σ for Controller such that $\mathbb{P}^\sigma_{\mathcal{G}_\delta^{\mathsf{MDP}}(\mathcal{A}),s_0}(\phi) = 1$?

Limit-sure winning: does there exist, for every $0 \leq \varepsilon \leq 1$, a perturbation upper bound δ, and a strategy σ for Controller such that $\mathbb{P}^\sigma_{\mathcal{G}_\delta^{\mathsf{MDP}}(\mathcal{A}),s_0}(\phi) \geq 1 - \varepsilon$?

Observe that if almost-sure winning cannot be ensured, then limit-sure winning still has a concrete interpretation in terms of controller synthesis: given a quantitative constraint on the quality of the controller, what should be the precision on clocks measurements to be able to synthesize a correct controller? Consider the timed automaton depicted on Figure 3.5. It is easy to see that Controller loses the (non-stochastic) robust game, the stochastic game and in the MDP semantics with almost-sure condition, but he wins in the MDP semantics with limit-sure condition.

Theorem 20 ([22])**.** *It is* ExpTime-c *to decide whether* Controller *wins almost-surely (resp. limit-surely) in the MDP semantics of a timed Büchi automaton.*

To prove this theorem, we will define decidable characterizations on $\mathcal{R}(\mathcal{A})$ which we will see as a finite Markov decision process. In this MDP, the non-determinism of actions is resolved according to a uniform distribution. Given a strategy $\hat{\sigma}$ for Controller and a state v, we denote by $\mathbb{P}^{\hat{\sigma}}_{\mathcal{R}(\mathcal{A}),v}$ the resulting measure on $\mathsf{Runs}(\mathcal{R}(\mathcal{A}), v)$. The initial state of $\mathcal{R}(\mathcal{A})$ is v_0. We will use well-known notions about finite MDPs; we refer to [Put94].

Almost-sure winning We introduce the winning condition $\mathcal{W}'$: Controller's strategy $\hat{\sigma}$ in $\mathcal{R}(\mathcal{A})$ is winning in state v iff $\mathbb{P}^{\hat{\sigma}}_{\mathcal{R}(\mathcal{A}),v}(\phi) = 1$ and all runs in $\mathsf{Runs}(\mathcal{R}(\mathcal{A})[\hat{\sigma}], v)$ contain infinitely many disjoint factors whose FOGs are complete. Observe that this combines an almost-sure requirement with a sure requirement. This winning condition is our characterization for almost-sure winning:

Proposition 7 ([22]). Controller *wins almost-surely in the MDP semantics of a timed Büchi automaton $\mathcal{A}$ iff* Controller *wins the game $(\mathcal{R}(\mathcal{A}), \mathcal{W}')$ in v_0.*

Intuitively, the first condition is required to ensure winning almost-surely, and the second condition allows to forbid blocking behaviors. Notice the resemblance with condition $\mathcal{W}$; the difference is that ϕ only needs to be ensured almost-surely rather than surely. We prove the decidability of this condition.

Lemma 12 ([22]). *The game $(\mathcal{R}(\mathcal{A}), \mathcal{W}')$ admits finite-memory strategies, and winning strategies can be computed in* ExpTime.

Limit-sure winning As illustrated in Figure. 3.5, it is possible, for any $\varepsilon > 0$, to choose the parameter $\delta > 0$ small enough to ensure a winning probability of at least $1 - \varepsilon$. The idea is that in such cases one can ensure reaching the set of almost-sure winning states with arbitrarily high probability, although the run can still be blocked with small probability before reaching this set.

To characterize limit-sure winning, we define condition $\mathcal{W}''$ as follows. If Win$'$ denotes the set of winning states for Controller in the game $(\mathcal{R}(\mathcal{A}), \mathcal{W}')$, then $\mathcal{W}''$ is defined as the set of states from which one can almost surely reach Win$'$.

Proposition 8 ([22]). Controller *wins limit-surely in the MDP semantics of a timed Büchi automaton $\mathcal{A}$ from s_0 iff* Controller *wins the game $(\mathcal{R}(\mathcal{A}), \mathcal{W}'')$ in v_0.*

The proof of this proposition relies on the following lemma, and uses techniques similar as those introduced to prove Proposition 7.

Lemma 13 ([22]). *The game $(\mathcal{R}(\mathcal{A}), \mathcal{W}'')$ admits finite-memory strategies, and winning strategies can be computed in* ExpTime.

3.3 Perspectives

A first natural perspective of the results presented in this chapter is their extension to more expressive timed games. It may indeed be useful for modelization purposes to allow the environment to play some uncontrollable edges. We believe that the tools we developed to handle non determinism will be useful to analyse such games. Another direction is the study of concurrent timed games, which are more symmetric. It is not clear yet what would be the good notion of robustness for these games.

As we pointed out in the perspectives of Chapter 2, in order to be able to apply our algorithms to case studies, we need to develop a prototype implementation. However, our approach relies on the region automaton construction, and it is well-known that this construction is not amenable to implementation. An important objective is thus to develop symbolic algorithms, based on zones, to solve the robust timed game problem. This is a difficult task as the notion of aperiodic path heavily relies on regions. Instead, one could try to develop

backward algorithms manipulating shrunk DBMs, and obtain this way good heuristics. We could also try to adapt the OTFUR algorithm used in UppAal Tiga for timed games analysis.

On a more theoretical side, our results rely on the notion of forgetful and aperiodic cycle. These notions have been introduced for the analysis of the entropy of deterministic timed automata. In order to handle non-determinism, we have shown how to use these notions for branching executions, and our results may thus be useful in the context of entropy.

Last, I would like to study the connections that may exist between our results and problems of discretization of timed automata. In this context, some authors have been interested in determining whether there exists a sampling rate under which the language is non-empty (with some Büchi accepting condition) [KP05]. The proofs of the results presented in this Chapter suggest that it could be possible to build a link between the existence of robust strategies and the existence of a sampling rate under which the language is non-empty. This would also allow to derive an original decision procedure looking for such a sampling rate using a reduction to SAT solvers, as in [NMA+02] for instance.

Part II

Analysis of transformations

Chapter 4

Preliminaries on transductions

Contents

The theory of regular languages constitutes a cornerstone in theoretical computer science. Initially studied on languages of finite words, it has since been extended in numerous directions, including finite and infinite trees.

Another natural extension is moving from languages to relations. This is precisely the purpose of transductions. We present in this section different fundamental models of transducers that allow to define string-to-string transductions, together with their main properties.

4.1 Transductions

Words/strings and languages Given a finite alphabet Σ, we denote by Σ^* the set of finite words over Σ, and by ε the empty word. These objects may also be called (finite) strings. A *language* over Σ is a set $L \subseteq \Sigma^*$.

The length of a word $u \in \Sigma^*$ is its number of symbols, denoted by $|u|$. For all $i \in \{1, \dots, |u|\}$, we denote by $u[i]$ the i-th letter of u. Given $1 \leq i \leq j \leq |u|$, we denote by $u[i..j]$ the word $u[i]u[i+1]\dots u[j]$ and by $u[j..i]$ the word $u[j]u[j-1]\dots u[i]$.

We say that $v \in \Sigma^*$ is a *prefix* (respectively a *suffix*, *factor*) of u if there exist $u_2 \in \Sigma^*$ (respectively $u_1 \in \Sigma^*$, $u_1, u_2 \in \Sigma^*$) such that $u = vu_2$ (respectively $u = u_1 v$, $u = u_1 v u_2$). If v is a prefix of u, we may write $v \preceq u$. Given two words u, v, we denote by $\mathsf{lcp}(u, v)$ their longest common prefix. Given two words $u, v \in \Sigma^*$, we define $\mathsf{delay}(u, v) = (u', v')$, where u', v' are two words such that $u = \mathsf{lcp}(u, v)u'$ and $v = \mathsf{lcp}(u, v)v'$. We also define the distance $d(u, v)$ between two words u, v as $d(u, v) = |u| + |v| - 2.|\mathsf{lcp}(u, v)|$. Observe that if we have $\mathsf{delay}(u, v) = (u', v')$, then we also have $d(u, v) = |u'| + |v'|$.

By $\overline{u}$ we denote the *mirror* of u, *i.e.* the word of length $|u|$ such that $\overline{u}[i] = u[|u| - i + 1]$ for all $1 \leq i \leq |u|$.

Transductions Let Σ and Δ be two alphabets. A *transduction* R from Σ^* to Δ^* is a binary relation between words of Σ^* and words of Δ^*, i.e. $R \subseteq \Sigma^* \times \Delta^*$. We say that v is a *transduction*, or an *image*, of u by R whenever $(u, v) \in R$. The set of all images of u by R is denoted $R(u) = \{v \in \Delta^* \mid (u, v) \in R\}$. We extend this notation to sets of words, let $L \subseteq \Sigma^*$, we denote by $R(L) = \cup_{u \in L} R(u)$ the set of all transductions by R of words in L. The *domain* of R, denoted by $\mathrm{dom}(R)$, is the set $\{u \in \Sigma^* \mid \exists v \in \Delta^* : (u, v) \in R\}$. The *range* of R, denoted by $\mathrm{range}(R)$, is the set $\{v \in \Delta^* \mid \exists u \in \Sigma^* : (u, v) \in R\}$.

Transductions are subsets of $\Sigma^* \times \Delta^*$, as such, the set-related operations do apply to transductions. For example, the intersection of two transductions $R_1 \cap R_2 = \{(u, v) \mid (u, v) \in R_1 \wedge (u, v) \in R_2\}$ is the standard set intersection. The *inverse* transduction of a transduction R from Σ^* to Δ^* is the relation $R^{-1} = \{(u, v) \mid (v, u) \in R\}$ from Δ^* to Σ^*. The *composition* of a transduction R_1 from ${\Sigma_1}^*$ to ${\Sigma_2}^*$ and R_2 from ${\Sigma_2}^*$ to Δ^* is the transduction $R_2 \circ R_1$ from ${\Sigma_1}^*$ to Δ^* such that $R_2 \circ R_1 = \{(u, v) \mid \exists w \in {\Sigma_2}^* : (u, w) \in R_1 \wedge (w, v) \in R_2\}$. The *restriction* of a transduction R to a language L is a transduction, denoted by $R_{|L}$, such that its domain is the domain of R restricted to words that belong to L, and the image of a word is the image of this word by R, *i.e.* $R_{|L} = \{(u, v) \mid (u, v) \in R \wedge u \in L\}$.

We say that a transduction R is *functional* if it defines a function, *i.e.* for every $u \in \mathrm{dom}(R)$, there is exactly one v such that $(u, v) \in R$. Given a natural number k, we say it is *k-valued* if, for every $u \in \mathrm{dom}(R)$, the size of $R(u)$ is at most k. Last, we say it is *finite-valued* if there exists k such that it is k-valued.

Decision Problems Viewing transductions as a tool to perform static analysis of programs manipulating strings, or trees, it is important to prove that we can decide some properties for classes of transductions under interest. The decision problems we will consider for transductions are the following. We consider a class of transductions $\mathcal{R}$ from Σ^* to Δ^*.

- Membership: given $R \in \mathcal{R}$ and $(u, v) \in \Sigma^* \times \Delta^*$, does $(u, v) \in R$ hold?
- Emptiness: given $R \in \mathcal{R}$, does there exist $(u, v) \in \Sigma^* \times \Delta^*$ such that $(u, v) \in R$ holds?
- Equivalence: given $R_1, R_2 \in \mathcal{R}$, do we have $R_1 = R_2$?
- Type Checking against a class of languages $\mathcal{L}$: given two languages $L_1 \in \mathcal{L}$ and $L_2 \in \mathcal{L}$, do we have $R(L_1) \subseteq L_2$?

The problem of type checking is in particular useful in database applications, as it allows for instance to verify that the update of the database does not alter the integrity of the database w.r.t. some property expressed as a regular language. It is also useful for instance when considering transformations of XML documents, which should always be compliant w.r.t. to a DTD.

It is worth observing that the type-checking problem can be reduced to the emptiness problem, provided that the class of transductions $\mathcal{R}$ is closed under composition, and the class of languages $\mathcal{L}$ is closed under complement. Indeed, one can then use the following equivalence:

$$R(L_1) \subseteq L_2 \iff \mathrm{dom}(Id_{\overline{L_2}} \circ R \circ Id_{L_1}) = \emptyset$$

where Id_L is the identity mapping defined exactly on the language L, and $\overline{L}$ denotes the complement of L.

4.2 Finite-state transducers

Automata A *non-deterministic two-way finite state automaton*[1] (2NFA) over a finite alphabet Σ is a tuple $A = (Q, q_0, F, \Delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of final states, and Δ is the transition relation, of type $\Delta \subseteq Q \times \Sigma \times Q \times \{+1, -1\}$. It is *deterministic* if for all $(p, a) \in Q \times \Sigma$, there is at most one pair $(q, m) \in Q \times \{+1, -1\}$ such that $(p, a, q, m) \in \Delta$. In order to see how words are evaluated by A, it is convenient to see the input as a right-infinite input tape containing the word (starting at the first cell) followed by blank symbols. Initially the head of A is on the first cell in state q_0 (the cell at position 1). When A reads an input symbol, depending on the transitions in Δ, its head moves to the left (-1) if the head was not in the first cell, or to the right ($+1$) and changes its state. A stops as soon as it reaches a blank symbol (therefore at the right of the input word), and the word is accepted if the current state is final.

A *configuration* of A is a pair $(q, i) \in Q \times (\mathbb{N} \setminus \{0\})$ where q is a state and i is a position on the input tape. A *run* ρ of A is a finite sequence of configurations. The run $\rho = (p_1, i_1) \dots (p_m, i_m)$ is a run on an input word $u \in \Sigma^*$ of length n if $p_1 = q_0$, $i_1 = 1$, $i_m \leq n + 1$, and for all $k \in \{1, \dots, m-1\}$, $1 \leq i_k \leq n$ and $(p_k, u[i_k], p_{k+1}, i_{k+1} - i_k) \in \Delta$. It is *accepting* if $i_m = n+1$ and $p_m \in F$. This means that for all the configurations but the last one, the reading head should be on a letter of the input word ($1 \leq i_k \leq n$ for all $k \leq m - 1$). For the run to be accepting, the last configuration must correspond to the reading head being immediately after the end of the input word ($i_m = n + 1$). The language of a 2NFA A, denoted by $L(A)$, is the set of words u such that there exists an accepting run of A on u.

A *non-deterministic (one-way) finite state automaton* (NFA) is a 2NFA such that $\Delta \subseteq Q \times \Sigma \times Q \times \{+1\}$, therefore we will often see Δ as a subset of $Q \times \Sigma \times Q$. Any 2NFA is effectively equivalent to an NFA (w.r.t. the accepted language). It was first proved by Rabin and Scott, and independently by Shepherdson [RS59, She59]. We denote by 2DFA (resp. DFA) the class of deterministic two-way finite state automata (resp. deterministic one-way finite state automata). Clearly, deterministic machines have a single computation on any input word. We way a 2NFA is *unambiguous* if for every input word u, there is at most one accepting run on u.

Transducers *Non-deterministic two-way finite state transducers* (2NFT) over Σ extend NFA with a one-way left-to-right output tape. They are defined as 2NFA except that the transition relation Δ is extended with outputs: $\Delta \subseteq Q \times \Sigma \times \Sigma^* \times Q \times \{-1, +1\}$. If a transition (q, a, v, q', m) is fired on a letter a, the word v is appended to the right of the output tape and the transducer goes to state q'. Wlog we assume that for all $p, q \in Q$, $a \in \Sigma$ and $m \in \{+1, -1\}$, there exists at most one $v \in \Sigma^*$ such that $(p, a, v, q, m) \in \Delta$. We also denote v by $\mathrm{out}(p, a, q, m)$.

A run of a 2NFT is a run of its underlying automaton, i.e. the 2NFA obtained by ignoring the output. A run ρ may be simultaneously a run on a word u and on a word $u' \neq u$. However, when the underlying input word is given, there is a unique sequence of transitions associated with ρ. Given a 2NFT T, an input word $u \in \Sigma^*$ and a run $\rho = (p_1, i_1) \dots (p_m, i_m)$ of T on u, the output of ρ on u, denoted by $\mathrm{out}^u(\rho)$, is the word obtained by concatenating the outputs of the transitions followed by ρ, i.e. $\mathrm{out}^u(\rho) =$

[1] We follow the definition of Vardi [Var89], but without stay transitions. This is without loss of generality though.

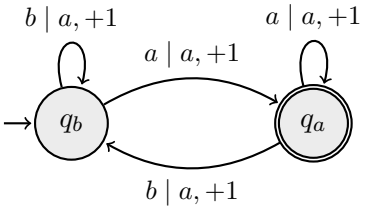


Figure 4.1: The transducer T_0 realizing the transduction R_0.

$\mathrm{out}(p_1, u[i_1], p_2, i_2{-}i_1) \cdots \mathrm{out}(p_{m-1}, u[i_{m-1}], p_m, i_m{-}i_{m-1})$. If ρ contains a single configuration, we let $\mathrm{out}^u(\rho) = \varepsilon$. When the underlying input word u is clear from the context, we may omit the exponent u. The transduction defined by T is the relation $[\![T]\!] = \{(u, \mathrm{out}^u(\rho)) \mid \rho \text{ is an accepting run of } T \text{ on } u\}$. We may often just write T when it is clear from the context.

A 2NFT T is *functional* if the transduction it defines is functional. The class of functional 2NFT is denoted by f2NFT. The class of functions realized by f2NFT is called the class of *regular functions*. A *deterministic two-way finite state transducer* (2DFT) is a 2NFT whose underlying input automaton is deterministic. Note that 2DFT are always functional, as there is at most one accepting run per input word. Note also that thanks to determinism, a configuration (q, i) cannot occur twice in an accepting run of a 2DFT T (otherwise the run is infinite). As a consequence, functions realized by 2DFT are linearly bounded, *i.e.* there exists an integer K such that for every $(u, v) \in [\![T]\!]$, we have $|v| \leq K|u|$.

A *non-deterministic (one-way) finite state transducer* (NFT) is a 2NFT whose underlying automaton is an NFA[2]. It is deterministic, or sequential, (written DFT) if the underlying automaton is a DFA. We will also consider the class of functional one-way transducers, denoted by fNFT. The class of functions realized by fNFT is called the class of *rational functions*.

Last, we say that a 2NFT T is unambiguous if its underlying automaton is unambiguous. When this is the case, we have that T is functional. A well-known result due to [EM65] states that every functional NFT can be transformed into an equivalent unambiguous NFT.

Example 8. *Let $\Sigma = \{a, b\}$ and $\# \notin \Sigma$, and consider the transductions*

1. $R_0 = \{(u, a^{|u|}) \mid u \in \Sigma^+, u[|u|] = a\}$
2. $R_1 = \{(u, b^{|u|}) \mid u \in \Sigma^+, u[|u|] = b\} \cup R_0$
3. $R_2 = \{(\#u\#, \#\overline{u}\#) \mid u \in \Sigma^*\}$.

R_0 is DFT*-definable: it suffices to replace each letter by a and to accept only if the last letter is a. Therefore it can be defined by the* DFT $T_0{=}(\{q_a, q_b\}, q_b, \{q_a\}, \{(q_x, y, a, q_y) \mid x, y \in \Sigma\})$. *This transducer is depicted on Figure 4.1.*

R_1 is fNFT*-definable but not* DFT*-definable: one can construct an* NFT T_1 *as follows. In its initial state, it non-deterministically guesses the last letter of the input word and, accordingly, goes to the left (states q_2, q_4) or to the right (states q_1, q_3). It also guesses whether the current letter is or not the last letter of the word. Even if R_1 is functional, it is*

[2]This definition implies that there is no ε-transitions that can produce outputs, which may cause the image of an input word to be an infinite language. Those NFT are sometimes called *real-time* in the literature.

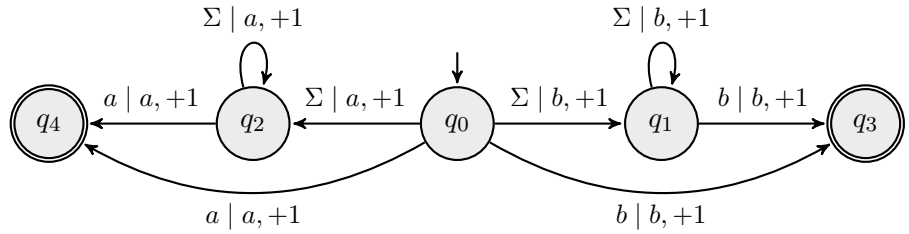


Figure 4.2: The transducer T_1 realizing the transduction R_1.

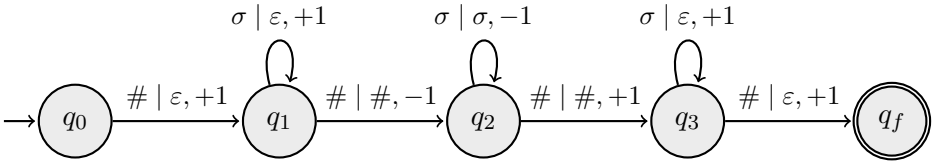


Figure 4.3: The transducer T_2 realizing the transduction R_2.

not DFT*-definable, as the transformation depends on the property of the last letter, which can be arbitrarily far away from the beginning of the string. This intuition will be made formal later in this section.*

R_2 is 2DFT*-definable: it suffices to go to the end of the word by producing ε each time a letter is read, to go back to the beginning while copying each input letter, and return to the end without outputting anything, and to accept. Hence it is defined by $T_2 = (\{q_0, q_1, q_2, q_3, q_f\}, q_0, \{q_f\}, \delta_2)$ where states q_1, q_2, q_3 denote passes, and δ_2 is made of the transitions $(q_0, \#, \varepsilon, q_1, +1)$, $(q_1, x{\in}\Sigma, \varepsilon, q_1, +1)$ (during the first pass, move to the right), $(q_1, \#, \varepsilon, q_2, -1)$, $(q_2, x{\in}\Sigma, x, q_2, -1)$, $(q_2, \#, \#, q_3, +1)$, $(q_3, x{\in}\Sigma, \varepsilon, q_3, +1)$, $(q_3, \#, \#, q_f, +1)$. This transducer is depicted on Figure 4.3.*

Deterministic and functional transducers Functions play an important role among relations and, as we will see later, they enjoy better decidability properties. This motivates the study of classes of transducers which are functional. It is trivial that deterministic transducers are functional. While for languages, every non-deterministic automaton can be made deterministic, this is not anymore the case for transducers, if one considers one-way transducers. However, the result holds for functional two-way transducers, as stated below:

Theorem 21. *The following results hold:*

- DFT *are strictly less expressive than* fNFT*.*
- 2DFT *and* f2NFT *are expressively equivalent* [3]*.*

The first point is illustrated by example R_1. The second point is a more difficult result that has been proven in [Eng82, EH01, dS13].

[3] To be precise, this equivalence holds if left and right end-markers are added to the input word, so that the deterministic transducer can identify the beginning and the end of the input.

Determining whether a transducer recognizes a function is thus a natural question. This problem has been solved in 1983 for 2NFT in [CK87]. For NFT, the problem can be solved in polynomial time, and different proofs have been proposed. An elegant one, based on a squaring construction of transducers, can be found in [BCPS03]. We will use this construction later, so we briefly present it.

The square T^2 of a transducer T is simply the product of T by itself. It is a transducer which outputs pairs of words. The characterization of [BCPS03] states that T is functional iff the two following properties are satisfied:

1. For every two runs $(i_1, i'_1) \xrightarrow{u_1|(v_1,v'_1)} (q, q')$ and $(i_2, i'_2) \xrightarrow{u_2|(v_2,v'_2)} (q, q')$ in the trimmed [4] part of T^2, we have $\mathsf{delay}(v_1, v'_1) = \mathsf{delay}(v_2, v'_2)$.

2. For every run $(i, i') \xrightarrow{u|(v,v')} (q, q')$ in T^2 such that $q, q' \in F$, we have $v = v'$.

It yields a simple PTime algorithm which builds the square, trims it, and then applies an action computing the value of delay.

The first point of the previous theorem rises the problem of determining, given a fNFT, whether there exists an equivalent DFT. This determinization problem has been solved in 1977 by Choffrut, using the so-called twinning property. We give a short presentation of this result, as some of our further developments are based on the twinning property. The definition we give here is a different, but equivalent, presentation of the definition given by Choffrut. Let $T = (Q, I, F, \delta)$ be a trimmed fNFT. T satisfies the twinning property if for all $q_0, q'_0 \in I$, for all $q, q' \in Q$, for all words $u_1, v_1, w_1, u_2, v_2, w_2 \in \Sigma^*$, if:

$$q_0 \xrightarrow{u_1/v_1} q \xrightarrow{u_2/v_2} q \qquad \text{and} \qquad q'_0 \xrightarrow{u_1/w_1} q' \xrightarrow{u_2/w_2} q' \text{ are runs of } T,$$

then $\mathsf{delay}(v_1, w_1) = \mathsf{delay}(v_1v_2, w_1w_2)$.

The twinning property does not exactly characterize the class of functions that are DFT definable, but a slightly larger superclass, that of so-called *subsequential* functions. Such a function is described by a DFT with additionally finite words associated with final states. The image of an input word u is the word vw, where v is the word produced by the accepting run ρ on u, and w is the word associated with the final state reached by ρ.

Theorem 22. *Let T be a* fNFT*. The following results hold:*

- *T satisfies the twinning property iff $[\![T]\!]$ is subsequential.*
- *Determining whether T satisfies the twinning property can be done in polynomial time.*

The first point of this theorem has been proven in [Cho77]. The decidability in PTime is shown in [WK95].

Domain, range and closure properties The *domain* (resp. *range*) of a transducer T is defined as $\mathrm{dom}(T) = \mathrm{dom}([\![T]\!])$ (resp. $\mathrm{range}(T) = \mathrm{range}([\![T]\!])$). The domain $\mathrm{dom}(T)$ is a regular language that can be defined by the 2NFA obtained by projecting away the output part of the transitions of T, called the *underlying input automaton*.

[4] A finite-state automaton is trimmed whenever for every state q, there exists an accepting run going through state q.

Regarding the range, one easily observes that ranges of NFT are also regular languages. This is however not the case for 2NFT, as one can easily come up with the set of palindromes for instance (consider a transducer, obtained as a slight modification of the transducer T_2 of Example 8, that copies the input word once during a first left-to-right pass, and then copies its mirror image in a second right-to-left pass). The languages obtained as range of 2NFT can be described by means of linear context-free graph grammars. This is out of the scope of this thesis, and we refer to [EH91] for further details.

It is well known that the class of regular languages is closed under boolean operations. The situation for transductions is different as they are in general not closed under intersection and complement. Indeed, transductions defined by NFT and 2DFT are such that each word has finitely many images. This immediately forbids closure under complement. Non-closure under intersection is standard and can be proved by observing that the domain of the intersection of relations $\{(a^n b^m, c^n \mid n, m \in \mathbb{N})\}$ and $\{(a^n b^m, c^m \mid n, m \in \mathbb{N})\}$ is not a regular language.

Similarly, transductions defined by NFT and 2DFT are not closed under inverse as the inverse image of a word may be infinite.

Last, we consider closure under composition. This operation is natural for functions, and relations in general. It is actually rather easy to prove that NFT are effectively closed under composition. Concerning 2DFT, it has been proved in [CJ77] that these transducers are also closed under composition, but the construction is much more involved.

In contrast, 2NFT are not closed under composition, see [EH07] for further details.

Decidability properties We give the decidability status and the complexity class of the decision problems introduced previously. The results are summarized in Table 4.1.

First, emptiness in PTime for the classes of DFT, fNFT and NFT trivially follows from standard reachability in graphs. For two-way machines, first observe that we only have to consider the underlying input automaton. This two-way automaton can be turned into an equivalent DFA using Shepherdson's construction, resulting in an exponential size automaton. However, this automaton can be built on-the-fly, resulting in a PSpace procedure. Hardness follows for instance from emptiness of intersection of finitely many DFA.

Concerning type checking, the hardness comes from that of language inclusion, already for DFT (consider the identity function). We explain why type checking for NFT is decidable in PSpace. Let us denote by A, B the input and output NFA and by T the NFT under consideration. Using a previous remark, we explain how to build a NFT for the transduction $R = Id_{\overline{L(B)}} \circ [\![T]\!] \circ Id_{L(A)}$. First, the composition $[\![T]\!] \circ Id_{L(A)}$ follows from a standard product construction of A and T. Similarly, considering an automaton C such that $L(C) = \overline{L(B)}$, one can build the product with C in order to recognize R. As a consequence, one has to check emptiness of the resulting transducer whose size is exponential. However, this transducer can still be built on-the-fly, yielding the result. The same construction can be used for an input 2NFT T, resulting in an ExpSpace procedure.

Last, regarding equivalence, the undecidability for NFT has been proven in [Gri68], while the PSpace-completeness for 2DFT comes from [Gur82]. For the class of fNFT, the PSpace hardness follows from that of language inclusion (by checking the domains). Once the domains are known to be equal, the equivalence amounts to functionality checking, decidable in PTime.

	emptiness	type checking (vs NFA)	equivalence
DFT	PTIME	PSPACE-C	PTIME
fNFT	PTIME	PSPACE-C	PSPACE-C
2DFT	PSPACE-C	EXPSPACE	PSPACE-C
NFT	PTIME	PSPACE-C	undecidable
2NFT	PSPACE-C	EXPSPACE	undecidable

Table 4.1: Decision problems for finite-state transducers.

4.3 Streaming string transducers

Recently, a new model of transducer has been proposed by Rajeev Alur and Pavol Černý in [AČ11], which captures exactly the class of regular string functions [5]. Streaming String Transducers, unlike the models we have defined before, are not finite state. They are indeed equipped with a finite set of variables storing finite words that can be reset to a finite word, and that can be concatenated with a finite word or with another variable. Moreover, their finite state behavior is simple as it is defined by a deterministic one-way automaton. Intuitively, the complexity of the two-wayness of 2DFT is replaced by operations on string variables.

Let $\mathcal{X}$ be a finite set of variables denoted by $X, Y, \ldots$ and Σ be a finite alphabet. A substitution σ is defined as a mapping $\sigma : \mathcal{X} \to (\Sigma \cup \mathcal{X})^*$. Let $\mathcal{S}_{\mathcal{X},\Sigma}$ be the set of all substitutions. Any substitution σ can be extended to $\hat{\sigma} : (\Sigma \cup \mathcal{X})^* \to (\Sigma \cup \mathcal{X})^*$ in a straightforward manner. The composition $\sigma_1\sigma_2$ of two substitutions σ_1 and σ_2 is defined as the standard function composition $\hat{\sigma_1}\sigma_2$.

A streaming string transducer (SST) over Σ is a tuple $T = (Q, q_0, \delta, \mathcal{X}, \rho, F)$ where Q is a finite set of states with initial state q_0, $\delta : Q \times A \to Q$ is a transition function, $\mathcal{X}$ is a finite set of variables, $\rho : \delta \to \mathcal{S}_{\mathcal{X},\Sigma}$ is a variable update function, and $F : Q \rightharpoonup (\mathcal{X} \cup \Sigma)^*$ is a (partial) output function.

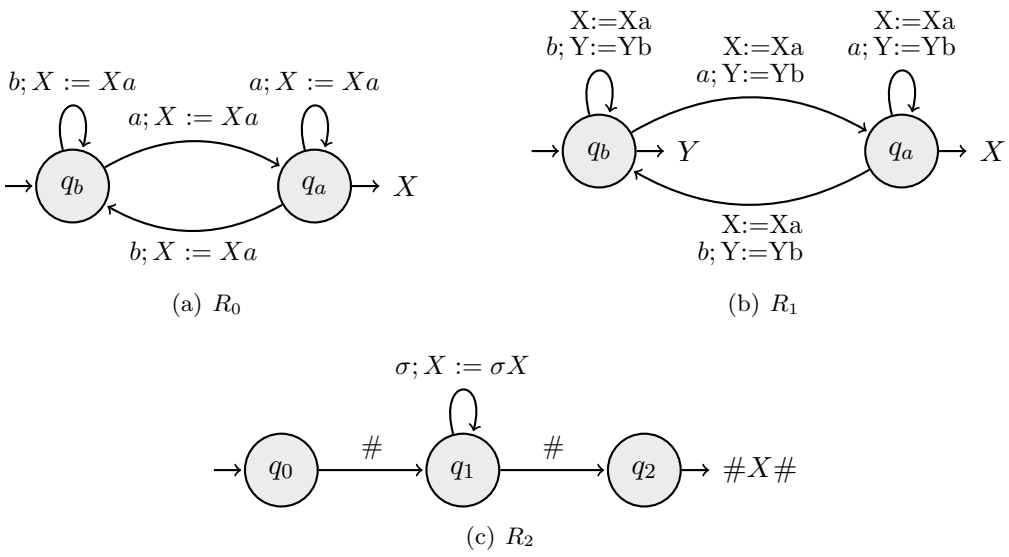
The concept of a run of an SST is defined in an analogous manner to that of a finite state automaton. The sequence $\langle \sigma_{r,i} \rangle_{0 \leq i \leq |r|}$ of substitutions induced by a run $r = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \ldots q_{n-1} \xrightarrow{a_n} q_n$ is defined inductively as the following: $\sigma_{r,i}{=}\sigma_{r,i-1}\rho(q_{i-1}, a_i)$ for $1 < i \leq |r|$ and $\sigma_{r,1} = \rho(q_0, a_1)$. We denote $\sigma_{r,|r|}$ by σ_r.

As the underlying automaton of an SST is deterministic, given a state q and a word u, there is at most one run r on u starting in state q. We then denote the substitution σ_r by $\sigma_{q,u}$.

If r is accepting, *i.e.* $q_n \in Q_f$, we can extend the output function F to r by $F(r) = \sigma_\varepsilon \sigma_r F(q_n)$, where σ_ε substitutes all variables by their initial value ε. For all words $w \in A^*$, the output of w by T is defined only if there exists an accepting run r of T on w, and in that case the output is denoted by $T(w) = F(r)$. The *domain* of T, denoted by $\mathrm{dom}(T)$, is defined as the set of words w on which there exists an accepting run of T. The transformation $[\![T]\!]$ defined by T is the function which maps any word $w \in \mathrm{dom}(T)$ to its output $T(w)$.

It is not difficult to observe that this definition yields exponential size increase transformations, *i.e.*, one can define an SST with a single variable which maps the input word a^n to the output word a^{2^n}, by considering a rule $X := XX$. As transformations defined by finite-state

[5] The model introduced in [AČ11] considers an infinite data domain but for the sake of coherence we present it for a finite alphabet Σ.



(a) R_0 (b) R_1 (c) R_2

Figure 4.4: SST realizing transformations R_0, R_1 and R_2.

transducers are linear size increase, one needs to restrict SST to recover the expressiveness of regular string functions. The good restriction is to require the variable update function to be *copyless*. A substitution σ is copyless if for any variable X, there exists at most one variable Y such that X occurs in $\sigma(Y)$. We say that an SST T is copyless if every substitution of its variable update function is copyless. Alur and Černý proved the following result:

Theorem 23 ([AČ10a])**.** *Copyless* SST *and* 2DFT *are expressively equivalent.*

It is worth observing that the translations are effective, and go through the equivalent model of monadic second-order transductions, defined in Section 4.4, resulting in non-elementary complexity (from SST to 2DFT). We propose new constructions in Section 5.4.

Example 9. *Let us consider again the examples given to present finite state transducers. One can easily come up with simple* SST *realizing the transformations R_0, R_1 and R_2. Observe that the* SST *realizing transformations R_0 and R_1, which are definable by one-way transducers, do only use the operation of appending a string to a variable. This will be formalized later.*

Other restrictions than the copyless one have been introduced in the literature. In order to present them, we introduce the notion of flow of variable. Let T be an SST (non-necessarily copyless) with set of states Q and set of variables $\mathcal{X}$. Let u be a string in Σ^* such that there exists a run r in T from state p to state q, inducing substitution σ. Let X, Y be two variables, and $j \in \mathbb{N}$, we say that Y flows j times in X from p to q on u if Y occurs j times in $\sigma(X)$. If this holds, we denote it by $(p, Y) \overset{u}{\rightsquigarrow}_j (q, X)$.

Observe that if T is copyless, then by definition for any input word u and any variable Y, there is at most one variable X such that Y flows in X, and if this holds then Y flows exactly once in X.

Weaker restrictions have been considered. We say that the SST T is:

- 1-bounded if for every p, q, X, Y, u, if $(p, Y) \overset{u}{\rightsquigarrow}_j (q, X)$, then $j = 0$ or $j = 1$,

- k-bounded if for every p, q, X, Y, u, if $(p, Y) \overset{u}{\rightsquigarrow}_j (q, X)$, then $0 \leq j \leq k$,
- bounded if there exists k such that it is k-bounded.

Theorem 24 ([AFT12])**.** *Copyless* SST*,* 1*-bounded* SST*, and bounded* SST *are expressively equivalent.*

Closure and decidability properties. Thanks to the (effective) equivalence with 2DFT, several properties follow immediately. However, they can also be proved directly on (copyless) SST. The decidability of the type-checking against regular languages is proven to be decidable in PSpace in [AČ11], as well as equivalence. Closure under composition is proven in [AČ11].

As we have explained before, when removing the copyless assumption, one obtains a model that can induce exponential size increase. Together with Emmanuel Filiot, we have studied this model, and proven that it is actually very similar to HDT0L systems (HDT0L systems roughly consist in the iterations of a finite set of morphisms on a given word). Intuitively, we simulate the applications of morphisms using substitutions, and vice versa. As a consequence, using the result of Culik and Karhumaki stating the decidability of the equivalence of two HDT0L systems [IK86], we prove:

Theorem 25 ([17])**.** *The equivalence problem for copyfull* SST *is decidable.*

The subclass of rational functions One-way transducers play an important role in transducer theory as they are for instance heavily used in applications related to language processing. It is worth mentioning that there exists a simple and natural restriction of SST which exactly characterizes the class of rational functions.

Given an SST $T = (Q, q_0, \delta, \mathcal{X}, \rho, F)$, we say that T is *right-appending* if the only updates of variables are of the form $X := Yu$ with $X, Y \in \mathcal{X}$ and $u \in \Sigma^*$, and if the output functions are of the form Xu with $X \in \mathcal{X}$ and $u \in \Sigma^*$. We denote by ra-SST this class of SST.

Theorem 26. *Right-appending* SST *recognize exactly rational functions.*

This result holds actually for an arbitrary semiring $\mathcal{S}$, and in this case one can characterize unambiguous weighted automata on $\mathcal{S}$ (see [ADD+13] for details). We present the proof of this result in the setting of transducers for completeness.

Given a right-appending SST $T = (Q, q_0, \delta, \mathcal{X}, \rho, F)$, one can build a one-way transducer $T' = (Q', I', F', \delta')$ as follows. States are enriched with variables, *i.e.* $Q' = Q \times \mathcal{X}$. Initial states are defined as $I' = \{q_0\} \times \mathcal{X}$. Final states are defined as $F' = \{(q, X) \in Q' \mid F(q) = X\}$. Elements of δ' are tuples $((p, X), a, u, (q, Y))$ such that $\delta(p, a) = q$ and $\rho(p, a)(Y) = Xu$.

One easily verifies that the fact that the underlying automaton of T is deterministic implies that T' is unambiguous, and thus functional.

Conversely, given a functional one-way transducer $T = (Q, I, F, \delta)$, one can build an equivalent right-appending SST by using a kind of subset construction. Formally, we define an appending SST $T' = (Q', q'_0, \delta', \mathcal{X}, \rho, F')$ as follows:

- states of T' are subsets of Q, *i.e.* $Q' = 2^Q$,
- the initial state is $q'_0 = I$,
- we consider one variable for each state, *i.e.* $\mathcal{X} = \{X_q \mid q \in Q\}$,

- δ' is defined by $\delta'(S,a) = \{q \in Q \mid \exists p \in S, u \in \Sigma^*.(p,a,u,q) \in \delta\}$
- ρ is defined as follows: let $S \in Q'$, $a \in \Sigma$ and $p \in Q$, then there are two cases. If $p \in \delta'(S,a)$, then there exist $q \in S$, $u \in \Sigma^*$ such that $(q,a,u,p) \in \delta$. Then we choose such elements q and u, and let $\rho(S,a)(X_p) = X_q u$. Otherwise, if $p \notin \delta'(S,a)$, we let $\rho(S,a)(X_p) = \varepsilon$. In the former case, the fact that the choice of q and u is not important relies on the functionality of T. In the latter case, the variable X_p will not be involved in the final output, and we reset it as we have to define $\rho(S,a)(X_p)$, but its value is useless.
- last, the final output function F' is defined only for states S such that $S \cap F \neq \emptyset$, and we let $F'(S) = X_p$ for some $p \in S \cap F$. Again, the choice of such a state p is not important, thanks to the functionality of T.

4.4 Logical presentation of transducers

As the class of regular languages has a logical presentation, one may be interested in a logical presentation for transductions. In this section, we present a logic-based formalism introduced by Bruno Courcelle [Cou94] to define transformations of graphs seen as logical structures. For further details, we refer the reader to the recent survey [Fil15] on logic-automata connections for transformations.

Recall that a word u over an alphabet Σ can be seen as a logical structure over the signature $\mathcal{S}_\Sigma$ composed of the unary predicates $(L_a)_{a\in\Sigma}$ that define the labels of the positions in u, and the binary predicate $\preceq$ that defines the order on the word positions. Given an alphabet Σ, monadic second-order formulas (MSO formulas) over the signature $\mathcal{S}_\Sigma$ are built over first-order variables $x, y \ldots$ and second-order variables $X, Y \ldots$ They are defined by the following grammar:

$$\phi ::= \exists X \cdot \phi \mid \exists x \cdot \phi \mid \phi \wedge \phi \mid \neg\phi \mid x \in X \mid L_a(x) \mid x \preceq y$$

Universal quantifiers as well as other Boolean connectives are defined naturally, and we use the standard elementary logical definitions: free variable, bound variable...

A logical transducer defines, given an input logical structure corresponding to some word u, a new logical structure corresponding to some word v. The output structure is defined by taking a finite number of copies of the input structure domain. Formally, a logical MSO transducer (MSOT for short) over the alphabet Σ is a tuple $T = (C, \phi_{\text{dom}}, (\phi^c_{\text{pos}})_{c\in C}, (\phi^c_a)_{c\in C, a\in\Sigma}, (\phi^{c,d}_{\preceq})_{c,d\in C})$ where C is a finite copy set, and ϕ_{dom} is an MSO sentence over $\mathcal{S}_\Sigma$, ϕ^c_{pos} and ϕ^c_a are MSO formula over $\mathcal{S}_\Sigma$ with one free first-order variable, while $\phi^{c,d}_{\preceq}$ are MSO formula over $\mathcal{S}_\Sigma$ with two free first-order variables.

Given an input word u, the output structure computed by such an MSOT T is defined by the domain $D = \{(x,c) \in \text{dom}(u) \times C \mid u \models \phi^c_{\text{pos}}(x)\}$ (each node (x,c) is denoted hereafter x^c) and for all $x^c, y^d \in D$, x^c is labeled by $a \in \Sigma$ iff $u \models \phi^c_a(x)$, and $x^c \preceq y^d$ holds iff $u \models \phi^{c,d}_{\preceq}(x,y)$. The sentence ϕ_{dom} gives the domain of the transduction realized by T, denoted $\text{dom}(T)$. We assume the following consistency property: for every $u \in \text{dom}(T)$, this output structure encodes a word v (note that this is decidable as it can be expressed as an MSO property on words, see [Fil15]). In this case, we say that the pair (u,v) is recognized by T, and denote by $[\![T]\!]$ the set of such pairs of words.

Example 10. *We consider again the transductions R_0, R_1 and R_2. Recall that $\Sigma = \{a, b\}$. We start with R_0 which is realized by the* MSOT *defined as follows: we let C be a singleton (and thus omit superscript c in the sequel), $\phi_{\text{dom}} = \exists x \cdot \mathsf{last}(x) \wedge L_a(x)$ (where $\mathsf{last}(x)$ characterizes the last position of the word), $\phi_{\text{pos}}(x) = \texttt{true}$, $\phi_a(x) = \texttt{true}$ and $\phi_b(x) = \texttt{false}$, and finally we let $\phi_{\preceq}(x, y) = x \preceq y$. Intuitively, this MSOT selects using its domain formula input words $u \in \Sigma^+$ such that the last letter is a, then copies the structure and replaces every label by the letter a.*

The transduction R_1 is realized by the MSOT *defined as follows: we let C be a singleton, $\phi_{\text{dom}} = \exists x \cdot \texttt{true}$, $\phi_{\text{pos}}(x) = \texttt{true}$, $\phi_\sigma(x) = \exists y \cdot \mathsf{last}(y) \wedge L_\sigma(y)$ for $\sigma \in \Sigma$, and finally we let $\phi_{\preceq}(x, y) = x \preceq y$. This MSOT selects using its domain formula input words $u \in \Sigma^+$, copies the structure and replaces every label by the one of the last letter of u.*

Last, the transduction R_2 is realized by the MSOT *defined as follows: we let C be a singleton, we do not detail ϕ_{dom} but it is clear that $\#\Sigma^*\#$ is a regular language, $\phi_{\text{pos}}(x) = \texttt{true}$, $\phi_\sigma(x) = L_\sigma(x)$ for $\sigma \in \Sigma$, and finally we let $\phi_{\preceq}(x, y) = y \preceq x$. This MSOT simply inverses the order relation using $\phi_{\preceq}$.*

Observe that these examples use a singleton copy set C, but for instance, in order to realize the function copy : $\Sigma^* \to \Sigma^*$ defined as copy$(u) = uu$, we would need a copy set of size two.

Joost Engelfriet and Hendrik Jan Hoogeboom proved the following theorem, which justifies to denote this class by the class of *regular* string functions:

Theorem 27 ([EH01])**.** MSOT *and* 2DFT *are expressively equivalent.*

As it has been done for regular languages, a natural objective is to provide similar logic-automata connections for subclasses of regular functions. We start with rational functions that own a simple characterization in terms of logic. Intuitively, we will simply impose the one-way-ness in the output structures produced by the MSOT.

We say that an MSOT $T = (C, \phi_{\text{dom}}, (\phi^c_{\text{pos}})_{c \in C}, (\phi^c_a)_{c \in C, a \in \Sigma}, (\phi^{c,d}_{\preceq})_{c,d \in C})$ is *order-preserving* if for all words $u \in \text{dom}(T)$, all positions x, y in u, all copies $c, d \in C$, if $u \models \phi^{c,d}_{\preceq}(x, y)$, then $x \preceq y$ holds true. Emmanuel Filiot proved recently the following theorem.

Theorem 28 ([Fil15])**.** *Order-preserving* MSOT *and* fNFT *are expressively equivalent.*

In Section 5.4, we consider the fragment of transductions that are definable using only first-order formulae with the predicate of linear order. As we will see, this fragment owns representations in terms of aperiodic transducers.

4.5 Summary

We have presented three different models of transducers, and have identified two important classes of string-to-string functions.

First, regular functions are characterized as f2NFT, 2DFT, (copyless or bounded) SST, and MSOT. It is natural to study translations between these models, and part of the results can be found in the literature; they are summarized on Figure 4.5. Observe that there exists no direct translation from SST to 2DFT. We will propose such a construction in Section 5.4.

A second important class of functions is that of rational string-to-string functions. This class can be equivalently described by means of fNFT, ra-SST, and order-preserving MSOT. We will study (and refine) this class in Section 5.3.

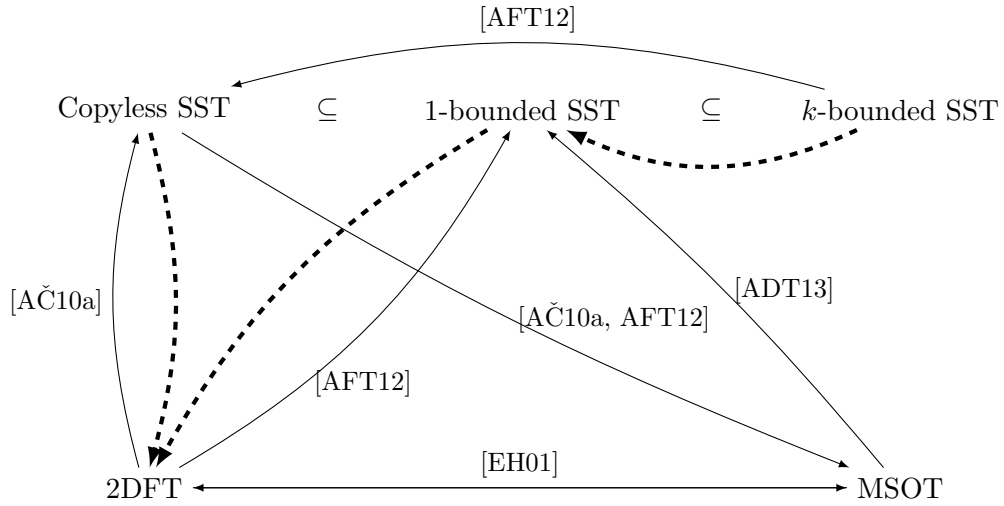


Figure 4.5: Summary of transformations between equivalent models. Dashed arrows correspond to new constructions that are presented in Section 5.4.

We have focused on functions here, but the generalization to relations is worth being studied too as, for instance, the equivalence problem is known to be decidable for finite-valued rational relations.

Chapter 5

String-to-string transductions

Contents

The results presented in the first three sections of this chapter deal with memory analysis of programs processing words. This is an important tool for ensuring system robustness, and in particular when the input words are streams.

We investigate *static analysis* of memory usage for a special kind of programs on words, namely programs defined by *transducers*. We assume that the transducers are *functional* and *non-deterministic*. Non-determinism is required as input words are read from left to right in a single pass and some actions may depend on the future of the stream. For instance, the XML transformation language XSLT [Cla99] uses XPath for selecting nodes where local transformations are applied, and XPath queries rely on non-deterministic moves along tree axes, such as a move to any descendant.

The results presented in the last section are more of an algebraic nature. They aim at improving our understanding of the class of transductions which are definable in the first-order logic with the order relation. Characterizations in terms of aperiodic two-way transducers, and aperiodic streaming string transducers are studied.

5.1 Streamability

In order to formally define the complexity classes for evaluation that we target, we introduce a *deterministic* computational model for word transductions that we call *Turing Transducers*. Turing transducers have three tapes: one read-only left-to-right input tape, one write-only left-to-right output tape, and one standard working tape. Such a machine naturally defines a transduction: the input word is initially on the input tape, and the result of the transduction is the word written on the output tape after the machine terminates in an accepting state. We denote by $[\![M]\!]$ the transduction defined by M. The space complexity is measured on the working tape only.

We consider in this section finite-state string transductions that can be evaluated with a constant amount of memory if we fix the machine that defines the transduction. Intuitively, this corresponds to what we denote by *streamable* transformation.

Definition 6. *A (functional) transduction $R \subseteq \Sigma^* \times \Sigma^*$ is* bounded memory (BM) *if there exists a Turing transducer M and $K \in \mathbb{N}$ such that $[\![M]\!] = R$ and on any input word $u \in \Sigma^*$, M runs in space complexity at most K.*

It is not difficult to verify that for string-to-string functions, bounded memory characterizes the class of subsequential functions.

Proposition 9. *Let $f : \Sigma^* \rightharpoonup \Sigma^*$. Then f is BM iff f is a subsequential function.*

This result follows from the two following simple observations: first, a subsequential transducer can be evaluated using bounded memory; second, a Turing transducer using a bounded memory can be understood as a DFT.

Corollary 1 ([13])**.** *Given a* fNFT *T, we can decide in polynomial time whether $[\![T]\!]$ is bounded memory.*

5.2 From two-way to one-way transducers

On finite strings, it is well-known that the expressive power of finite state automata does not increase when the reading head can move left and right, even in presence of non-determinism: the class 2NFA is no more powerful than the class NFA. The proof of this result was first shown in the seminal paper of Rabin and Scott [RS59], and independently by Shepherdson [She59]. An alternative proof can be found in [Var89]. This equivalence also holds for weighted automata over commutative semi-rings, as it is shown in [Ans90, CL14].

For string-to-string transducers (which can be viewed as weighted automata over a non-commutative semi-ring), two-wayness *does* increase the expressive power (consider for instance the mirror transformation). This statement holds both for general transducers, and for functional ones. In other terms, regular string functions strictly contain rational string functions. In this section, we address the following natural definability problem: given a regular string function, is it rational? In terms of transducers, the problem reads as follows: given a f2NFT, does there exist an equivalent fNFT?

Theorem 29 ([14])**.** *For all functional* 2NFT *T, it is decidable whether the transduction defined by T is definable by an* NFT*.*

The proof of Theorem 29 extends the proof of Rabin and Scott [RS59] from automata to transducers [1].

This result implies the decidability of the streamability (*i.e.* bounded memory evaluation by a Turing transducer) for regular string functions. The next result follows indeed from Proposition 9 and Theorems 22 and 29.

Corollary 2 ([14])**.** *Given a* f2NFT *T, we can decide whether $[\![T]\!]$ is bounded memory.*

As a consequence of Theorems 27 and 28, our result also implies:

Corollary 3. *Let T be an* MSOT*, we can decide whether $[\![T]\!]$ is order-preserving* MSOT*-definable.*

[1] Shepherdson and then Vardi proposed arguably simpler constructions for automata. It is however not clear to us how to extend these constructions to transducers.

Rabin and Scott's approach in a nutshell

The proof of Theorem 29 relies on the same ideas as Rabin and Scott's construction for automata [RS59]. It is based on the following key observation: any accepting run is made of many zigzags, and those zigzags are organized by a nesting hierarchy: zigzag patterns may be composed of simpler zigzag patterns. The simplest zigzags of the hierarchy are those that do not nest any other zigzag: they are called z-motions. Rabin and Scott described a procedure that removes those zigzags by iterating a construction that removes z-motions.

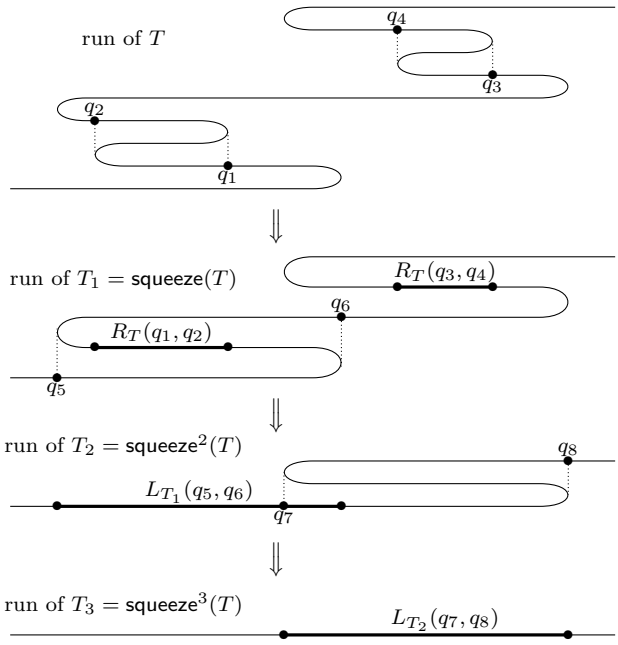


Figure 5.1: Zigzags removal by applications of squeeze.

If T is a 2NFA, it is possible to construct a new automaton denoted by $\mathsf{squeeze}(T)$ such that, for all accepting runs ρ of T on some input word u, there exists a "simpler" accepting run of $\mathsf{squeeze}(T)$ on u, obtained from ρ by replacing some z-motions by one-way runs that simulate three passes in parallel. It is illustrated by Fig. 5.1. For instance at the first step, there are two z-motions from q_1 to q_2 and from q_3 to q_4 respectively. Applying $\mathsf{squeeze}(T)$ consists in non-deterministically guessing those z-motions and simulating them by one-way runs. This is done by the NFA $R_T(q_1, q_2)$ and $R_T(q_3, q_4)$ respectively. Depending on whether the z-motions enter from the left or the right, z-motions are replaced by runs of NFA $R_T(.,.)$ (that read the input backwardly) or $L_T(.,.)$, as illustrated by the second iteration of squeeze on Fig. 5.1.

An N-crossing run ρ (*i.e.* each position of the input is visited at most N times) can be simplified into a one-way run after at most N^2 applications of squeeze. This result was not published in [RS59] so we proved it in [14]. At the automata level, it is known that for all words u accepted by a 2NFA T with N states, there exists an N-crossing accepting run on u. Therefore it suffices to apply squeeze N^2 times to T. One gets an equivalent 2NFA T^* from which the backward transitions can be removed while preserving equivalence with T^*, and so T.

Z-motion transducers

The basic brick of the Rabin and Scott's approach is the simulation of z-motions by one-way runs. While this is always doable for automata, this is not the case for transducers. Formally, a z-motion is a run $\rho = (q_j, i_j)_j$ such that its projection on the second component is of the form $1, \dots, n, n-1, \dots, 1, \dots, n$ for some n (the form $n, \dots, 1, 2, \dots, n, \dots, 1$ should also be considered but we omit it to simplify the presentation). We introduce in this paragraph a special class of transducers whose runs are z-motions, and show in the next paragraph how to decide whether such transducers are equivalent to some NFT. This result will be the basic brick of our procedure for proving Theorem 29.

z-motion transducers are defined like 2NFT except that they must define *functions* and to be accepting, a run on a word of length n must be of the form $\rho.(q_f, n+1)$ where ρ is a z-motion run and q_f is an accepting state. The class of z-motion transducers is denoted by ZNFT.

Let $T \in$ ZNFT and $\rho = (p_1, 1) \dots (p_n, n)\ (q_{n-1}, n-1) \dots (q_1, 1)(r_2, 2) \dots (r_{n+1}, n+1)$ be a run of T on a word of length n. In order to make our notations uniform, we let $q_n = p_n$ and $r_1 = q_1$ and define the following shortcuts: for $1 \leq i \leq j \leq n$, $\mathrm{out}_1[i,j] = \mathrm{out}((p_i, i) \dots (p_j, j))$, and $\mathrm{out}_2[i,j] = \mathrm{out}((q_j, j) \dots (q_i, i))$ and $\mathrm{out}_3[i,j] = \mathrm{out}((r_i, i) \dots (r_j, j))$, and $\mathrm{out}_3[i, n+1] = \mathrm{out}((r_i, i) \dots (r_{n+1}, n+1))$.

We characterize the NFT-definability of a ZNFT by a property that we prove to be decidable. Intuitively, this property which relies on word combinatorics gives the most general form the output words of the run of a ZNFT may have to be NFT definable.

Definition 7 ($\mathcal{P}$-property)**.** *Let $T \in$ ZNFT with m states, and let $(u, v) \in R(T)$ where u has length n. Let $K = 2.o.m^3.|\Sigma|$ where $o = max\{|v| \mid (p, a, v, q, m) \in \Delta\}$. The pair (u, v) satisfies the property $\mathcal{P}$, denoted by $(u, v) \models \mathcal{P}$, if for all accepting runs ρ on u, there exist two positions $1 \leq \ell \leq \ell' \leq n$ and $w, w', t_1, t_2, t_3, x_1, x_2, y_1, y_2, y_3, z_1, z_2 \in \Sigma^*$ such that $v \in w t_1 t_2^* t_3 w'$ and:*

$$
\begin{array}{lll}
\mathrm{out}_1[1,\ell] = w & \mathrm{out}_1[\ell,\ell'] = x_1 & \mathrm{out}_1[\ell', n] = x_2 \\
\mathrm{out}_2[1,\ell] = y_3 & \mathrm{out}_2[\ell,\ell'] = y_2 & \mathrm{out}_2[\ell', n] = y_1 \\
\mathrm{out}_3[1,\ell] = z_1 & \mathrm{out}_3[\ell,\ell'] = z_2 & \mathrm{out}_3[\ell', n] = w' \\
|s| \leq K,\ \forall s \in \{t_1, t_2, t_3, x_2, y_1, y_3, z_1\} & & \\
(\dagger)\quad x_1 x_2 y_1 y_2 y_3 z_1 z_2 \in t_1 t_2^* t_3 & &
\end{array}
$$

This decomposition is depicted in Fig. 5.2. T satisfies property $\mathcal{P}$, denoted $T \models \mathcal{P}$, if all $(u, v) \in R(T)$ satisfy it.

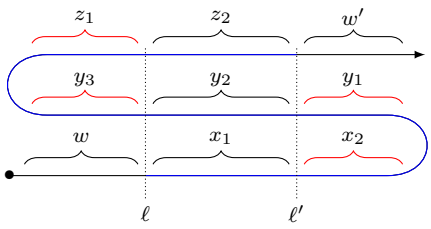


Figure 5.2: Output decomposition in property $\mathcal{P}$.

The following key lemma is proved in the next paragraph.

Lemma 14. *Let $T \in$ ZNFT. $T \models \mathcal{P}$ iff T is* NFT*-definable. Moreover, $\mathcal{P}$ is decidable and if $T \models \mathcal{P}$, one can (effectively) construct an equivalent* NFT*.*

Definition 8 (z-motion transductions induced by a f2NFT)**.** *Let $T = (Q, q_0, F, \Delta)$ be a* f2NFT *and $q_1, q_2 \in Q$. The transduction $\mathcal{L}_T(q_1, q_2)$ (resp. $\mathcal{R}_T(q_1, q_2)$) is defined as the set of pairs (u_2, v_2) such that there exist $u \in \Sigma^*$, two positions $i_1 < i_2$ (resp. $i_2 < i_1$), an accepting run ρ of T on u which can be decomposed as $\rho = \rho_1(q_1, i_1)\rho_2(q_2, i_2)\rho_3$ such that $u_2 = u[i_1 \dots i_2]$ and*

- *$(q_1, i_1)\rho_2(q_2, i_2)$ is a z-motion run*
- *$\mathrm{out}((q_1, i_1)\rho_2(q_2, i_2)) = v_2$*

Proposition 10. *The transductions $\mathcal{R}_T(q_1, q_2)$ and $\mathcal{L}_T(q_1, q_2)$ are* ZNFT*-definable.*

To prove the next lemma, it is crucial in Definition 8 to make sure that the z-motion $(q_1, i_1)\rho_2(q_2, i_2)$ can be embedded into a global accepting run of T. Without that restriction, it might be the case that $\mathcal{L}_T(q_1, q_2)$ or $\mathcal{R}_T(q_1, q_2)$ is not NFT-definable although the 2NFT T is. Indeed, the domain of $\mathcal{L}_T(q_1, q_2)$ or $\mathcal{R}_T(q_1, q_2)$ would be too permissive and accept words that would be otherwise rejected by other passes of global runs of T. This is another difficulty when lifting Rabin and Scott's proof to transducers, as for automata, the context in which a z-motion occurs is not important.

Lemma 15. *If T is* NFT*-definable, then so are the transductions $\mathcal{R}_T(q_1, q_2)$ and $\mathcal{L}_T(q_1, q_2)$ for all states q_1, q_2. Moreover, it is decidable whether the transductions $\mathcal{R}_T(q_1, q_2)$ and $\mathcal{L}_T(q_1, q_2)$ are* NFT*-definable.*

From Z-motion to one-way transducers

We give some details on the proof of Lemma 14 which is the core of our decision procedure. To prove that property $\mathcal{P}$ is a necessary condition, one uses pumping in the input word so as to identify loops which allow to apply word combinatorics techniques. Conversely, we present a construction which turns a ZNFT into an NFT.

Let T be a ZNFT. We sketch the construction of a NFT T' such that $[\![T']\!] = \{(u, v) \in [\![T]\!] \mid (u, v) \models \mathcal{P}\}$. As a consequence, we immediately obtain that if $T \models \mathcal{P}$, then T is NFT-definable.

Initially, T' guesses words $t_1, t_2, t_3, x_2, y_1, y_3$ and z_1 (of bounded length). These guesses also imply constraints on words x_1, y_2 and z_2, thanks to property (†). Describing these constraints is a tedious case analysis that we do not detail here. This case analysis depends on the lengths of words x_1, x_2y_1, y_2, y_3z_1 and z_2, w.r.t. the lengths of words t_1, t_2 and t_3. In order to illustrate our construction, we assume from now that the following inequalities hold: $|x_1| \geq |t_1|$, $|x_2y_1| \geq |t_2|$, $|y_2| \geq |t_2|$, $|y_3z_1| \geq |t_2|$, and $|z_2| \geq |t_3|$. The other cases (there are finitely many cases) should be considered similarly. The previous inequalities imply that t_1 is a prefix of x_1 and that t_3 is a suffix of z_2. One can then refine the property (†) and write the new set of equalities (‡) as follows:

$$(\ddagger) \qquad x_1 = t_1 t_2^{n_1} \alpha_1, \quad x_2 y_1 \in \beta_1 t_2^{m_1} \alpha_2, \quad y_2 \in \beta_2 t_2^{n_2} \alpha_3, \quad y_3 z_1 \in \beta_3 t_2^{m_2} \alpha_4, \quad z_2 \in \beta_4 t_2^{n_3} t_3$$

where $t_2 = \alpha_i \beta_i$ for all $i \in \{1, 2, 3, 4\}$, $n_1, n_2, n_3, m_1, m_2 \in \mathbb{N}$. As t_2 is of bounded length, so are its decompositions $\alpha_i \beta_i$. We can thus assume that T' memorizes these decompositions

in its state. Similarly, m_1, m_2 are known (they can be deduced from x_2, y_1, y_3, z_1 which are known), while $n_1, n_2, n_3 \in \mathbb{N}$ are unknown.

The execution of T' is decomposed into three different phases, switching from one to another one is done non-deterministically. T' has to simultaneously verify that its guesses (of bounded length words and of positions ℓ and ℓ') are correct, and produce the correct output word. We now describe the behavior of T':

1. before ℓ: T' replicates the output of the first run, and verifies that the second and third run produce words y_3 and z_1.

2. between ℓ and ℓ': this part needs more explanations. Our objective is that T' produces $x_1x_2y_1y_2y_3z_1z_2$ during this phase. Thanks to (†), we know that this word is of the form $t_1t_2^*t_3$. In order to produce the correct output word, it remains only to identify the right power of t_2. For simplicity, we assume we are in the sub case described by (‡). Thus, the output is exactly $t_1t_2^nt_3$ where $n = n_1 + n_2 + n_3 + m_1 + m_2 + 4$. T' immediately outputs $t_1t_2^{m_1+m_2+4}$. Then, while simulating the three runs of T and verifying that words x_1, y_2 and z_2 can be decomposed according to (‡), T' identifies the values of n_1, n_2 and n_3, and produces a new copy of the word t_2 each time one of these three counters is incremented. When T' decides (non-deterministically) to move to the last phase, it outputs the word t_3.

3. after ℓ': during this last phase, T' only has to replicate the output of the third run of T, and verify that the first and the second run produce the words x_2 and y_1 respectively.

It should be clear from this description that a pair $(u, v) \in [\![T]\!]$ can be produced by T' if, and only if, $(u, v) \models \mathcal{P}$, yielding the expected equality $[\![T']\!] = \{(u, v) \in [\![T]\!] \mid (u, v) \models \mathcal{P}\}$. As a consequence, we obtain that for an arbitrary ZNFT T, we have $[\![T']\!] \subseteq [\![T]\!]$, and the equality holds iff $T \models \mathcal{P}$. As we are considering *functions*, checking the equality of the relations amounts to check the following domain inclusion: $\mathrm{dom}(T) \subseteq \mathrm{dom}(T')$. This last property is decidable, yielding the expected result.

A decision procedure

Construction of $\mathsf{squeeze}(T)$ Let T be a f2NFT which is NFT-definable, we explain how to construct the f2NFT $\mathsf{squeeze}(T)$. By hypothesis, the transductions $\mathcal{L}_T(q_1, q_2)$ and $\mathcal{R}_T(q_1, q_2)$ are NFT-definable for all q_1, q_2 by NFT $L_T(q_1, q_2)$ and $R_T(q_1, q_2)$ respectively (they exist by Proposition 10 and Lemma 14). The main idea to define $\mathsf{squeeze}(T)$ is to non-deterministically (but repeatedly) apply $L_T(q_1, q_2)$, $R_T(q_1, q_2)$, or T, for some $q_1, q_2 \in Q$. The transducer $\mathsf{squeeze}(T)$ has two modes, Z-mode or T-mode. In T-mode, it works as T until it non-deterministically decides that the next zigzag is a z-motion from some state q_1 to some state q_2. Then it goes in Z-mode and runs $L_T(q_1, q_2)$ (or $\overline{R_T(q_1, q_2)}$) in which transitions to an accepting state have been replaced by transitions to state q_2 in T, so that $\mathsf{squeeze}(T)$ returns in T-mode. We easily obtain:

Proposition 11. *Let* $T \in$ f2NFT *such that* T *is* NFT*-definable. Then* $\mathsf{squeeze}(T)$ *is defined and equivalent to* T*.*

Let $T \in$ f2NFT. If T is NFT-definable, then the operator $\mathsf{squeeze}$ can be iterated on T while preserving equivalence with T, by the latter proposition. As for two-way automata,

if we denote by N the number of states of T, we know that T is N-crossing, and therefore it suffices to iterate $\mathsf{squeeze}$ N^2 times to remove all zigzags from accepting runs of T. We derive the following decision procedure: we set T_0 to T, and i to 0, and, while T_i satisfies the necessary condition stated in Lemma 15 and $i \leq N^2$, we increase i and set T_i to $\mathsf{squeeze}(T_{i-1})$. If the procedure exits the loops before reaching N^2, then T is not NFT-definable, otherwise it is NFT-definable by the NFT obtained by removing from T_{N^2} all its backward transitions. This concludes the proof of Theorem 29.

It is worth observing that the resulting decision procedure has non elementary complexity.

5.3 Beyond streamability: variable minimization

The results presented previously aim at characterizing streamable transformations. However, it is clear that not all transformations are streamable, consider for instance a transformation swapping two parts of a document at some unbounded distance. For some applications, the input is arbitrarily long and is accessed as a stream. The evaluation *must* then be done using a one-way machine. Rather than characterizing streamable transformations, a more challenging issue in this context is thus to identify the minimal amount of memory needed to evaluate a given transformation using a one-way machine.

We argue that the model of SST may provide means to address this problem. Indeed, let us recall that (copyless) SST recognize exactly regular string functions. As SST are deterministic finite-state automata with string variables, the evaluation of a transformation given by some SST T with k variables requires a memory that can be bounded by a constant (the number of states of T) plus k times the size of the input word (times the maximal length of a word used is some variable update). Minimizing the number of variables needed to encode a given transformation f gives thus information on the memory usage of the evaluation of f.

This problem, presented here for transducers, can easily be adapted to other classes of register automata as defined in [ADD+13]. Observe that this minimal number of variables heavily depends on the kind of operations allowed to update the variables. A first work in this direction has been published in [AR13]. The authors tackle this minimization issue for a class of deterministic automata equipped with integer valued variables (called additive cost-register automata, ACRA for short). The only update operation allowed is of the form $X := Y + c$, where X, Y denote variables and $c \in \mathbb{Z}$. In particular, it is not possible to reset a variable ($X := c$) nor to combine two variables ($X := Y + Z$, with X, Y, Z variables).

We will present an original approach to variable minimization, based on the twinning property. To be coherent with the setting described in this thesis, we present our results for transducers but, as we will discuss at the end of this section, our results can be applied to a more general setting. These results have not been published yet but can be found in [12].

Right-appending SST and rational string functions The update operations of SST are of the form $X := YZ$ or $X := uYv$, where X, Y, Z are variables and $u, v \in \Sigma^*$. The copyless restriction forbids the use of the same variable in different right hand sides.

As we have proven in Theorem 26, the class of right-appending SST, obtained by only considering updates of the form $X := Yu$, without the copyless restriction, is expressively equivalent to rational string functions.

We denote by ra-SST the class of right-appending SST, and by ra-SST$_k$ the class of right-appending SST with k variables. We present a solution to the following minimization

problem:
Input: A fNFT T and a natural number k
Question: Is the function $[\![T]\!]$ realized by an element of the class ra-SST$_k$?

It is easy to observe that the expressive power of the class ra-SST$_1$ is exactly the class of subsequential functions. As this class is decidable, using the twinning property, the above decision problem is decidable for $k = 1$. In addition, this class is also characterized by the co-called bounded variation property introduced by Choffrut in [Cho77]. Based on these observations, we will introduce a twinning property of order k, and a k-bounded variation property, that will both characterize the class ra-SST$_k$.

A twinning property of order k Recall that the twinning property requires that two runs "synchronized" on some loop should not diverge, meaning that they can be simulated by a single run. This situation is depicted on Figure 5.3, describing two loops on the input word u_2, both reachable from some initial state by a run on the input word u_1. In this case, the twinning property requires that the equality $\mathsf{delay}(v_1, w_1) = \mathsf{delay}(v_1v_2, w_1w_2)$ holds. When proving that the twinning property allows to determinize a functional transducer, one shows that the distance between the outputs of two runs over the same input word is bounded, thanks to this twinning property.

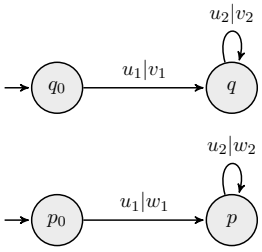


Figure 5.3: The classical twinning property

In order to generalize this idea, we introduce a twinning property of order k, which intuitively requires that for arbitrary $k + 1$ runs synchronized on k loops, there should exist two runs which are "close", meaning that they can be simulated by a single one. As for the classical twinning property, we consider a trimmed transducer. The formal definition is:

Definition 9. *A functional* NFT *satisfies the twinning property of order k (denoted by TP_k) if for all $k + 1$ runs $\rho_0, \ldots, \rho_k$ as described on Figure 5.4, there exist $j \neq \ell$ such that for all $i \in [1, k]$, we have:*

$$\mathsf{delay}(\alpha_1^j \cdots \alpha_i^j, \alpha_1^\ell \cdots \alpha_i^\ell) = \mathsf{delay}(\alpha_1^j \cdots \alpha_i^j \beta_i^j, \alpha_1^\ell \cdots \alpha_i^\ell \beta_i^\ell).$$

A k-bounded variation property The bounded variation property is defined on functions and is thus a machine independent property: whenever two NFT are equivalent, either both or none of them satisfy this property.

Given a partial mapping $f : \Sigma^* \rightharpoonup \Sigma^*$, the bounded variation property introduced by Choffrut in [Cho77] states that for every $N \in \mathbb{N}$, there exists $n \in \mathbb{N}$ such that for all $w, w' \in \Sigma^*$ such that $f(w), f(w')$ are defined, if $d(w, w') \leq N$, then $d(f(w), f(w')) \leq n$. Intuitively, this property states that whenever two words only differ by a small suffix, so do their images by f. This corresponds to the intuition that the function can be expressed by means of a right-appending SST with a single variable (a behavior can be deduced from the other one).

When lifting this property to functions that can be expressed using at most k variables, we consider $k + 1$ input words pairwise close, and require that two of them must have close images by f. We obtain the following definition:

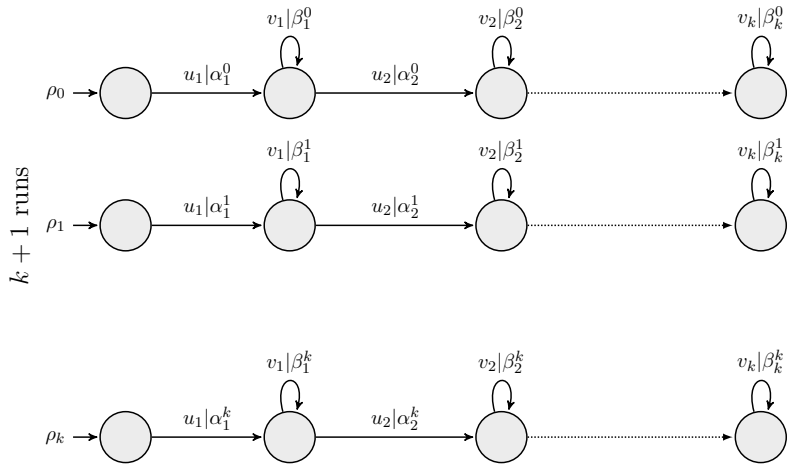


Figure 5.4: The twinning property of order k

Definition 10. *Let $f : \Sigma^* \rightharpoonup \Sigma^*$ and a number k. We say that f satisfies the* k-bounded variation property *iff for all naturals $N > 0$, there is a natural n such that for all words $w_0, \ldots, w_k \in \mathrm{dom}(f)$, if for all $0 \leq i, j \leq k, d(w_i, w_j) \leq N$, then there is $i \neq j$ such that $d(f(w_i), f(w_j)) \leq n$.*

Generalizing the reasoning used for the twinning property, we manage to prove the following key lemma, which corresponds to the bound on the distance between different runs on the same input word:

Lemma 16. *If a functional* NFT T *satisfies TP_k then for all words u and all runs $q^j \xrightarrow{u|\alpha^j} p^j$ with $j \in \{0, \ldots, k\}$ (where q^j are initial states), there are $j \neq \ell$ such that $d(\alpha^j, \alpha^\ell) \leq 2n^{k+1}M$ where n is the number of states of T and M is the maximal length of an output word.*

Conversely, we prove that delays can grow arbitrarily if the transducer does not satisfy the twinning property:

Lemma 17. *If a functional* NFT T *does not satisfy TP_k, then for all positive integers m, there are a word u and $k+1$ runs $q^j \xrightarrow{u|\alpha^j} p^j$ with $j \in \{0, \ldots, k\}$ (where q^j are initial states), such that for all $j \neq j'$, $d(\alpha^j, \alpha^{j'}) > m$.*

Characterization of ra-SST$_k$ The result of Choffrut states that a functional NFT T is DFT-definable iff T satisfies the twinning property iff $[\![T]\!]$ satisfies the bounded variation property. We state now our generalization of this result:

Theorem 30 ([12])**.** *Let T be a* fNFT *and $k \in \mathbb{N}$. The following assertions are equivalent:*

(i) T satisfies the twinning property of order k,

(ii) $[\![T]\!]$ satisfies the k-bounded variation property,

(iii) $[\![T]\!]$ is realized in the class ra-SST$_k$.

Sketch of proof. The proof proceeds as follows: we start with the equivalence $(i) \iff (ii)$, and then show the equivalence $(i) \iff (iii)$.

Observe that the equivalence $(i) \iff (ii)$ corresponds to a machine independent characterization of the twinning property of order k. This equivalence follows rather easily from Lemmas 16 and 17.

Consider now the equivalence $(i) \iff (iii)$. We first prove that TP_k is a necessary condition for a functional NFT to be definable by some ra-SST_k. Indeed, if it is violated, then for any integer N, we can find an input word u such that there are at least $k+1$ runs on u whose output words are pairwise at distance at least N. Each of these runs can be completed using some input word w_i into an accepting run. Considering the configuration reached by some ra-SST with k variables after reading the input u, we can prove that one of these runs will not be simulated faithfully.

Conversely, we consider the implication $(i) \Rightarrow (iii)$. First, we recall that every fNFT can be transformed into an equivalent unambiguous NFT [EM65]. By the equivalence $(i) \iff (ii)$ showing that the twinning property is machine independent, we know that the equivalent unambiguous transducer satisfies the TP_k. We sketch now a construction that, given an unambiguous NFT T satisfying TP_k, returns an equivalent ra-SST_k S. The standard technique to build a deterministic machine consists in using a subset construction, as this is done for instance to build a DFT. In our setting, we intuitively need to partition the set of runs in at most k subsets, the runs belonging to the same element of the partition being represented by the same variable of the ra-SST. Therefore, we use a notion similar to the notion of separability introduced in [AR13]: there exists a constant N depending on k such that if two runs have their outputs which are at a distance larger than N, then these two runs cannot be simulated using the same variable. This constant can be derived from Lemma 16. This is used to justify the fact that we can store in the states of S the delays between each pair of runs, provided they are smaller than N.

Let Q be the set of states of T (w.l.o.g., we assume T trimmed). Formally, states of S are square matrices whose entries belong to $\left(\Sigma^{\leq N}\right)^2 \cup \{\omega\}$, and which are indexed by subsets of Q. For such a matrix M, its set of indices is denoted by d_M and called the domain of M. The initial state is the matrix M such that d_M is the set of initial states of T, and all the entries are equal to $(\varepsilon, \varepsilon)$. The transition function of S is defined in the expected way with the following intuition: the state M reached in S after reading some input word u is such that:

- d_M is exactly the set of states of T reachable along u,
- given $p, q \in d_M$, $M[p,q] = \omega$ iff there exists a prefix u' of u such that the two runs $i_1 \xrightarrow{u'|v_1} p$ and $i_2 \xrightarrow{u'|v_2} q$ on the word u' reaching respectively p and q from some initial states, satisfy the condition $d(v_1, v_2) > N$,
- given $p, q \in d_M$ such that $M[p,q] \neq \omega$, consider the two runs $i_1 \xrightarrow{u|v_1} p$ and $i_2 \xrightarrow{u|v_2} q$ on the word u reaching respectively p and q from some initial states, then we have $M[p,q] = \mathsf{delay}(v_1, v_2)$.

Observe that as T is unambiguous and trimmed, for every $p \in d_M$, there exists a unique run $i \xrightarrow{u|v} p$ such that i is an initial state.

Thanks to Lemma 16, we know that for every such reachable matrix M, we are able to select at most k states $p_i, \ldots, p_k$ such that for every $q \in d_M$, there exists $i \in \{1, \ldots, k\}$

with $M[q, p_i] \neq \omega$. This property is used to define variable updates in a consistent way so as to ensure the correctness of the construction. More precisely, we actually derive first a right-appending SST whose updates use elements in the free group on Σ. In a second step, we show that as the function realized is a function from strings to strings, it is possible to modify this SST so as to only use updates in Σ^*. □

Decidability of the twinning property of order k We will sketch the proof of the following theorem:

Theorem 31 ([12])**.** *Let T be a* fNFT *and k be a number given in unary. Determining whether T satisfies the TP_k is in* PSpace*.*

Proof. We first recall how Weber and Klemm show in [WK95] that the twinning property can be decided in PTime. They first show that a fNFT T does not satisfy the TP iff there exist two runs as in the premises of the TP (see Figure 5.3) such that one of the two following conditions holds:

- either $|v_2| \neq |w_2|$,
- or v_1 and w_1 have a mismatch, *i.e.* a position on which they differ.

Using a similar reasoning, we prove the following lemma describing how a witness of the violation of the TP_k can be exhibited:

Lemma 18. *Let T be a* fNFT *violating the TP_k. Then there exist:*

- *states $\{q_i^j\}_{0 \leq i \leq m, 0 \leq j \leq k}$ with $k \leq m \leq k^2$, and q_0^j initial and q_k^j co-accessible for all j,*
- *words $u_1, \ldots, u_m$, $v_1, \ldots, v_m$ such that there are $k+1$ runs satisfying for all $0 \leq j \leq k$, for all $1 \leq i \leq m$, $q_{i-1}^j \xrightarrow{u_i|\alpha_i^j} q_i^j$ and $q_i^j \xrightarrow{v_i|\beta_i^j} q_i^j$*

and such that for all $0 \leq j < j' \leq k$:

- *either there exists $1 \leq i \leq m$ such that $|\beta_i^j| \neq |\beta_i^{j'}|$,*
- *or there exists $1 \leq i \leq m$ such that $|\beta_i^j| = |\beta_i^{j'}| \neq 0$, the words $\alpha_1^j \ldots \alpha_i^j$ and $\alpha_1^{j'} \ldots \alpha_i^{j'}$ have a mismatch, and the runs $q_0^j \xrightarrow{u_1 \ldots u_i} q_i^j$ and $q_0^{j'} \xrightarrow{u_1 \ldots u_i} q_i^{j'}$ are such that for every prefixes $q_0^j \xrightarrow{u|\alpha}$ and $q_0^{j'} \xrightarrow{u|\alpha'}$ of these runs on the same input, we have $d(\alpha, \alpha') \leq Mn^{k+1}$.*

This allows to derive a non-deterministic procedure running in polynomial space: we first guess the global structure of the witness, *i.e.* the value of m, the vectors of states $(q_i^j)_j$ for each $0 \leq i \leq m$, and, for every pair $0 \leq j < j' \leq k$, which of the two cases holds. First the reachability between two vectors of states can be checked in polynomial space. Second, it is possible to decide in polynomial space the existence of a cycle around a given vector of states such that the length of output words on j-th and j'-th runs are different. Indeed, one can bound the length of such cycles by $2|Q|^{k+1}$. Last, we explain how to verify the existence of the mismatches. Given a pair of runs (ρ, ρ'), we proceed as follows: one non-deterministically guesses ρ and ρ' and stores the difference between the lengths of the outputs of the two runs. Non-deterministically, one can record the letter produced by the run that is ahead (say ρ). Then one continues the simulation until ρ' catches up ρ, and checks that the letter produced by ρ' is different. This can be achieved in polynomial space using the fact that the distance between ρ and ρ' is always bounded by Mn^{k+1}. □

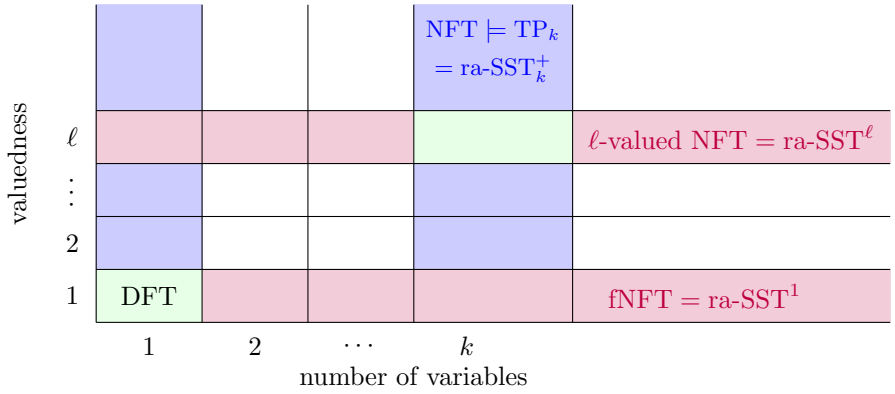


Figure 5.5: Hierarchy for finite-valued rational relations. The class with a valuedness equal to ℓ and k variables corresponds to ℓ-valued NFT satisfying the TP$_k$ and to the class ra-SST$^\ell_k$.

Corollary 4 ([12])**.** *Let S be an* ra-SST*, and k be a number (coded in unary). Then determining whether there exists $S' \in$ ra-SST$_k$ such that $[\![S]\!] = [\![S']\!]$ is* PSPACE-C*.*

The hardness follows from that of a similar problem considered in [AR13]. The membership result follows from the polynomial translation of ra-SST into NFT, and from Theorem 31.

Extension to relations Until now, we focused on functions, rather than general relations. It may however be useful to relax this constraint. It is well-known that finite-valued relations enjoy very good decidability properties, as functions do. The main result is the decidability of the equivalence of two finite-valued rational relations.

The model of SST we have presented only recognizes functions. A non-deterministic extension, which thus allows to recognize relations, has been considered in [AD11]. It is a quite expressive framework, incomparable with non-deterministic MSO transductions, and that may define infinite-valued transductions.

We suggest here an alternative extension of SST that allows to recognize finite-valued relations. We present it in the setting of right-appending SST, but it could similarly be used for general SST. The idea is simply to modify the output function as follows: instead of associating to some state p a variable X (or more generally a string involving variables, such as $uYvZw$), we may associate to state p a finite set of such expressions. Formally, considering some SST $T = (Q, q_0, \delta, \mathcal{X}, \rho, F)$, the output function F is now of the type $F : Q \rightharpoonup 2^{(\mathcal{X} \cup B)^*}$. Given such a SST T, letting $\mathsf{val}(T) = \max\{|F(q)| \mid q \in Q, F(q) \text{ is defined}\}$, we immediately have that the relation defined by T is $\mathsf{val}(T)$-valued. We now denote by ra-SST^p_k the class of ra-SST using k variables, and output functions of size at most p. ra-SST^p is the union of ra-SST^p_k over k, and ra-SST^+_k is the union of ra-SST^p_k over p.

The generalized twinning property as well as the generalized bounded variation property presented before can be adapted to the setting of finite-valued NFT, and we can extend Theorem 30 to finite-valued relations. We can also show that our decision procedure is valid for this more general setting, with the same theoretical complexity, and thus we obtain that the minimization of variable is PSPACE-C for the class ra-SST with extended output functions.

As a corollary, we obtain a decidable and strict hierarchy for the whole class of finite-valued rational relations depicted on Figure 5.5.

Extension to infinitary groups In [12], these results are presented not only for the monoid of finite words equipped with concatenation, but also in the setting of so-called infinitary groups [FGR15]. This includes the classical settings $(\mathbb{N}, +, 0)$, $(\mathbb{Z}, +, 0)$, but also groups allowing to encode sum automata, discounted sum automata and average automata, as shown in [FGR15]. In particular, we obtain this way a generalization of the minimization result stated in [AR13] for so-called additive cost regular functions.

5.4 First-order definable transformations

Based on the automata-logic connection known for regular languages, several works have studied similar connections for other logics. In particular, it has been shown that languages definable using the first-order logic (FO) enjoy several other characterizations: star-free expressions, aperiodic syntactic monoid, linear temporal logic, counter-free automata... These logical fragments are also very useful to study circuits synthesis [Imm87].

These positive results have motivated the study of similar connections between first-order definable string transformations (FOT) and restrictions of state-based transducers models. Two recent words provide such characterizations for SST and 2DFT respectively [FKT14, CD15]. To this end, the authors introduce a notion of monoid for these transducers, and prove that FOT is expressively equivalent to transducers with aperiodic monoid.

Although SST (resp. aperiodic 1-bounded SST) and 2DFT (resp. aperiodic 2DFT) are known to be equivalent as they are both equivalent to MSOT (resp. to FOT), there is no direct transformation from SST to 2DFT, as it can be observed from Figure 4.5, and thus the only existing transformation has non-elementary complexity [2]. In this section, we present direct constructions from SST to 2DFT (and back), and prove that these constructions preserve the aperiodicity of the underlying monoid. We hope that our direct translations provide better intuition about the workings of the models concerned and may help understand them better.

In addition, we also provide a transformation from k-bounded SST to 1-bounded SST that preserves aperiodicity. As a consequence, this yields that FOT is not only equivalent to the class of aperiodic 1-bounded SST, but also to the larger class of aperiodic bounded SST. These results have been obtained with Luc Dartois recently and are not published yet, but they are presented in [11]. These results are summarized in Figures 4.5 (general case) and 5.6 (case of aperiodic transducers).

A *first-order definable string transformation* (FOT) is an MSOT $T = (C, \phi_{\text{dom}}, (\phi^c_{\text{pos}})_{c\in C}, (\phi^c_a)_{c\in C, a\in\Sigma}, (\phi^{c,d}_{\preceq})_{c,d\in C})$ where ϕ_{dom} a FO sentence over $\mathcal{S}_\Sigma$, and ϕ^c_{pos}, ϕ^c_a and $\phi^{c,d}_{\preceq}$ are FO formula over $\mathcal{S}_\Sigma$ with respectively one, one and two free first-order variables.

Example 11. *We consider the example given in [CD15]. Let Σ be the alphabet $\{a, b\}$. We consider the function $f : \Sigma^* \to \Sigma^*$ that maps the input word $u = a^{k_0}ba^{k_1}b\dots a^{k_n}$ to the output word $f(u) = a^{k_0}b^{k_0}a^{k_1}b^{k_1}\dots a^{k_n}b^{k_n}$. For instance, we have $f(aababb) = aabbab$.*

A FOT *T realizing this function uses two copies of the input structure, identifies blocks of a, and uses them to produce the blocks of a's (using the first copy) and the new blocks of b's*

[2] We have recently been aware of the unpublished work [Led13] that proposes an SST to 2DFT construction similar to ours.

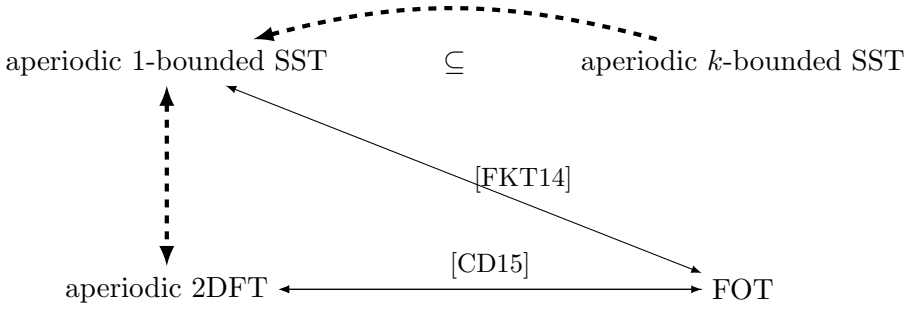
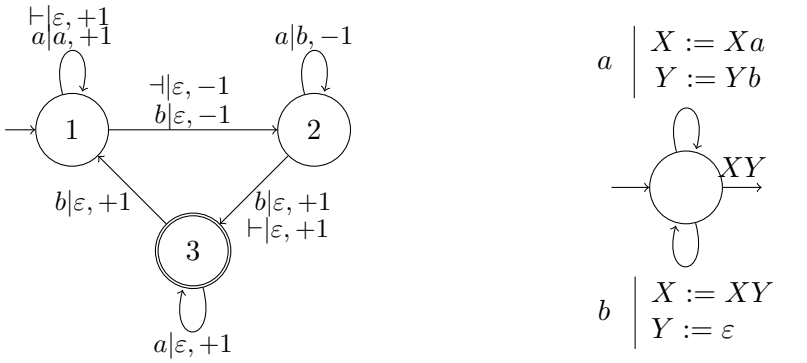


Figure 5.6: Tranformations between aperiodic 2DFT, FOT and aperiodic variants of SST. The original constructions presented in this section are depicted by thick dashed arrows.



(a) An aperiodic 2DFT

(b) An aperiodic SST

Figure 5.7: Two transducers realizing the function f.

(using the second copy). Formally, we let $C = \{1, 2\}$, $\phi_{\text{dom}} = \texttt{true}$, $\phi_a^1(x) = \phi_b^2(x) = L_a(x)$ *and* $\phi_b^1(x) = \phi_a^2(x) = \texttt{false}$, *and the order formula* $\phi_{\preceq}^{i,j}$ *are defined by:*

- $\phi_{\preceq}^{i,i}(x, y) = x \preceq y$ *for* $i = 1, 2$,
- $\phi_{\preceq}^{1,2}(x, y) = x \preceq y \vee (\forall z \cdot y \preceq z \preceq x \Rightarrow L_a(z))$,
- $\phi_{\preceq}^{2,1}(x, y) = \exists z \cdot x \preceq z \preceq y \wedge L_b(z)$.

An aperiodic 2DFT *and an aperiodic (copyless)* SST *realizing* f *are depicted respectively on Figures 5.7.(a) and 5.7.(b).*

A monoid for 2DFT The monoid of a 2DFT is defined as the one of its underlying 2DFA, *i.e.* ignoring the output words. Though this definition may seem surprising, it is sufficient to obtain an equivalence with FOT. However, as we will discuss later, it is problematic in order to obtain an algebraic characterization of FOT.

As for DFA, the transition monoid of a 2DFA A is defined as the quotient of the free monoid Σ^* by the congruence relation $\sim_A$ which is such that two words u, v verify $u \sim_A v$ iff they induce the same behaviors in A. While for a DFA such behaviors are only left-to-right

runs, in 2DFA we have to consider four types of behaviors, namely left-to-left, left-to-right, right-to-right and right-to-left. Formally, given a 2DFA A and a non-empty word u, we define for instance the left-to-right behaviors [3] of A on the word u, denoted $\mathrm{bh}^A_{\ell r}(u)$, as the set of pairs of states (p, q) of A such that *the* run of A starting in state p on the first letter of u, and leaving u by the right, reaches state q. Sets $\mathrm{bh}^A_{\ell\ell}(u)$, $\mathrm{bh}^A_{rr}(u)$ and $\mathrm{bh}^A_{r\ell}(u)$ are defined similarly. We then define $u \sim_A v$ iff $\mathrm{bh}^A_{xy}(u) = \mathrm{bh}^A_{xy}(v)$ for all $x, y \in \{\ell, r\}$. We denote by M_A the resulting monoid. Recall that a monoid M is *aperiodic* if there exists a positive integer α such that for every element $m \in M$, we have $m^\alpha = m^{\alpha+1}$. The minimal such α is called the aperiodicity index of M.

The transition monoid M_T of a 2DFT T is defined as the transition monoid M_A of its underlying 2DFA A. We say that a 2DFT T is *aperiodic* if the transition monoid M_T is aperiodic.

Theorem 32 ([CD15])**.** FOT *and aperiodic* 2DFT *are expressively equivalent.*

A monoid for SST The monoid M_T of an SST T (non-necessarily copyless) with set of states Q and set of variables $\mathcal{X}$ is defined using the notion of flow of variable. We say that two words u and v are equivalent w.r.t. T, denoted $u \sim_T v$, iff for all p, q, X, Y, k, we have $(p, Y) \overset{u}{\rightsquigarrow}_k (q, X) \iff (p, Y) \overset{v}{\rightsquigarrow}_k (q, X)$. The monoid M_T is defined as the quotient of Σ^* by the congruence $\sim_T$. Elements of M_T can be represented by square matrices with indices in $Q \times \mathcal{X}$, and whose entries are elements of $\mathbb{N}$. We denote by η_T the morphism of monoid from Σ^* to M_T. We say that T is aperiodic iff M_T is aperiodic.

It is worth observing that T is k-bounded iff for every matrix $m \in M_T$, all its entries are less than or equal to k.

Theorem 33 ([FKT14])**.** FOT *and aperiodic* 1*-bounded* SST *are expressively equivalent.*

Observe that this result does not give the equivalence with copyless aperiodic SST, nor with k-bounded aperiodic SST for an arbitrary k.

From SST **to** 2DFT We provide the first direct translation of SST to 2DFT:

Theorem 34 ([11])**.** *Let T be a* 1*-bounded* SST *with n states and m variables. We can construct an equivalent* 2DFT *T_{2W} with $O(m2^{m2^m}n^n)$ states. If T is copyless, then T_{2W} has $O(mn^n)$ states. In addition, if T is aperiodic, so is T_{2W}.*

Let $T = (Q, q_0, \delta, \mathcal{X}, \rho, F)$ be a 1-bounded SST. The transducer T_{2W} will follow the output structure (see Figure 5.8) of T and construct the output as it appears in the structure. To make the proof easier to read, we define T_{2W} as the composition of a left-to-right sequential transducer T_1, a right-to-left sequential transducer T_2 and a deterministic 2-way transducer T_3. Remark that this proves the result as 2DFT are closed under composition. The transducer T_1 does a single pass on the input and enriches it with the transition used by T in the previous step (the new alphabet is composed of pairs $(a, t) \in \Sigma \times \delta$. The second transducer uses this information, and enriches the input word with the set of variables corresponding to the variables that will be produced from this position. The new alphabet is thus composed of triples $(a, t, S) \in \Sigma \times \delta \times 2^{\mathcal{X}}$. The last transducer will then follow the output structure of T

[3] We define these objects as relations but when the two-way automaton is deterministic it is worth observing they are functions.

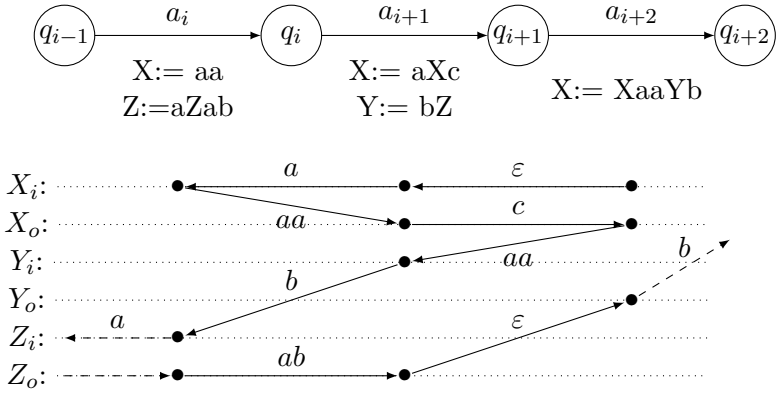


Figure 5.8: The output structure of a partial run of an SST. Formally, this structure can be defined as a graph in which edges are labeled by finite strings, and that allows to describe how the final output of the SST is obtained. See [AČ10a, AFT12] for details.

and use the transitions t of the enriched alphabet to rewind the run of T when the reading head moves left. The set of variables S will be used to clear the non determinism due to the 1-bounded property. Indeed, it may be the case, when executing some transition t, that some variable X is used to define two (or even more) variables Y and Z. However, due to the 1-boundedness property of T, at most one of Y and Z will be part of the final output. The set S which is now part of the input alphabet of T_3 allows to resolve this ambiguity, and to follow the good edge in the output structure of T.

Observe that if T is copyless, this last ambiguity does not exist, and thus we do not need to use transducer T_2. The statements on the size of T_{2W} follow from the study of the construction proposed in [CJ77] for the composition of a 2DFT with a DFT.

Last, we argue on the aperiodicity of T_{2W}. First, thanks to results proved in [CD15], it is sufficient to prove that each of the transducer involved in the composition is aperiodic. This is rather trivial for T_1 and T_2. Concerning T_3, this property is more difficult to prove. To this end, we need to show that for α large enough, the behaviors of u^α and $u^{\alpha+1}$ are the same in T_3. Informally, runs in T_3 follow the output structure of T, and we thus need to show that this output structure is "aperiodic". The definition of aperiodicity for T exactly ensures that for α large enough, for any u, q, X, the variables appearing in $\sigma_{q,u^\alpha}(X)$ and in $\sigma_{q,u^{\alpha+1}}(X)$ are the same (with the same multiplicity, which in the case of a 1-bounded SST is necessarily equal to 1). However, to prove the aperiodicity of the structure, we need a stronger property, *i.e.* the equality [4] $\pi_{\mathcal{X}}(\sigma_{q,u^\alpha}(X)) = \pi_{\mathcal{X}}(\sigma_{q,u^{\alpha+1}}(X))$ (informally, the same variables appear *in the same order*). We prove that this property holds true (with a possibly larger aperiodicity index), which concludes the proof of Theorem 34.

Proposition 12. *Let T be an aperiodic k-bounded* SST *with m variables of aperiodicity index α_T. Then for any u, q, X, and for any $\alpha \geq \alpha_T + (k+1)m$, we have:*

$$\pi_{\mathcal{X}}(\sigma_{q,u^\alpha}(X)) = \pi_{\mathcal{X}}(\sigma_{q,u^{\alpha+1}}(X))$$

[4] $\pi_{\mathcal{X}}$ is the projection from $(\mathcal{X} \cup \Sigma)^*$ onto $\mathcal{X}^*$.

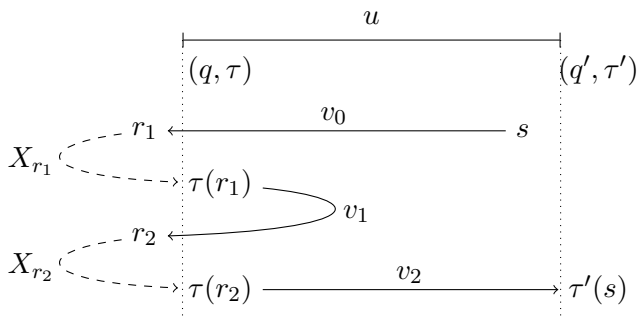


Figure 5.9: The flow of variables in the SST T_{SST}.

From 2DFT **to** SST We present a simplification of a construction introduced in [AFT12] in the more general context of infinite words. We prove that this construction preserves aperiodicity.

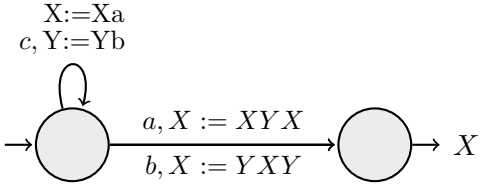
Theorem 35 ([11])**.** *Let T be a* 2DFT *with n states. We can construct an equivalent* 1*-bounded* SST T_{SST} *with $O(n^n)$ states and $n+1$ variables. In addition, if T is aperiodic, so is T_{SST}.*

The construction relies on Shepherdson's construction that allows to transform a 2DFA into a DFA [She59]. To this end, this construction stores in its states the right-to-right behaviors on the word read so far in order to be able to compute the global left-to-right behavior. More formally, states of the resulting DFA are pairs $(q, \tau) \in S \times S^S$, where S denotes the set of states of the input 2DFA A. After having read the input word u, the DFA reaches the state (q, τ) where q is the unique state such that $(i, q) \in \mathrm{bh}^A_{\ell r}(u)$ (i denotes the initial state of A), and $\tau = \mathrm{bh}^A_{rr}(u)$. In the context of transducers, one can use a string variable X_r to store the output word produced by the behavior starting in r (see Figure 5.9). One can observe that the SST constructed this way is 1-bounded (it is not copyless due to a kind of non-determinism on which right-to-right behavior will be part of the final accepting run).

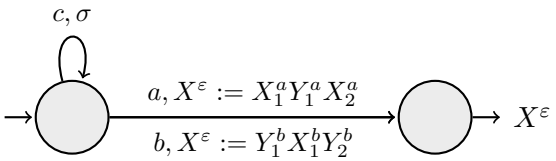
Assuming that T is aperiodic, the aperiodicity of T_{SST} is a direct consequence of the following property, which is proven by induction. Given a run $(q, \tau) \xrightarrow{u} (q', \tau')$ in T_{SST} inducing a substitution σ, we claim that we have, for all states s:

- if the run of T on u starting in s on the last letter of u leaves u on the right producing some word v, then $\sigma(X_s) = v$;
- if the run of T on u starting in s on the last letter of u leaves u on the left, then $\sigma(X_s) = v_0 X_{r_1} v_1 \dots X_{r_n} v_n$, where $n \geq 1$ and:
  1. $(s, r_1) \in \mathrm{bh}_{r\ell}(u)$ produces v_0
  2. $(\tau(r_i), r_{i+1}) \in \mathrm{bh}_{\ell\ell}(u)$ produces v_i for all $i < n$
  3. $(\tau(r_n), \tau'(p)) \in \mathrm{bh}_{\ell r}(u)$ produces v_n

The situation corresponding to the second case is illustrated in Figure 5.9.



(a) A 2-bounded SST recognizing the function $f : c^n\alpha \mapsto a^n b^n a^n$ if $\alpha = a$, and $b^n a^n b^n$ if $\alpha = b$.



(b) An equivalent 1-bounded SST. The update σ is defined by $\sigma(X_i^\alpha) = X_i^\alpha a$ and $\sigma(Y_i^\alpha) = Y_i^\alpha b$ for $i \in \{1, 2\}$ and $\alpha \in \{a, b\}$.

Figure 5.10: Removing bounded copy

From k-bounded SST to 1-bounded SST The existing construction from k-bounded to 1-bounded, presented in [AFT12], builds a copyless SST. This construction is rather complex, involving so-called dependency graphs, and it is thus difficult to prove that it preserves aperiodicity (though we conjecture this is the case). We present an alternative construction that, given a k-bounded SST, builds an equivalent 1-bounded SST (but not copyless). We prove that this construction preserves aperiodicity.

Theorem 36 ([11])**.** *Let T be a k-bounded* SST *with n states and m variables. We can construct an equivalent* 1*-bounded* SST *T_{SST} with $n2^N$ states and mkN variables, where $N = O(n^n(k+1)^{nm^2})$ is the size of the monoid M_T of T. In addition, if T is aperiodic, so is T_{SST}.*

Proof. In order to move from a k-bounded SST to a 1-bounded SST, the natural idea is to use copies of each variable. However, we cannot maintain k copies of each variable all the time: suppose that X flows into Y and Z, which both occur in the final output. If we have k copies of X, we cannot produce in a 1-bounded way k copies of Y and k copies of Z. We thus limit, for each variable X, the number of copies of X we maintain. In order to get this information, we use a look-ahead information on the suffix of the run.

Suppose that we know at each step what is the substitution induced by the suffix of the run at the current state, then, for each variable X we know the value of the integer n such that X will be involved exactly n times in the final output. This gives us the precise number of copies of each variable we have to maintain, and we can use these copies to produce the substitution in a copyless manner.

As an illustration, consider the SST depicted on Figure 5.10.(a). It is obviously 2-bounded, and there are actually only 2 different cases corresponding to non-trivial suffixes of input words: either X flows twice in X, and Y flows once in X (if the last letter of the input word is a) or Y flows twice in X, and X flows once in X (if the last letter of the input word is b). By guessing which of the two cases holds, one can produce the adequate number of copies. As we cannot guess, because our aim is to build a deterministic automaton, we have to consider the two possible cases. We thus consider variables X_1^a, X_2^a, Y_1^a corresponding to the first case (last letter is a) and similarly X_1^b, Y_1^b, Y_2^b for the second case. We do not try to share variables used for the different cases. Variable X^ε corresponds to the trivial suffix ε. The resulting equivalent 1-bounded SST is depicted on Figure 5.10.(b).

One can observe that this information on flow of variables is contained in the monoid M_T. However, assuming that the image of the current suffix av in M_T is some element m, one cannot, in a deterministic left-to-right fashion, determine the value m' of the next suffix

v. There can indeed be several candidates $m_1, \dots, m_\ell$. As we have to build a deterministic input automaton, we will use a subset construction on these possible elements of M_T.

More formally, the 1-bounded SST T_{SST} has the set of states $Q' = \{(q, S) \mid q \in Q, S \subseteq M_T\}$. The transitions are of the form $(q, S) \xrightarrow{a} (q', S')$ where $q' = \delta(q, a)$ and $S' = \{m \in M_T \mid \eta_T(a)m \in S\}$.

We use variables of the form X_i^m for $X \in \mathcal{X}$, $1 \leq i \leq k$ and $m \in M$. The successors of an element $m \in M_T$ by letter a are the elements $m_1, \dots, m_\ell \in M_T$ such that $m = \eta_T(a)m_j$. The variables indexed by m may thus flow to variables indexed by $m_1, \dots, m_\ell$. As we stored enough copies for m, we can produce enough copies for each m_j. Observe also that the update from m to m_j is copyless (for each j). As variables associated with m_j and $m_{j'}$ with $j \neq j'$ are never recombined (the current suffix belongs to exactly one of the m_j's), we get the 1-boundedness of the construction.

The aperiodicity of the produced SST comes from the fact that its runs are of the form $(q, S) \xrightarrow{u} (q', S')$ where $q \xrightarrow{u}_T q'$ and $S' = \{m \in M_T \mid \eta_T(u)m \in S\}$, which holds by construction. The update for such a run is then the update of T over the run $q \xrightarrow{u}_T q'$, where variables are labelled by elements from S and S' and numbered accordingly. As M_T is aperiodic, this implies that the transition function of the new SST is aperiodic, as well as its update function. □

As a corollary, we obtain that for every integer $k \in \mathbb{N}_{>0}$, the class of aperiodic k-bounded SST is expressively equivalent to first-order definable string-to-string transductions. This constitutes a new result, as the equivalence was only known for the class of 1-bounded aperiodic SST (see Figure 5.6). Combining with existing results, we obtain:

Corollary 5 ([FKT14, CD15],[11])**.** *Let $f : \Sigma^* \to \Sigma^*$ be a string-to-string function. The following assertions are equivalent:*

- *f is realized by an aperiodic* 2DFT*,*
- *f is realized by an aperiodic* 1*-bounded* SST*,*
- *f is realized by an aperiodic k-bounded* SST *for some k,*
- *f is realized by an* FOT*.*

5.5 Perspectives

The decidability of rational functions among regular ones presented in Section 5.2 is a promising result, since we introduced combinatorics techniques that could be useful for other decidability problems. Moreover, there are also direct extensions which are worth being studied: one could consider other semi-rings (our proof implies the known result in the case of a commutative semi-ring), infinite words... Another research direction concerns the complexity of our decision procedure. Due to our proof approach (Rabin and Scott procedure), we obtain a non elementary procedure. However, we only have a PSpace lower bound, and we conjecture that the problem is elementary, more precisely it may be decidable in 2-ExpTime. We have already studied the possibility of an adaptation of the construction of Shepherdson [She59], but this approach is highly technical. Recently, Anca Muscholl and other colleagues in Bordeaux have proposed in [BGMP15] a procedure in order to build a one-way transducer equivalent to

a given sweeping transducer, when it exists (a sweeping transducer is a deterministic two-way transducer that only turn back at the extremities of the input word). Last, another line of works consists in trying to directly characterize the class DFT inside that of 2DFT. This class enjoys indeed several characterizations, as explained in Section 5.3.

Concerning our generalized twinning property presented in Section 5.3, there are also some promising extensions, as we have a very general characterization of the required number of variables (for consistency with the rest of this document, we presented it in the setting of transducers, but it holds for more general weighted automata). The long-term objective is of course to solve the minimization problem for the general setting of copyless SST. However, this requires the development of new tools, as we have to deal with concatenation of variables. A different (and probably easier) objective is to lift our results from functions on words to functions defined on nested-words, as described in the next Chapter. These functions would be given as weighted visibly pushdown automata. Indeed, we have identified a twinning property (the matched TP presented in Section 6.4) that characterizes the weighted visibly pushdown automata that can be determinized. It seems thus possible to apply our proof approach to this setting, though we have already identified some notable differences. For instance, while unambiguous weighted automata are can be encoded using finitely many variables, this does not hold for unambiguous weighted automata on nested words. This leads to another interesting decision problem.

Very recently, a variation of the twinning property has been introduced in [JF15] in order to characterize rational relations that can be represented as a finite union of sequential transducers (the class of so-called multi-sequential relations). One can notice that this class also owns a nice characterization as the restriction of right-appending SST (considered with generalized output function) to updates of the form $X := Xu$ (instead of $X := Yu$). An open problem for the class of multi-sequential relations consists in minimizing the size of the union. A natural perspective thus consists in identifying an adequate twinning property characterizing the number of variables in the previous subclass of right-appending SST.

Last, a very ambitious project is to develop algebraic tools characterized by the transformation itself. Indeed, the monoids considered in Section 5.4 rely on the structure of the transducer, and thus they do not characterize the transformation. As a consequence, though we have expressiveness results, we have no decision procedure to determine whether or not a regular string function is definable in FOT. In [RS91], a congruence is defined directly *via* the transformation, and is used to characterize rational functions. This tool has recently been used in [Lho15] to derive a procedure determining, given a rational function, whether it is FOT-definable. Similar algebraic tools do exist also for subsequential functions, but it is still a big challenge to extend them to regular string functions.

Regarding first-order definable transformations, another research direction concerns star-free expressions. Indeed, the well-known equivalence between first-order logic and star-free expressions on words has been lifted to a weighted setting in [DG08, MR13] under some hypotheses on the semi-ring under consideration. The extension of these results to transductions constitutes an interesting objective, and the description of regular string functions using a kind of regular expressions that has been recently proposed in [ADR15] could serve as a good starting point.

Chapter 6

Tree-to-string transductions

Contents

While the previous section was concerned with transformations of strings, we want to consider trees as input. With the widespread of the XML data model, trees, and more precisely unranked trees (or hedges), are widely used to organize information. Motivated by the study of the streamability of transformations, we will be interested in the model of nested words, which correspond to linearizations of trees, as they are used in XML documents.

6.1 Visibly pushdown automata

Nested words Nested words are finite sequences of symbols equipped with a nesting structure. The term *nesting structure* refers to a structure that is organized on the basis of layers some of which are contained into the others. Special symbols, namely *call* and *return* symbols, can be used to add a nesting structure, through a call/return matching relation. Other (untyped) symbols are called *internal* symbols. Formally, a structured alphabet is a triple $\Sigma = (\Sigma_c, \Sigma_r, \Sigma_\iota)$ composed of the sets of call, return and internal symbols. A nested word is an element of Σ^*. The set of well-nested words over Σ, denoted Σ^*_{wn}, is the smallest subset of Σ^* such that $\Sigma^*_\iota \subseteq \Sigma^*_{\mathsf{wn}}$, and for all $u, v \in \Sigma^*_{\mathsf{wn}}$, for all $c \in \Sigma_c$, $r \in \Sigma_r$, we have $uv \in \Sigma^*_{\mathsf{wn}}$ and $cur \in \Sigma^*_{\mathsf{wn}}$. In a nested word, some call symbols (resp. return symbols) may be pending if they do not match any return symbol (resp. call symbol). *Well-nested words* are nested words without any pending symbols. We say that a nested word is open-call (resp. open-return) if it only contains call pending symbols (resp. return pending symbols), which holds precisely if it is the prefix (resp. the suffix) of some well-nested word. Given an open-call nested word u, we define the *current height* of u as the number of pending calls. We denote it by $\mathsf{hc}(u)$. Given a well-nested word u, the *height* of u is the maximal number of pending calls on any prefix of u, and is denoted $\mathsf{h}(u)$.

Nested words arise naturally in many applications as models of data which are both linearly and hierarchically ordered. As mentioned above, XML documents are examples of data naturally modeled as nested words, but this is also the case of executions of structured programs [AM09]. The sequence of calls and returns (resp. opening and closing tags), adds nesting structures to program executions and XML data respectively. More generally, any tree-structured data can be linearized into a nested word by considering a depth-first left-to-right traversal of the tree.

Visibly pushdown automata Rajeev Alur and Parthasarathy Madhusudan introduced *visibly pushdown automata* (VPA) as a pushdown machine to define languages of nested words [AM09], called visibly pushdown languages. More precisely, VPA operate on a structured alphabet, and when reading a *call* symbol the automaton must push one symbol onto its stack, when reading a *return* symbol it must pop the symbol on top of its stack, and when reading an *internal* symbol it cannot touch its stack. Nested words can be naturally viewed as trees (and conversely), and back and forth translations between visibly pushdown automata and tree automata have been proposed. As a consequence, VPA inherit several good properties of tree automata [CDG$^+$07]. Most notably, they are closed under all Boolean operations, and all the classical decision problems such as emptiness, universality and finiteness are decidable. This is in contrast to general pushdown automata that define context-free languages, for which it is well-known that most interesting problems are undecidable. The fundamental reason why VPA enjoy good properties is that the stack behavior, *i.e.* whether it pops, pushes or does not change, is driven by the input symbol. Therefore, all the stacks on a same input have the same height (we say that the stacks are *synchronized*). Although VPA are equivalent to tree automata, they run on words and operate from left to right, *i.e.* on linearization of the input tree (or hedge), which is often the case in practice. This perspective has led to many applications, such as new logics for nested words [AAB$^+$08], streaming XML validation and queries [Alu07, GNT09, MV09, KMV07], and program analysis [HHP10, HJR12, AC10b].

Formally, a VPA is a tuple $A = (Q, I, F, \Gamma, \delta)$ where Q is a finite set of states, $I \subseteq Q$, is the set of initial states, $F \subseteq Q$, is the set of final states, Γ is the (finite) stack alphabet, and $\bot \notin \Gamma$ is the bottom of the stack symbol, and $\delta = \delta_c \uplus \delta_r \uplus \delta_\iota$ is the transition relation partitioned into subrelations on call, return and internal symbols respectively.

Due to the constraints associated with the structure of the alphabet, the type of the elements of δ depends indeed on the input symbol. We detail the example of call symbols. Given $c \in \Sigma_c$, an element of δ_c on c is of the form $(p, c, \gamma, q) \in Q \times \Sigma_c \times \Gamma \times Q$, with the intended meaning that the stack symbol γ is pushed onto the stack when this transition is used in state p, reaching state q. Let us also recall that elements of δ_r may include return transitions on the empty stack.

A configuration of A is a pair (q, σ) where $q \in Q$ is a state and $\sigma \in \bot \cdot \Gamma^*$ a stack. Let $w = a_1 \dots a_\ell$ be a word on Σ, and $(q, \sigma), (q', \sigma')$ be two configurations of A. A *run* of the VPA A over w from (q, σ) to (q', σ') is a sequence of transitions $\rho = t_1 t_2 \dots t_\ell \in \delta^*$ such that there exist $q_0, q_1, \dots q_\ell \in Q$ and $\sigma_0, \dots \sigma_l \in \bot \cdot \Gamma^*$ with $(q_0, \sigma_0) = (q, \sigma)$, $(q_\ell, \sigma_\ell) = (q', \sigma')$, and for each $0 < k \leq \ell$, we have either:

- $t_k = (q_{k-1}, a_k, \gamma, q_k) \in \delta_c$ and $\sigma_k = \sigma_{k-1}\gamma$,
- $t_k = (q_{k-1}, a_k, \gamma, q_k) \in \delta_r$, and $\sigma_{k-1} = \sigma_k\gamma$ or $\sigma_{k-1} = \sigma_k = \gamma = \bot$, or
- $t_k = (q_{k-1}, a_k, q_k) \in \delta_i$, and $\sigma_{k-1} = \sigma_k$.

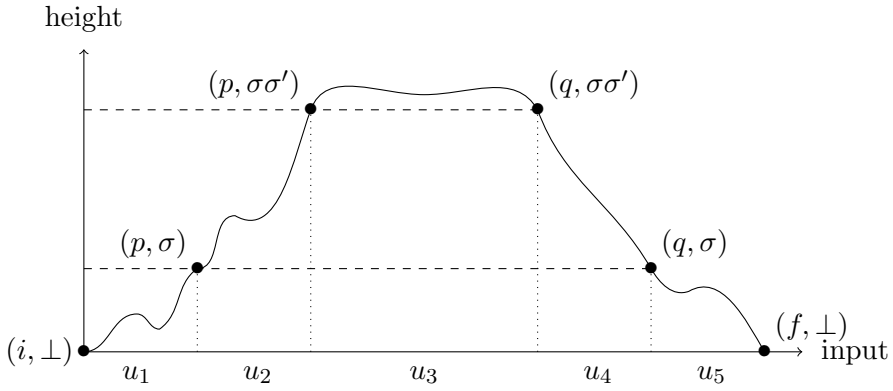


Figure 6.1: A run of a VPA exhibiting matched loops.

Such a run is *accepting* if $q \in I$, $\sigma = \perp$, and $q' \in F$. The definitions of accepted word and accepted language follow. The latter one is denoted $L(A)$.

A language L over Σ is a *visibly pushdown language* if there is a VPA A over Σ such that $L(A) = L$.

A VPA $A = (Q, I, F, \Gamma, \delta)$ is said to be *deterministic* if $|I| = 1$ and given any configuration (q, σ) and any symbol $\sigma \in \Sigma$, there exists at most one $t \in \delta$ labelled by σ and which can be executed from (q, σ). The set of deterministic VPA is denoted dVPA.

Loops of VPA We present some elementary facts on loops in VPA.

Loops in finite-state machines are simply runs on some input word u around some state q. Such loops have their counter part in VPA as runs on some input word u around some configuration (q, σ). Note that this does not imply that u is a well-nested word, but we can in general consider a well-nested conjugate u' of u which induces a loop. Such loops induce a "horizontal" concatenation in terms of hedges: consider a well-nested accepting run on the input word $u_1u_2u_3$ such that there is a loop on the well-nested word u_2. Then the words $u_1u_2^nu_3$ are accepted for all $n \geq 0$. In terms of hedges, u_1 and u_3 represent a context in which the hedge represented by u_2 is inserted. We can thus copy this hedge as many times as we want.

Viewing VPA as tree automata, one obtains a second type of loops, which corresponds to the repetition of some context. This situation is present in VPA as a pair of so called matched loops: consider an accepting run $(i, \perp) \xrightarrow{u_1} (p, \sigma) \xrightarrow{u_2} (p, \sigma\sigma') \xrightarrow{u_3} (q, \sigma\sigma') \xrightarrow{u_4} (q, \sigma) \xrightarrow{u_5} (f, \perp)$ where the words u_1u_5, u_2u_4 and u_3 are well-nested. We thus have two loops, one on u_2, which is increasing the content of the stack, and a second one on u_4 which pops from the stack exactly what the first loop pushed. This is illustrated in Figure 6.1. We verify easily that the words $u_1u_2^nu_3u_4^nu_5$ are accepted for all $n \geq 0$. In terms of hedges, the pair (u_2, u_4) represents the context which is iterated.

We will use these observations when we will define twinning properties for VPT in Section 6.4.

Trimming VPA Trimming a finite state automaton amounts to removing useless states, *i.e.* states that do not occur in some accepting computation of the automaton: every state of the automaton should be both reachable from an initial state, and co-reachable from a

final state. This property is important from both a practical and a theoretical point of view. Indeed, most of the algorithmic operations performed on an automaton will only be relevant on the trimmed part of this automaton. Removing useless states may thus avoid the study of irrelevant paths in the automaton, and speed up the analysis. From a theoretical aspect, there are several results holding for automata provided they are trimmed. For instance, consider the twinning property introduced by Choffrut in [Cho77], and presented in Chapter 4. Similarly, the boundedness of finite-state automata with multiplicities can be characterized by means of simple patterns for trimmed automata (see [MS77, SdS08]).

As we will see in our further developments, this property will also be important for VPA. However, the notion itself of trimmed automaton is not clear when considering pushdown automata. Indeed, the set of final configurations is too large (recall that any stack content is allowed), and thus it would be too strong to require that every final configuration is reachable from an initial configuration.

Together with Jean-Marc Talbot and our PhD student Mathieu Caralp, we studied the problem of trimming VPA. Our results have been published in [8, 9] and we only give here a short description of our results. First, we provide a simple (polynomial time) construction which allows to trim (in the natural sense) so called well-nested VPA, in which we require that acceptance is only with empty stack, and in which return transitions on the empty stack are disallowed. Then, we propose to address the issue of final configurations by restring our attention to final configurations that are reachable from some initial configuration. This yields a notion of trimmed (arbitrary) VPA. We then provide a polynomial time construction that, given a VPA, returns an equivalent trimmed VPA. Last, we have also proven that one can combine our trimming procedure with the standard determinization procedure, which implies the non-trivial result stating that every visibly pushdown language can be recognized by a visibly pushdown automaton which is both trimmed and deterministic.

6.2 Visibly pushdown transducers

We introduce now *visibly pushdown transducers* (VPT) as the extension of VPA with outputs. While the input alphabet is some structured alphabet Σ, the output alphabet Δ, however, may not be structured. VPT are therefore *nested word to word* transducers. If the output alphabet is structured, then obviously VPT output nested words. The particular case of transducers producing as output well-nested words is considered in the next section.

The goal of this set of results that have been published in [16] is to show that, like VPA, VPT enjoy good properties and form a robust class of transducers. To this end, we will successively consider basic properties of this new class of transducers, namely expressiveness, closure properties, and decidability of the main problems such as functionality and equivalence.

Visibly pushdown transducers A *visibly pushdown transducer* (VPT) from Σ to Δ is a pair $T = (A, \Omega)$ where $A = (Q, I, F, \Gamma, \delta) \in \text{VPA}$ is the underlying automaton and Ω is a morphism from δ to Δ^* called the output.

A *run* ρ of T over a word $u = a_1 \dots a_l$ is a run of its underlying automaton A, *i.e.* it is a sequence of transition $\rho = t_1 \dots t_l \in \delta^*$. The output of ρ is the word $v = \Omega(\rho) = \Omega(t_1 \dots t_l) = \Omega(t_1) \dots \Omega(t_l)$. We write $(q, \sigma) \xrightarrow{u/v} (q', \sigma')$ when there exists a run on u from (q, σ) to (q', σ') producing v as output.

The transducer T defines a binary word relation $[\![T]\!] = \{(u, v) \mid \exists q \in I, q' \in F, \sigma \in \Gamma^*,\ (q, \bot) \xrightarrow{u/v} (q', \sigma)\}$. We say that a transduction is a VPT transduction whenever there exists a VPT that defines it.

We use the (naturally defined) notions of domain and range of T and the classes of functional, resp. deterministic, VPT, denoted by fVPT, resp. dVPT.

Remark 1. *As* VPA *do not allow ε-transitions, neither do* VPT *on the input. However some transitions might output the empty word ε. In that sense the class of* VPT *we define is the class of real-time* VPT.

Visibly pushdown transducers have been first introduced in [RS08] and independently in [TVY08]. The definition considered in these papers do allow for ε-transitions that can produce outputs and only a single letter can be produced by each transition. However, such a definition causes many interesting problems to be undecidable, such as functionality and equivalence (even of functional transducers). Therefore we prefer to stick to our definition of VPT*, they are exactly the so called nested word to word transducers of [SLLN09] and correspond to the definition of [16].*

Relations between the classes of transducers (and thus, of transductions) we have considered so far can be depicted as :

$$\text{dVPT} \subsetneq \text{fVPT} \subsetneq \text{VPT}$$

The last inclusion is trivially strict. For the first one, we give an example below proving that the inclusion is strict.

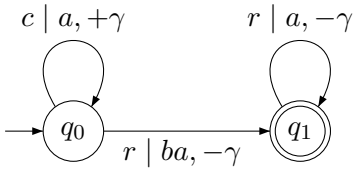


Figure 6.2: A deterministic VPT T_1 on $\Sigma_c = \{c\}$ and $\Sigma_r = \{r\}$.

Example 12. *This first example is a deterministic* VPT $T_1 = (A, \Omega)$ *represented in Figure 6.2. The input alphabet is composed of the call symbol c, and the return symbol r. The output alphabet is $\Delta = \{a, b\}$. Clearly T_1 implements the following transduction:*

$$c^n r^m \to a^n b a^m \quad \textit{for all} \quad 1 \leq m \leq n$$

The range of T_1 is a context-free language but it is not a visibly pushdown language for any partition of the output alphabet:

$$\text{range}(T_1) = \{a^n b a^m \mid 1 \leq m \leq n\}$$

The next example contains a non-deterministic VPT. This VPT is, however, functional, furthermore one can easily show that there is no equivalent deterministic VPT. In other words, it is an example of non-determinizable functional VPT.

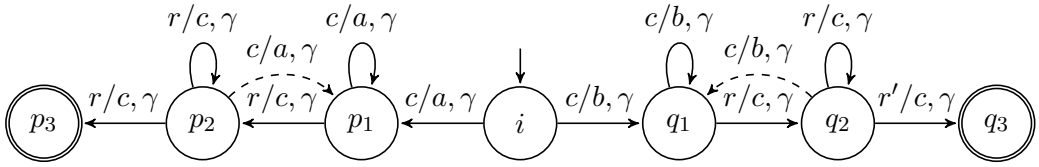


Figure 6.3: A functional VPT T_2 with $\Sigma_c = \{c\}$, $\Sigma_r = \{r, r'\}$ and $\Sigma_\iota = \{a, b\}$

Example 13. *Consider the* VPT T_2 *of Figure 6.3 represented in plain arrows. It starts in state i, first reads letter c and guesses non-deterministically the last letter to be either r or r', deciding to move left or right. The left and right parts accept indeed the same input words except for the last letter of the word. The domain of T_2 is* $\mathrm{dom}(T_2) = \{c^n r^n \mid n \geq 2\} \cup \{cc^n r^n r' \mid n \geq 1\}$. *Any word $c^n r^n$ is translated into $a^n c^n$, and any word $cc^n r^n r'$ is translated into $b^{n+1} c^{n+1}$. Therefore the translation of the first sequence of calls depends on the last letter r or r'.*

Expressiveness The domain of a VPT is the language accepted by its underlying automaton, therefore the domain is a visibly pushdown language. The range of a VPT is the image of the language of its runs by the output morphism. The runs of a VPA form a CFL (even a VPL), and as the image of a CFL by a morphism is a CFL, so is the range of a VPT. Finally, it is easy to show that for any CFL L there exists a VPT such that its range is L.

The class of VPT is a strict subclass of pushdown transducers. Indeed, any VPT is clearly a non-deterministic pushdown tranducer. But some non-deterministic pushdown transductions are not VPT transduction, for instance those non-deterministic pushdown transduction whose domain is a context-free language but not a VPL. Obviously, VPT forms a strict superclass of finite state transducers.

Closure properties The closure status under boolean operations is the same as for NFT. Thanks to non-determinism, transductions defined by VPT are closed under *union*. However, the classes of VPT and dVPT are not closed under intersection, complement, nor inverse. Last, the class of dVPT is not closed under union. The proofs of these results are similar to those for NFT.

The difference with NFT comes from the composition. Indeed, VPT are not closed under composition, nor dVPT. However, this makes sense as we did not consider a structured output alphabet. In the next section, we prove that the closure under composition can be obtained when structured output alphabets are considered.

Decision problems We study relevant decision problems for transductions for the class of VPT: emptiness, membership, type checking, functionality and k-valuedness, inclusion and equivalence.

The emptiness and translation membership problems are decidable in PTime for pushdown transducers. Therefore, since VPT are pushdown transducers, these results trivially hold for VPT.

We consider the *type checking problem* for a VPT T against VPL. By reducing the inclusion of a CFL in a VPL to this problem, we easily show that this type checking problem is undecidable. One way to overcome this problem is to restrict the class of (output) languages used for testing to regular languages. In this case, the type checking becomes decidable in

ExpTime. Another way that we will detail later to get decidability of type checking against VPA is by restricting the class of VPT.

Functionality Recall that a VPT T is *functional* if $R(T)$ is a function. We will prove:

Theorem 37 ([16])**.** *Functionality of* VPT *is decidable in* PTime.

Let us start with some example.

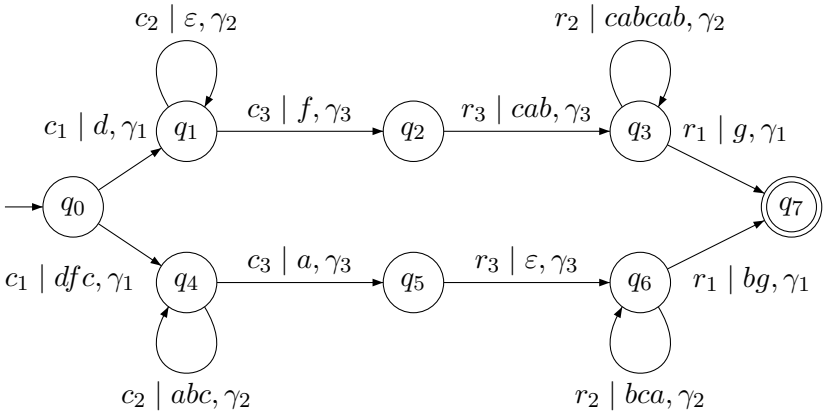


Figure 6.4: A functional VPT T_3 on $\Sigma_c = \{c_1, c_2, c_3\}$ and $\Sigma_r = \{r_1, r_2, r_3\}$.

Example 14. *Consider the* VPT T_3 *of Figure 6.4. Call (resp. return) symbols are denoted by c (resp. r). The domain of* T_3 *is* $\mathrm{dom}(T_3) = \{c_1(c_2)^n c_3 r_3 (r_2)^n r_1 \mid n \in \mathbb{N}\}$*. For each word of* $\mathrm{dom}(T_3)$*, there are two accepting runs, corresponding respectively to the upper and lower part of* T_3 *(therefore it is not unambiguous). For instance, when reading* c_1*, it pushes* γ_1 *and produces either d (upper part) or dfc (lower part). By following the upper part (resp. lower part), it produces words of the form* $dfcab(cabcab)^n g$ *(resp.* $dfc(abc)^n a(bca)^n bg$*).*

It is not difficult to verify that the VPT T_3 of Example 14 is functional. Recall that the characterization of functional NFT provided in [BCPS03] relies on the notion of *delay* between runs on the same input word. It states that these delays are bounded. The challenge for deciding functionality for VPT lies in the fact that for some functional VPT outputs are produced in some desynchronised manner and the difference in their lengths might get arbitrarily long. As shown by this example, the delay between two outputs for a VPT can grow arbitrarily, even if it is functional. Indeed, after reading $c_1 c_2^n$, the upper part outputs just d, while the lower part outputs $dfc(abc)^n$: the delay between both outputs is $fc(abc)^n$ and grows linearly with the height of the stack. For T to be functional, the upper run must catch up its delay before reaching an accepting state. In this case, it will catch up with the outputs produced on the return transitions.

The decision procedure for functionality of VPT is based on the squaring construction [BCPS03] and on the decidability of the morphism equivalence problem [Pla94]. The *square* T^2 of a VPT T is realized through a product construction of the underlying automaton with itself. Intuitively, it is a transducer that simulates any two parallel transductions of T, where by *parallel transductions* we mean two transductions over the same input word. Each transition of the square simulates two transitions of T that have the same input letter.

The output of a transition t of T^2 that simulates the transitions t_1 and t_2 of T is the pair of output words formed by the outputs of t_1 and t_2. If Ω is the output morphism of T, we write O_T for $\Omega(\delta)$, that is, the set of words that are output of transitions of T, then the output alphabet of T^2 is the set of pairs of words in O_T, *i.e.* $O_T \times O_T$.

Note that the square T^2 is a VPT and therefore its range is a context-free language over the alphabet $O_T \times O_T$.

The procedure to decide functionality of a VPT T is based on the following observation. A run $\rho = (t_1, t'_1) \dots (t_k, t'_k)$ of T^2 simulates two runs, say $\rho_1 = t_1 \dots t_k$ and $\rho_2 = t'_1 \dots t'_k$, of T over a same input word. The output of ρ is the sequence of pairs of words $\Omega'(\rho) = \Omega'((t_1, t'_1) \dots (t_k, t'_k)) = (\Omega(t_1), \Omega(t'_1)) \dots (\Omega(t_k), \Omega(t'_k))$. Therefore the projection on the first component, resp. second, of the output of ρ gives the output of the corresponding run of T, that is $\Omega(\rho_1)$ and $\Omega(\rho_2)$ respectively. These projections on the first and second components can be defined by two morphisms Π_1 and Π_2 as follows: $\Pi_1((u_1, u_2)) = u_1$ and $\Pi_2((u_1, u_2)) = u_2$. Clearly, T is functional if and only if these two morphisms are equal on any output of T^2, *i.e.* if for all $w = (u_1, u'_1) \dots (u_k, u'_k) \in \text{range}(T^2)$ we have $\Pi_1(w) = \Pi_2(w)$ that is $u_1 \dots u_k = u'_1 \dots u'_k$. In other words the two morphisms must be equivalent on the range of T^2.

Given two morphisms and a language, checking whether the two morphisms are equivalent on this language is known as the *morphism equivalence problem*. Plandowski has shown that this problem is decidable in PTime for context-free languages given by a grammar in Chomsky normal form. This concludes the proof of Theorem 37.

Equivalence and inclusion The equivalence and inclusion problems are already undecidable for finite-state transducers [Gri68]. Therefore, they are also undecidable for VPT.

Let us consider now functional VPT: given two functional VPT, T_1 and T_2, they are equivalent, resp. T_1 is included into T_2, if and only if their union is functional and they have the same domains, resp. the domain T_1 is included into the domain of T_2. The domains being VPL, testing their equivalence or the language inclusion is ExpTime-c [AM09]. Testing the equivalence or the inclusion of the domains is easier when the VPT are deterministic, it can be done in PTime [AM09]. Therefore both procedures, *i.e.* equivalence or inclusion testing, can be done in PTime when the VPT are deterministic. If we assume that both transducers are total, then obviously we do not have to test the equivalence or inclusion of their domains. Therefore, in that case equivalence and inclusion of their transductions can also be tested in PTime.

Theorem 38 ([16])**.** *The inclusion and the equivalence problems are :*

- *undecidable for* VPT
- ExpTime-c *for functional* VPT
- *in* PTime *for* deterministic VPT *[SLLN09], resp. for* total *functional* VPT

k-valuedness Given some (fixed) integer $k \in \mathbb{N}$, the *k-valuedness problem* asks, given as input a VPT T, whether it is k-valued.

The k-valued transductions are an interesting generalization of functional transductions. In the case of finite state transducers, k-valued and functional transducers share the important characteristic to have decidable equivalence and inclusion problems. In this section we sketch

how to decide k-valuedness for VPT in co-NPTime. We, however, leave open the question of deciding the equivalence or the inclusion of k-valued VPT.

The idea is to generalize the construction for deciding functionality presented in the previous section. This previous construction is based on the square and the morphism equivalence problem. We use here the k-power and the multiple morphisms equivalence problem [HIKS02], which generalizes the morphism equivalence problem to k morphisms, instead of 2.

The main tool of the decision procedure is the class of bounded reversal counter automata introduced by Ibarra [Iba78]. We proceed in three steps:

1. The procedure of [Iba78] for deciding emptiness of such machines can be executed in co-NPTime. This relies on a recent result allowing the construction in linear time of an existential Presburger formula representing the Parikh image of a context-free language [VSS05]. [1]

2. As a direct consequence, the procedure for deciding the multiple morphism equivalence problem in [HIKS02] can be executed in co-NPTime.

3. This last result allows to decide the k-valuedness problem for VPT in co-NPTime using a product construction similar to that described for functionality.

Theorem 39 ([16])**.** *Let $k \geq 0$ be fixed. The problem of deciding whether a* VPT *is k-valued is in co-*NPTime*.*

We conjecture that the equivalence of two k-valued VPT is decidable, but leave it as an open problem. Using techniques based on reversal bounded counter machines, we have proven the following decidability result in a journal version of [16] which is under submission:

Theorem 40 (Unpublished)**.** *The following problem is decidable: given k_1 functional* VPT *$T_1, \ldots, T_{k_1}$, and k_2 functional* VPT *$G_1, \ldots, G_{k_2}$, does $\bigcup_{i=1}^{k_1} R(T_i) = \bigcup_{i=1}^{k_2} R(G_i)$ hold ?*

For k-valued NFT, the equivalence problem is known to be decidable, and the procedure relies on the existence of a decomposition of any k-valued NFT as a finite union of functional NFT [Web93, SdS10]. This decomposition is based on a notion of delay between output of runs on the same input. Generalising this notion of delay to VPT is a challenging open problem, and would likely lead to an effective decomposition result for k-valued VPT, as finite unions of functional VPT. As a consequence of the above decidability result, such a decomposition result for VPT would give decidability of equivalence for k-valued VPT.

6.3 Tree-to-tree transformations

Tree-to-tree transformations arise in many natural applications such as database management systems, or transformations of XML documents, in which we need to model transformations of (the linearization of) some input tree into (the linearization of) some output tree. Moreover, as we will see, considering a structured output alphabet and restricting the VPT so as to only produce well-nested output words yields a more robust class of VPT.

[1] This result has been improved in [HL11] where it is shown that the co-NPTime upper bound still holds even if the number of counters is not fixed, for a slightly more general model. Moreover, [HL11] shows that this co-NP upper bound is also a lower bound.

The main results of the previous section showed that functionality and, more generally, k-valuedness are decidable for VPT. As a consequence, the equivalence and inclusion problems for functional VPT are decidable. All these problems are undecidable for pushdown transducers.

On the other hand, contrary to the class of NFT, the class of VPT is not closed under composition and the type checking problem against VPA is undecidable. A closer look at the reason for this non-closure and undecidability results points to an interesting subclass of VPT. Both of these weaknesses of the model can be solved by adding a 'visibly' constraint on the output of the VPT.

Indeed, the non-closure under composition can be viewed as a consequence of the fact that the stack of the two involved VPT are not synchronized. The stack of the first VPT is guided by the input word, while the stack of the second is guided by the output of the first VPT. Constraining the VPT with some synchronization between its input and its output yields closure under composition. This leads to the definition of *locally well-nested* VPT that we will study in the first part of this section.

Secondly, we will prove that replacing this syntactical condition by a semantical one, which only requires that every output word is well-nested is sufficient to obtain a class of VPT closed under composition and for which type-checking against VPA is decidable.

Locally well-nested VPT A *locally well-nested* VPT (lwnVPT for short) is a VPT with a notion of synchronization between the input and the output. This synchronization is enforced with the following *syntactic* restriction: for all call and return transitions that use the same stack symbol, the concatenation of the output word of the call transition with the output word of the return transition must be a well-nested word. Moreover the output word of any internal transition must be a well-nested word. We can thus interpret this condition as a mapping $s : \Gamma \to \mathbb{N}$ which gives, for each stack symbol γ, the number $s(\gamma)$ of unmatched calls (respectively returns) of the output words produced on some call transition pushing γ onto the stack (resp. some return transition popping γ from the stack).

Let us consider a rule $q(f(x_1, x_2)) \to C[q_1(x_1), q_2(x_2)]$ of some ranked tree transducer. In such a rule, C is a context, and thus the linearization of the right-hand side of the rule is a well-nested word. This is the intuition underlying the definition of lwnVPT.

As a model of tree transducers, lwnVPT will only be considered on well-nested input words. It is then easy to prove that every output word is well-nested. Actually, when considering a run $(p, \bot) \xrightarrow{u|v} (q, \sigma)$ in some lwnVPT, we know precisely the "form" of the output word v. Indeed, while u is an open-call word with $|\sigma|$ unmatched call symbols, the word v is an open-call word with $\sum_{\gamma \in \sigma} s(\gamma)$ unmatched call symbols. This formalizes the intuition that the input and the output are synchronized.

Given two transducers T_1 and T_2, we sketch the construction of a transducer T_3 accepting their composition. Intuitively, T_3 needs to simulate the execution of T_2 on the output of T_1. States of T_3 are simply pairs of states of T_1 and T_2. Consider some transition $(p_1, \bot) \xrightarrow{c|v} (q_1, \gamma_1)$ in T_1. As v contains $s(\gamma_1)$ unmatched calls, the simulation of T_2 on v will be of the form $(p_2, \bot) \xrightarrow{v|w} (q_2, \sigma_2)$, where $|\sigma_2| = s(\gamma_1)$. We thus consider a kind of product stack alphabet composed of stack symbols of the form (γ_1, σ_2), where γ_1 is a stack symbol of T_1 and σ_2 is a stack word of T_2 of length $s(\gamma_1)$. This allows us to prove:

Theorem 41 ([16]). *Let T_1, T_2 be two* VPT. *If T_1 is locally well-nested, then we can construct in* PTime *a* VPT T_3 *such that $[\![T_3]\!] = [\![T_2]\!] \circ [\![T_1]\!]$. Moreover, if T_2 is local well-nested, then so is T_3. As a consequence, the class* lwnVPT *is closed under composition.*

VPL are closed under complement, and it is easy to define the identity function on some VPL as a lwnVPT. Using this and the closure under composition of lwnVPT, we easily prove:

Theorem 42 ([16]). *The type checking problem for* lwnVPT *against* VPA *is* ExpTime-c. *It is in* PTime *if the* VPA *constraining the output language is deterministic.*

Globally well-nested VPT The class of locally well-nested VPT is defined by a syntactic condition which can very easily be checked on a VPT. However, the lwnVPT-definability problem, which consists in determining, given a VPT T, whether there exists an equivalent lwnVPT T', is open, and seems to be non trivial. Another drawback of the class lwnVPT is that there exist simple VPT-definable tree-to-tree transformations which are not lwnVPT definable.

To overcome these issues, we introduce the new class of *globally well-nested* VPT (gwnVPT for short) which is defined semantically. Given a VPT T from the structured alphabet Σ to the structured alphabet Δ, we simply require that every output of some well-nested word over Σ is a well-nested word over Δ. We will prove the following results showing that we obtain an interesting class of VPT for modeling tree-to-tree transformations:

1. gwnVPT are strictly more expressive than lwnVPT
2. the gwnVPT-definability problem is decidable in PTime
3. the class gwnVPT is closed under composition, and its type-checking problem against VPA is decidable in 2-ExpTime

We start with an example, showing point 1. Consider the VPT T_4 depicted on Figure 6.5. It is easy to verify that it belongs to gwnVPT. However, we can prove that there exists no equivalent lwnVPT T'. Indeed, assuming the existence of such a T', it should own two matched loops corresponding to the two loops around states p_1 and p_2 in T_4. By a case analysis on the form of the outputs of these loops, one obtains a contradiction.

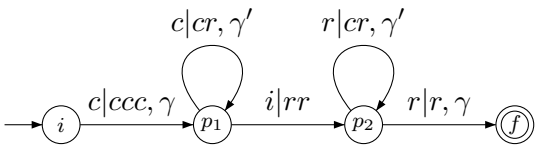


Figure 6.5: The VPT T_4 with input/output alphabet $\Sigma = (\{c\}, \{r\}, \{i\})$.

Let us consider now the gwnVPT-definability problem. Let T be a VPT from Σ^* to Δ^*. By definition, T belongs to the class gwnVPT iff $T(\Sigma^*_{\mathsf{wn}}) \subseteq \Delta^*_{\mathsf{wn}}$. As we have shown before, the range of a VPT may be any CFL, and the language Δ^*_{wn} is a VPL. As it is well-known that the inclusion of a CFL into a VPL is undecidable, we don't get a solution to our problem directly. However, the recent work [BCR11] provides tractable cases of the above undecidable problem. In particular, the inclusion of a CFL into a Dyck language is in PTime. Using this result, we prove:

Theorem 43 ([26]). *Let A be a* VPT*. Whether $A \in$* gwnVPT *can be decided in* PTime.

We turn now to the closure under composition, from which the decidability of the type-checking problem against VPL easily follows. The construction of composition for lwnVPT heavily relies on the synchronisation between the stacks of the two composed transducers, which itself directly follows from the definition of the class lwnVPT. This is not anymore the case for gwnVPT, and we need new tools to prove that such a synchronization does exist.

To this end, we study a superclass, defined as those VPT such that the supremum of the number of unmatched symbols over all output words is finite. As this corresponds to a boundedness property, we manage to characterize this property by means of patterns on loops of the transducers. Using these patterns, we prove the following key lemma:

Lemma 19. *Let $T \in$* gwnVPT *and consider a run $(p, \bot) \xrightarrow{u|v} (q, \bot)$ in T, such that p and q are respectively reachable and co-reachable with the same stack σ. Then the number of unmatched calls in v (as well as its number of unmatched returns) is bounded exponentially in the number of states of T.*

We use this result to prove:

Theorem 44 ([26]). *Let T_1, T_2 be two* VPT*. If A is globally well-nested, then one can compute in exponential time in the size of T_1 and T_2 a* VPT *T_3 such that $[\![T_3]\!] = [\![T_2]\!] \circ [\![T_1]\!]$. Moreover, if T_2 is also globally well-nested, then so is T_3. As a consequence, the class* gwnVPT *is closed under composition.*

Intuitively, Lemma 19 gives us a bound on the buffer of stack symbols of T_2 that we need to store. However, unlike the construction used for lwnVPT, part of this buffer needs to be stored in the states of T_3.

Corollary 6 ([26]). *The type checking problem for* gwnVPT *against* VPA *is in 2-*ExpTime*. It is in* ExpTime *if the* VPA *constraining the output language is deterministic.*

Comparison with other models of tree transducers We investigate the relative expressive power of VPT and different classes of tree transducers. We characterize the expressive power of VPT w.r.t. their ability to express hedge-to-string (H2S), and hedge-to-hedge (H2H) transformations.

Hedges Let Λ be an alphabet. We let $S(\Lambda)$ be the signature $\{0, \cdot\} \cup \{a \mid a \in \Lambda\}$ where 0 is a constant symbol, $a \in \Lambda$ are unary symbols and $\cdot$ is a binary symbol. The set of *hedges* $\mathcal{H}_\Lambda$ over Λ is the quotient of the free $S(\Lambda)$-algebra by the associativity of $\cdot$ and the axioms $0 \cdot h = h \cdot 0 = h$. The constant 0 is called the empty hedge. We may write a instead of $a(0)$, and omit $\cdot$ when it is clear from the context. *Unranked trees* are particular hedges of the form $a(h)$ where $h \in \mathcal{H}_\Lambda$. Note that any hedge h is either empty or can be decomposed as $h = a(h_1) \cdot h_2$. Hedges over Λ can be naturally encoded as well-nested words over the structured alphabet Λ_s that contains the call symbol c_f (resp. the return symbol r_f) for each $f \in \Lambda$. We denote by $\mathsf{lin}(h)$ the linearization of an hedge h. Conversely, any well-nested word over a structured alphabet Σ can be encoded as an hedge over the product alphabet $\Sigma_c \times \Sigma_r$.

Hedge-to-string transducers We present now a model of hedge-to-string transducers (H2S) that run directly on hedges, and is closer to classical transducers than VPT are. In particular, this model is a syntactic subclass of macro forest transducers (MFT) [PS04] with no parameters, no swapping and no copy. Let Λ and Δ be two finite alphabets. An *hedge-to-string*

transducer from Λ to Δ (the class is denoted $\mathsf{H2S}(\Lambda, \Delta)$) is a tuple $T = (Q, I, \delta)$ where Q is a set of states, $I \subseteq Q$ is a set of initial states and δ is a set of rules of the form: [2]

$$q(0) \to \varepsilon \qquad\qquad q(f(x_1) \cdot x_2) \to w_1 q_1(x_1) w_2 q_2(x_2) w_3$$

where $q, q_1, q_2 \in Q$, $f \in \Lambda$ and $w, w_1, w_2, w_3 \in \Delta^*$.

The semantics of T is defined via mappings $[\![q]\!] : \mathcal{H}_\Lambda \to 2^{\Delta^*}$ for all $q \in Q$ as follows:

$$[\![q]\!](0) \quad = \begin{cases} \{\varepsilon\} & \text{if } q(0) \to \varepsilon \in \delta \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![q]\!](f(h) \cdot h') \;=\; \bigcup_{\substack{q(f(x_1)\cdot x_2)\to \\ w_1 q_1(x_1) w_2 q_2(x_2) w_3}} w_1 \cdot [\![q_1]\!](h) \cdot w_2 \cdot [\![q_2]\!](h') \cdot w_3$$

The transduction of an $\mathsf{H2S}$ $T = (Q, I, \delta)$ is defined as the relation $\{(h, s) \mid \exists q \in I,\ s \in [\![q]\!](h)\}$. We say that T is *tail-recursive* whenever in any rule, we have $w_3 = \varepsilon$. We denote by $\mathsf{H2S_{tr}}$ the class of tail-recursive $\mathsf{H2S}$.

Example 15. *Let Λ be a finite alphabet. Consider $T_1 \in \mathsf{H2S}(\Lambda, \Lambda)$ defined by $Q = I = \{q, q'\}$ and the following rules, for all $f \in \Lambda$:*

$$q(0) \to \varepsilon \qquad q'(0) \to \varepsilon \qquad q(f(x_1) \cdot x_2) \to q'(x_1) q(x_2) f$$

The domain of T_1 is the set of strings over Λ (viewed as a particular case of hedges) and T_1 defines the mirror image of strings.

Example 16. *Let Λ be a finite alphabet and Λ_s be its structured version. We define $T_2 \in \mathsf{H2S}(\Lambda, \Lambda_s \cup \{c_\#, r_\#\})$ which can non-deterministically root any subhedge of the input hedge under a new symbol $\#$ and output the linearization of the new hedge. For instance, the input tree $f(abcd)$ can be non-exhaustively translated into the string $\mathsf{lin}(f(a\#(bc)d))$ or the string $\mathsf{lin}(f(\#(ab)\#(cd)))$. Formally, T_2 is defined by $Q = \{q_0, q_1, q_2\}$, $I = \{q_0\}$ and δ defined as the following set of rules: (observe that $T_2 \in \mathsf{H2S_{tr}}$) $q_i(0) \to \varepsilon \quad \forall i \in \{0, 2\}$, and*

$$\begin{array}{ll} q_0(f(x_1) \cdot x_2) \to c_f q_0(x_1) r_f q_0(x_2) & q_1(f(x_1) \cdot x_2) \to c_f q_2(x_1) r_f r_\# q_0(x_2) \\ q_0(f(x_1) \cdot x_2) \to c_\# c_f q_2(x_1) r_f r_\# q_0(x_2) & q_1(f(x_1) \cdot x_2) \to c_f q_2(x_1) r_f q_1(x_2) \\ q_0(f(x_1) \cdot x_2) \to c_\# c_f q_2(x_1) r_f q_1(x_2) & q_2(f(x_1) \cdot x_2) \to c_f q_2(x_1) r_f q_2(x_2) \end{array}$$

Hedge-to-hedge transducers We consider now transducers running on hedges but producing (representations of) hedges as well-nested words. We define them as restrictions of the two models we have considered so far. Of course, lwnVPT and gwnVPT can be interpreted as hedge-to-hedge transducers.

We assume the output alphabet Δ to be structured as (Δ_c, Δ_r). We define an $\mathsf{H2S}(\Lambda, \Delta)$ to be an hedge-to-hedge transducer ($\mathsf{H2H}(\Lambda, \Delta)$) if any rhs $w_1 q_1(x_1) w_2 q_2(x_2) w_3$ of its transition rules satisfies $w_1 w_2 w_3 \in \Delta^*_{\mathsf{wn}}$. We denote $\mathsf{H2H_{tr}}$ the class of $\mathsf{H2H}$ that are additionally tail-recursive.

Let us consider two structured alphabets Σ and Δ, and the $\mathsf{H2S}$ transducer T_1 of Example 15 over the input alphabet $\Sigma_c \times \Sigma_r$. We naturally have $\mathsf{lin}(\mathrm{dom}(T_1)) = (\Sigma_c.\Sigma_r)^*$. Over such input words, any VPT is equivalent to some NFT, and it is well-known that the mirror transformation is not NFT-definable. We obtain: ($\mathrm{VPT}(\Sigma, \Delta)$ denotes the class of VPT from Σ^* to Δ^*)

[2] We consider linear and order-preserving rules only.

Lemma 20. *There exists* $T \in \mathsf{H2S}(\Sigma_c \times \Sigma_r, \Delta)$ *such that for all* $T' \in \mathrm{VPT}(\Sigma, \Delta)$, $[\![T]\!] \neq [\![T']\!]$.

Intuitively, this is due to the ability that H2S have to "complete" the output once the current hedge is processed. This ability vanishes when tail-recursive H2S are considered, and we prove:

Theorem 45 ([6])**.** *The class* $\mathsf{H2S_{tr}}(\Sigma_c \times \Sigma_r, \Delta)$ *is as expressive as the class* $\mathrm{VPT}(\Sigma, \Delta)$.

(Sketch). Intuitively, in order to transform $A \in \mathrm{VPT}(\Sigma, \Delta)$ into $T \in \mathsf{H2S_{tr}}(\Sigma_c \times \Sigma_r, \Delta)$, we proceed as follows. States of T are pairs of states of A, corresponding to states reached respectively at the beginning and at the end of the processing of an hedge. More formally, the following rule exists in T iff there exist a call transition $p \xrightarrow{c|w_1,\gamma} p_1$, a matching return transition $p_2 \xrightarrow{r|w_2,\gamma} q_1$, and the hedge represented by x_1 (resp. by x_2) can be processed from state p_1 to state p_2 (resp. from q_1 to q):

$$(p, q)((c, r)(x_1) \cdot x_2) \to w_1 \cdot (p_1, p_2)(x_1) \cdot w_2 \cdot (q_1, q)(x_2)$$

It is worth observing that this encoding directly implies the tail-recursive property of T.

The converse construction follows the same ideas. The stack is used to store the transition used on the call symbol, to recover it when reading the return symbol. □

Lemma 20 still holds even if we restrict H2S to H2H, because the transducer defining the transduction of Example 15 is actually an H2H. Similarly, Theorem 45 also holds when restricted to hedge-to-hedge transductions (the same constructions apply):

Theorem 46 ([6])**.** *The class* $\mathsf{H2H_{tr}}(\Sigma_c \times \Sigma_r, \Delta)$ *is as expressive as the class* $\mathrm{lwnVPT}(\Sigma, \Delta)$.

6.4 Streamability

We present now our results concerning the streamability of nested-word to word transformations, represented as visibly pushdown transducers. More precisely, we investigate *static analysis* of memory usage for this kind of programs on nested words, which are well-suited to represent XML transformations.

As for finite-state transducers, some transductions defined by (functional and non-deterministic) VPT cannot be evaluated efficiently in streaming (consider for instance the transformation swapping the first and last letter of a nested word). Our aim is thus to identify decidable classes of transductions for various memory requirements that are suitable to space-efficient streaming evaluation. We first consider the bounded memory requirement already introduced for string-to-string transformations. However when dealing with nested words in a streaming setting, the bounded memory requirement is quite restrictive. Indeed, even performing such a basic task as checking that a word is well-nested or checking that a nested word belongs to a regular language of nested words requires a memory dependent on the height (the level of nesting) of the input word [SS07]. This observation leads us to the second question: decide, given a transducer, whether the transduction can be evaluated with a memory that depends only on the size of the transducer and the height of the word (but not on its length). In that case, we say that the transduction is *height bounded memory* (HBM). This is particularly relevant to XML transformations as XML documents can be very long but have usually a small depth [BMV05]. HBM does not specify *how* memory depends on the height. This motivates

the introduction of a third class of nested word transductions which we prove decidable, and whose evaluation can be done with a memory that depends *polynomially* on the height of the input word.

Bounded memory transformations

In Chapter 5, we have defined bounded memory string transformations, and proved that this property is decidable for both rational functions (Corollary 1) and regular functions (Corollary 2) using their equivalence with subsequential functions.

We can also prove the following negative result using a reduction of the problem of deciding whether a CFL is regular:

Theorem 47 ([13]). *It is undecidable whether a pushdown transduction is BM.*

For VPT, BM is quite restrictive as it imposes to verify whether a word is well-nested by using a bounded amount of memory. This can be done only if the height of the words of the domain is bounded by some constant which depends on the transducer only: [3]

Theorem 48 ([13]). *Let T be a functional* VPT *with n states.*

1. *$[\![T]\!]$ is BM iff (i) for all $u \in \mathrm{dom}(T)$, $\mathsf{h}(u) \leq n^2$, and (ii) $[\![\mathrm{NFT}(T, n^2)]\!]$ is BM;*
2. *It is decidable in* Co-NPTime *whether $[\![T]\!]$ is BM.*

To prove property 2., the decision procedure first checks condition i) in PTime. In the sequel, we define the class of height bounded memory transductions, and show it is decidable in Co-NPTime (Theorem 49). On words of bounded height, this class collapses with bounded memory transductions, hence the result.

An algorithm for efficient evaluation of VPT

We present an online algorithm LcpIn to evaluate functional word transductions defined by fVPT. We first introduce some notations. We fix some fVPT T and define, given an input word u, the set $\mathsf{reach}(u) = \{(q, \sigma, v) \mid \exists q_0 \in I,\ (q_0, \bot) \xrightarrow{u|v} (q, \sigma)\}$. The algorithm LcpIn only applies on trimmed fVPT. The functionality of T then ensures that, for a given input word u and a configuration (q, σ), there is at most one word v such that $(q, \sigma, v) \in \mathsf{reach}(u)$.

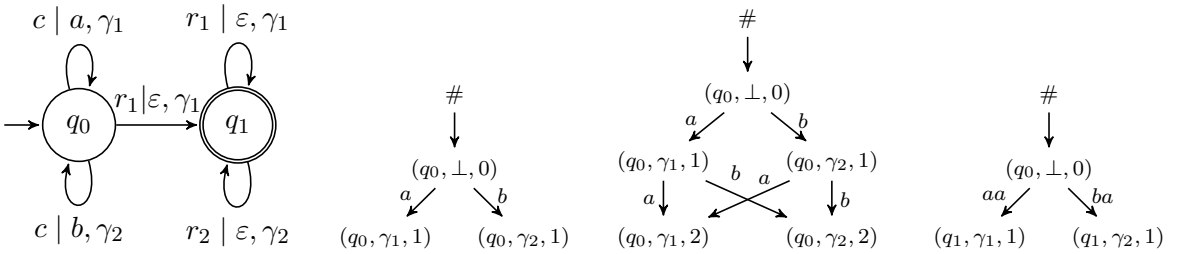
The output word produced by LcpIn after reading some input word u is the longest common prefix of the set $\pi_3(\mathsf{reach}(u))$, where π_3 denotes the projection on the third component. We denote this word by $\mathsf{lcp}_{\mathsf{out}}(u)$.

To achieve this, the core task of algorithm LcpIn is to maintain the set $\mathsf{reach}_{\mathsf{lcp}}(u)$ defined as $\mathsf{reach}_{\mathsf{lcp}}(u) = \{(q, \sigma, w) \mid (q, \sigma, \mathsf{lcp}_{\mathsf{out}}(u)w) \in \mathsf{reach}(u)\}$.

The set of reachable configurations may be of exponential size in the height of the input word (consider for instance the VPT of Figure 6.6.(a)). To avoid this blow-up, we use a DAG structure which aims at compacting this set. We illustrate it using an example depicted in Figure 6.6.(b)-(d).

The depth of the DAG is always equal to the current height of the input word read so far, and the DAG structure only stores the output words that have not been produced yet.

[3] In this result, the notation $\mathrm{NFT}(T, k)$ denotes the NFT obtained from the VPT T by restricting it to input words u such that $\mathsf{h}(u) \leq k$.



(a) A VPT. (b) After reading c. (c) After reading cc. (d) After reading ccr_1.

Figure 6.6: Data structure used by LcpIn.

Assume we have read the input word u and we want to handle a new symbol a. We can prove that we can use the DAG structure representing $\mathsf{reach}_{\mathsf{lcp}}(u)$ to perform the following operations: *i*) update the set $\mathsf{reach}_{\mathsf{lcp}}(u)$ with the transitions labeled by a, *ii*) compute the output word v as the lcp of the resulting set, and *iii*) remove this word v so as to obtain a DAG representing the set $\mathsf{reach}_{\mathsf{lcp}}(ua)$.

In order to establish the complexity of LcpIn, we introduce some notations. Let $\mathsf{out}_{\neq}(u)$ be the maximal size of outputs of T on u where their common prefix is removed: $\mathsf{out}_{\neq}(u) = \max\{|w| \mid (q, \sigma, w) \in \mathsf{reach}_{\mathsf{lcp}}(u)\}$. We prove the following complexity result:

Proposition 13 ([13])**.** *Given an* fVPT T*, one can build in* PTime *a Turing transducer, denoted* $M_{\text{LcpIn}}(T)$*, such that* $[\![M_{\text{LcpIn}}(T)]\!] = [\![T]\!]$*, and which, after reading a prefix* u' *of a well-nested word* $u \in \Sigma^*$*, uses space in* $O((\mathsf{hc}(u') + 1) \cdot \mathsf{out}_{\neq}(u'))$ *on the working tape.*

Height bounded memory transformations

We formalize the fact the a transformation can be evaluated with a memory whose size only depends on the height of the input word using the notion of Turing transducer:

Definition 11. *A (functional) transduction* $R \subseteq \Sigma^* \times \Sigma^*$ *is* height bounded memory (HBM) *if there exists a Turing transducer* M *and a function* $f : \mathbb{N} \to \mathbb{N}$ *such that* $[\![M]\!] = R$ *and on any input word* $u \in \Sigma^*$*,* M *runs in space at most* $f(\mathsf{h}(u))$*.*

Example 17. *Consider the* VPT T_2 *of Figure 6.3 page 98 represented in plain arrows. As we have seen it is functional but not sequentializable, and thus the transformation* $[\![T_2]\!]$ *cannot be evaluated with a bounded amount of memory. However, one can verify that it is possible with a memory that depends on the height of the input word.*

Recall that we have seen that BM functional NFT-transductions are characterized by the so called *twinning property*, which is decidable in PTime. We introduce a similar characterization of HBM fVPT-transductions, called the *horizontal twinning property* (HTP). Intuitively, the HTP requires that two runs on the same input cannot accumulate increasing output delay on loops.

Definition 12. *Let* T *be an* fVPT*.* T *satisfies the* horizontal twinning property *(HTP) if for all* $u_1, u_2, v_1, v_2, w_1, w_2 \in \Sigma^*$ *such that* u_2 *is well-nested, for all* $q_0, q_0' \in I$*, for all* $q, q' \in Q$*, and for all* $\sigma, \sigma' \in \Gamma^*$ *such that* (q, σ) *and* (q', σ') *are co-accessible,*

$$\textit{if} \begin{cases} (q_0, \perp) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \\ (q_0', \perp) \xrightarrow{u_1/w_1} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma') \end{cases} (1) \quad \textit{then } \mathsf{delay}(v_1, w_1) = \mathsf{delay}(v_1 v_2, w_1 w_2).$$

Proposition 14 ([13])**.** *The HTP is decidable in* Co-NPTime *for* fVPT.

Sketch. Let T be a fVPT. We construct in polynomial time a pushdown automaton with s counters[4] (one reversal) that accepts any word $u = u_1u_2u_3$ such that u_1u_2 satisfies the premise of the HTP but such that $\mathsf{delay}(v_1, w_1) \neq \mathsf{delay}(v_1v_2, w_1w_2)$ (*i.e.* the HTP is not satisfied). Therefore the HTP holds if and only if no word is accepted by the automaton. This can be checked in Co-NPTime [16]. □

We now show that HTP characterizes HBM fVPT-transductions.

Theorem 49 ([13])**.** *Let T be an* fVPT*. $[\![T]\!]$ is HBM iff the HTP holds for T, which is decidable in* Co-NPTime*. In this case, the Turing transducer $M_{\text{LcpIn}}(T)$ runs, on an input stream u, in space complexity exponential in the height of u.*

We can state more precisely the space complexity of $M_{\text{LcpIn}}(T)$ when T is reduced. In this case, it is in $O(3(\mathsf{h}(u)+1)^3 \cdot |Q|^{2(\mathsf{h}(u)+1)} \cdot M)$, where $M = \max\{|v| \mid (q, a, v, \gamma, q') \in \delta\}$.

Example 18. *Consider the* VPT T_2' *obtained from the* VPT T_2 *of Figure 6.3 page 98 by including dashed arrows. We can prove that $[\![T_2']\!]$ is not HBM. Indeed, T_2' does not satisfy the HTP: the delays increase when looping on cr around states p_2 and q_2.*

Online bounded memory transformations

We have shown that a VPT-transduction is in HBM iff the horizontal twinning property holds. The notion of height-bounded memory is quite permissive as for instance, any transformation of *ranked* tree linearizations (given a fixed rank) is HBM. In this section, we introduce a stronger constraint on memory: the amount of memory must depend only, at each moment (*i.e.* at any position in the stream), on the current height of the nested word. We call this requirement *online bounded memory* (OBM).

We give an effective characterization of OBM VPT-definable transformations using a new twinning property (*matched twinning property*, or *MTP* for short). Since it is a characterization of transformations, this property does not depend on the VPT that implement them: two equivalent VPT that implement the same transformation both satisfy, or both do not satisfy, this twinning property. Another appealing property of OBM, compared to HBM, is that the maximal amount memory needed when running the algorithm LcpIn is at most *quadratic* in the current height of the input nested word, while it is *exponential* for HBM transformations, and this latter bound is tight.

Definition 13. *A (functional) transduction $R \subseteq \Sigma^* \times \Sigma^*$ is* online bounded memory *(*OBM*) if there exists a Turing transducer M and a function $f : \mathbb{N} \to \mathbb{N}$ such that $[\![M]\!] = R$ and on any input word $u \in \Sigma^*$, if M has read the prefix v of u (but not more), then the amount of memory of M on the working tape is less than $f(\mathsf{hc}(v))$.*

The matched twinning property is a strengthening of the horizontal twinning property obtained by adding some new delay contraints on the matched loops. Intuitively, the MTP requires that two runs on the same input cannot accumulate increasing output delay on matched loops. They can accumulate delay on loops with increasing stack but this delay has to be caught up on the matching loops with descending stack. We show that this property is decidable, and that subsequential VPT satisfy it. Therefore the class of OBM VPT-transformations *subsumes* the class of subsequentializable VPT.

[4]The number s does not depend on the transducer T.

Definition 14. *Let $T = (Q, I, F, \Gamma, \delta)$ be an* fVPT. *T satisfies the* matched twinning property *(MTP) if for all $u_i, v_i, w_i \in \Sigma^*$ ($i \in \{1, \ldots, 4\}$) such that u_3 is well-nested, and u_2u_4 is well-nested, for all $i, i' \in I$, for all $p, q, p', q' \in Q$, and for all $\sigma_1, \sigma_2 \in \bot.\Gamma^*$, for all $\sigma'_1, \sigma'_2 \in \Gamma^*$, such that (q, σ_1) and (q', σ_2) are co-accessible:*

$$\textit{if} \quad \begin{cases} (i, \bot) \xrightarrow{u_1/v_1} (p, \sigma_1) \xrightarrow{u_2/v_2} (p, \sigma_1\sigma'_1) \xrightarrow{u_3/v_3} (q, \sigma_1\sigma'_1) \xrightarrow{u_4/v_4} (q, \sigma_1) \\ (i', \bot) \xrightarrow{u_1/w_1} (p', \sigma_2) \xrightarrow{u_2/w_2} (p', \sigma_2\sigma'_2) \xrightarrow{u_3/w_3} (q', \sigma_2\sigma'_2) \xrightarrow{u_4/w_4} (q', \sigma_2) \end{cases}$$

then $\mathsf{delay}(v_1v_3, w_1w_3) = \mathsf{delay}(v_1v_2v_3v_4, w_1w_2w_3w_4)$. *We say that a* VPT *$T$ is* twinned *whenever it satisfies the MTP.*

Note that any twinned VPT also satisfies the HTP (with $u_3 = u_4 = \varepsilon$). Note also that, as a consequence of Theorem 50 below, any subsequentializable VPT satisfies the MTP, but the converse is false, as witnessed by the following example.

Example 19 (MTP does not imply subsequentializable)**.** *The* VPT *of Figure 6.3 with plain arrows does not satisfy the MTP, as the delay between the two branches increases when iterating the loops. Consider now the* VPT *obtained by replacing r by r' in the transition (q_1, r, c, γ, q_2). It is obviously twinned, as we cannot construct two runs on the same input which have the form given in the premises of the MTP. However this transducer is not subsequentializable, as the output on the call symbols cannot be delayed to the matching return symbols.*

As for the HTP, we can decide the MTP using a reduction to the emptiness of a pushdown automaton with bounded reversal counters:

Proposition 15 ([13])**.** *The matched twinning property is decidable in* Co-NPTime *for* fVPT.

The next result states the equivalence between the matched twinning property and the class of online bounded memory transformations. This is a strong result, as it shows that the matched twinning property is a machine independent characterization. Proving that the MTP implies the membership in the class OBM follows from an analysis of the algorithm LcpIn. Conversely, while our proof in [13] was a tedious case analysis using word combinatorics, we recently managed to give a short proof using a recent result of Saarela [Saa15], pointed to us by Sylvain Salvati.

Theorem 50 ([13])**.** *Let T be an* fVPT. *T satisfies the MTP iff $[\![T]\!]$ is OBM. In particular, it is decidable in* Co-NPTime *whether $[\![T]\!]$ is OBM. In this case, the Turing transducer $M_{\text{LcpIn}}(T)$ runs, on an input stream u, in space complexity quadratic in the height of u.*

To conclude this section, we give a summary of the relations between the different classes of VPT we have considered. While the classes BM, OBM and HBM are decidable in Co-NPTime, the decidability status of subsequentializable VPT is open.

$$BM \subsetneq \text{subsequentialVPT} \subsetneq OBM \subsetneq HBM \subsetneq \text{fVPT}$$

6.5 Perspectives

Visibly pushdown automata have been successful to transfer decidability results and closure properties from finite state automata. Similarly, we believe that the model of visibly pushdown transducers we have introduced enjoys most of the decidability properties of finite state transducers. We have already proven this for functionality and k-valuedness. When the output alphabet is structured, we have also proven the closure under composition, and the decidability of type checking.

However, there are still some important problems that are decidable for finite state transducers but open for VPT. In particular, the three following problems are worth being studied: first, the *sequentiality problem* asks, given a functional VPT, whether there exits an equivalent deterministic VPT; second, the *decomposition problem* aims at decomposing a finite valued VPT as a finite union of functional VPT, this would yield the decidability of equivalence for finite valued VPT; third, the *finite valuedness problem* consists in determining whether there exists an integer k such that a given VPT is k-valued.

Concerning the sequentiality, we have already addressed this problem unsuccessfully. We have for instance introduced a new twinning property (the matched twinning property defined in Section 6.4) adapted to the nested structure of the input words of a VPT, but we have shown that it characterizes a strict superclass of sequentializable VPT. However, it is possible to prove that this twinning property, when considered over a commutative semi-ring, does characterize subsequential functions (this is an unpublished result).

The problem of finite valuedness was one of the objectives of the PhD thesis of Mathieu Caralp, and we obtained partial results. For instance, our work on the boundedness of weighted visibly pushdown automata solves the problem for one-letter output alphabets. There are essentially two approaches to solve the finite valuedness problem for finite state transducers, and both rely on the use of patterns. The first one studied by Andreas Weber and Helmut Seidl in [Web90, Sei94], uses a decomposition of the automaton in strongly connected components in order to prove that the valuedness is finite when the patterns are satisfied. The second one, proposed by Jacques Sakarovitch and Rodrigo De Souza in [SdS08], uses the notion of delay (also considered in this thesis) in order to reduce the finite valuedness of a transducer to a boundedness problem for weighted automata. For a subclass of visibly pushdown transducers, we have identified a set of patterns and proven that they constitute a necessary condition. In order to prove that they are sufficient, we are currently investigating an adequate notion of delay. Such a notion would allow us to apply the approach of De Souza and Sakarovitch, hence reducing the finite valuedness problem to the boundedness problem for weighted visibly pushdown automata that we solved in [7]. It is worth observing that the delay is also at the core of the decomposition result as presented in [SdS10].

Last, models equipped with variables as SST constitute another perspective. A model adapted to nested words has been proposed in [AD12], in which the authors prove that the model is expressively equivalent to MSO-definable transformations of nested words. The problem of variable minimization for such a highly expressive model seems very challenging. A less ambitious objective is to study such models for commutative semi-rings. This would require to adapt to nested words the generalized twinning property we introduced in Section 5.3, a perspective already mentioned in the previous chapter.

Conclusion

This thesis summarizes some of my contributions in the area of formal verification for software systems. These results are organized in two parts: the first one addresses the issue of robustness in timed systems, and the second one the efficient evaluation of transformations. They consist mostly of decidability results, as well as characterizations of different subclasses.

Most of the decidability results on timed systems, for model checking or controller synthesis, rely on adequate refinements of the region automaton, or on suitable representations of clock constraints. It is worth mentioning that for each decision problem studied so far (model checking of several temporal logics, controller synthesis for different kind of plants), the theoretical complexities we obtain are the same for the robust and the standard version of the problem. This validates the interest of the enlarged semantics.

Regarding transducers, we managed to obtain a characterization of different classes of transformations using patterns and combinatorics of words. Such characterizations constitute a powerful tool that allows in particular to derive efficient algorithms for deciding classes of transformations. Using automata-logic connections, we can then decide fragments of logics (for instance order-preserving MSOT inside MSOT). This kind of problems is known to be difficult, and often addressed by means of algebraic tools, but such tools are currently missing in this context.

At the end of each chapter, I have already given detailed perspectives. I describe now some additional possible developments that I have not investigated yet. Of course, several other research directions could be considered, as automata based approaches can be used for various purposes, from verification to performance evaluation. For instance, let us mention that robustness issues have recently been considered in transducers [SDC13, HOS14], and even more recently in a model of timed transducers [HOS15].

Timed systems *Stochastic perturbations and timed automata* We have already considered different stochastic models of perturbations in our work on robust controller synthesis. It seems also relevant to consider such models for robust model checking. More generally, it would be interesting to generalize our works to timed automata including stochastic transitions. Such models have indeed already been studied as they are useful to describe real systems or protocols in which some behaviors are naturally modeled using a probability distribution.

Nash equilibria The games underlying our controller synthesis problems are two-player zero-sum games. In the more general setting of non-zero-sum games, the notion of equilibrium is relevant: a Nash equilibrium is a behavior of the players (*i.e.* a strategy for each player – there may be more than two players) such that no player can get a better payoff by modifying its strategy. It could be interesting to study how to introduce a notion of robustness in this setting: for instance, given some Nash equilibrium, could it be the case that even if the players

slightly diverge from their strategy, the resulting global behavior of the system stays close from the Nash equilibrium?

Transformations *From transducers to quantitative functions.* Several results presented in this thesis also hold for functions from words to a structure more general than that of words. In particular, cost register automata generalize the model of streaming string transducers, and our results presented in Section 5.3 do apply to this more general setting. Lifting some of our results to such a general setting is thus a promising research direction.

Streamability of regular hedge-to-string transformations For string-to-string functions, we managed to characterize the class of rational functions among regular ones. One of our objectives is to extend this result to hedge-to-string transformations. Regular hedge-to-string transformations correspond to the class of MSO-definable transformations from hedges to strings. A first step consists in identifying an equivalent model based on two-way VPT and to this end we conjecture that the class of two-way VPT extended with regular look-around and satisfying a finite-visit property coincides with the class of regular hedge-to-string transformations. The second step will then consist in generalizing the Rabin and Scott procedure described in Section 5.2 from words to nested words.

Specification languages for transformations An important asset of the model-checking approach is the specification logic LTL. This logic can also be used to specify input/ouput constraints of reactive systems. While temporal operators present in LTL are adequate to specify correctness properties of programs, it is not the case to describe programs manipulating strings. The recently introduced language Drex allows to describe any regular string-to-string function and a fragment of this language has been identified in order to admit an efficient evaluation (cubic in the size of the input word). However, the translation of Drex programs into transducers may be non-elementary. An interesting objective is then to introduce a specification formalism for transformations, and more generally for quantitative functions. This formalism should be sufficiently expressive to modelize standard programs manipulating strings, and should admit efficient translations into automata/transducers.

Bibliography

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, vol. 1855 of *Lecture Notes in Computer Science*, pp. 419–434. Springer-Verlag, July 2000.

[AAB+08] R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-order and temporal logics for nested words. *Logical Methods in Computer Science*, 4(4), 2008.

[AČ10a] R. Alur and P. Černý. Expressiveness of streaming string transducers. In *Proc. IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010*, vol. 8 of *LIPIcs*, pp. 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[AC10b] R. Alur and S. Chaudhuri. Temporal reasoning for procedural programs. In G. Barthe and M. V. Hermenegildo, editors, *VMCAI*, vol. 5944 of *Lecture Notes in Computer Science*, pp. 45–60. Springer, 2010.

[AČ11] A. Alur and P. Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proc. of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011*, pp. 599–610. ACM, 2011.

[ACS13] T. Abdellatif, J. Combaz, and J. Sifakis. Rigorous implementation of real-time systems - from theory to application. *Mathematical Structures in Computer Science*, 23(4):882–914, 2013.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theor. Comp. Sci.*, 126(2):183–235, 1994.

[AD11] R. Alur and J. V. Deshmukh. Nondeterministic streaming string transducers. In *Proc. 38th International Colloquium on Automata, Languages and Programming, ICALP 2011, Part II*, vol. 6756 of *Lecture Notes in Computer Science*, pp. 1–20. Springer, 2011.

[AD12] R. Alur and L. D'Antoni. Streaming tree transducers. In *Proc. 39th International Colloquium on Automata, Languages and Programming, ICALP 2012, Part II*, Lecture Notes in Computer Science, pp. 42–53. Springer, 2012.

[ADD+13] R. Alur, L. D'Antoni, J. V. Deshmukh, M. Raghothaman, and Y. Yuan. Regular functions and cost register automata. In *Proc. 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013*, pp. 13–22. IEEE Computer Society, 2013.

[ADR15] R. Alur, L. D'Antoni, and M. Raghothaman. Drex: A declarative language for efficiently evaluating regular string transformations. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015*, pp. 125–137. ACM, 2015.

[ADT13] R. Alur, A. Durand-Gasselin, and A. Trivedi. From monadic second-order definable string transformations to transducers. In *Proc. 28th Annual IEEE Symposium on Logic in Computer Science (LICS'13)*, pp. 458–467. IEEE Computer Society, 2013.

[AFH96] R. Alur, T. Feder, and Th. A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.

[AFT12] R. Alur, E. Filiot, and A. Trivedi. Regular transformations of infinite strings. In *Proc. 27th Annual IEEE Symposium on Logic in Computer Science (LICS'12)*, pp. 65–74. IEEE Computer Society, 2012.

[AHV93] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pp. 592–601, 1993.

[AKY10] P. A. Abdulla, P. Krcál, and W. Yi. Sampled semantics of timed automata. *Logical Methods in Computer Science*, 6(3), 2010.

[Alu07] R. Alur. Marrying words and trees. In *26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2007*, vol. 5140 of *LNCS*, pp. 233–242, 2007.

[AM09] R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3):1–43, 2009.

[AMPS98] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proceedings of the 5th IFAC Cconference on System Structure and Control (SSC'98)*, pp. 469–474. Elsevier Science, 1998.

[Ans90] M. Anselmo. Two-way automata with multiplicity. In *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings*, vol. 443 of *Lecture Notes in Computer Science*, pp. 88–102. Springer, 1990.

[AR13] R. Alur and M. Raghothaman. Decision problems for additive regular functions. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, vol. 7966 of *Lecture Notes in Computer Science*, pp. 37–48. Springer, 2013.

[AT05] K. Altisen and S. Tripakis. Implementation of timed automata: An issue of semantics or modeling? In P. Pettersson and W. Yi, editors, *Proceedings of the 3rd International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'05)*, vol. 3829 of *Lecture Notes in Computer Science*, pp. 273–288. Springer, 2005.

[BA11] N. Basset and E. Asarin. Thin and thick timed regular languages. In U. Fahrenberg and S. Tripakis, editors, *Formal Modeling and Analysis of Timed Systems*, vol. 6919 of *Lecture Notes in Computer Science*, pp. 113–128. Springer, 2011.

[BBB+07] C. Baier, N. Bertrand, P. Bouyer, Th. Brihaye, and M. Größer. Probabilistic and topological semantics for timed automata. In *Proc. 27th Conf. Found. Softw. Tech. & Theor. Comp. Sci. (FSTTCS'07)*, vol. 4855 of *Lecture Notes in Computer Science*, pp. 179–191. Springer, 2007.

[BCH+05a] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. Comparison of different semantics for time Petri nets. In *Proc. of ATVA*, vol. 3707 of *LNCS*, pp. 293–307. Springer, 2005.

[BCH+05b] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. Comparison of the expressiveness of timed automata and time petri nets. In *Proc. of FORMATS*, vol. 3829, pp. 211–225. Springer, 2005.

[BCPS03] M.-P. Béal, O. Carton, C. Prieur, and J. Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science*, 292(1):45–63, 2003.

[BCR11] A. Bertoni, C. Choffrut, and R. Radicioni. The inclusion problem of context-free languages: Some tractable cases. *International Journal of Foundations of Computer Science*, 22(2):289–299, 2011.

[BFH+01] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. 4th Int. Work. Hybrid Systems: Computation and Control (HSCC'01)*, vol. 2034 of *LNCS*, pp. 147–161. Springer, 2001.

[BGMP15] F. Baschenis, O. Gauwin, A. Muscholl, and G. Puppis. One-way definability of sweeping transducers. In *35th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. To appear.

[BMOW07] P. Bouyer, N. Markey, J. Ouaknine, and J. Worrell. The cost of punctuality. In *Proc. 22nd Ann. Symp. Logic in Computer Science (LICS'07)*. IEEE Comp. Soc. Press, 2007. 109–118.

[BMS11] P. Bouyer, N. Markey, and O. Sankur. Robust model-checking of timed automata via pumping in channel machines. In U. Fahrenberg and S. Tripakis, editors, *Proceedings of the 9th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'11)*, vol. 6919 of *Lecture Notes in Computer Science*, pp. 97–112, Aalborg, Denmark, September 2011. Springer.

[BMV05] D. Barbosa, L. Mignet, and P. Veltri. Studying the XML web: Gathering statistics from an xml sample. *World Wide Web*, 8:413–438, 2005.

[BO14] D. Bundala and J. Ouaknine. Advances in parametric real-time reasoning. In *Proc. 39th International Symposium on Mathematical Foundations of Computer Science (MFCS 2014)*, vol. 8634 of *Lecture Notes in Computer Science*, pp. 123–134. Springer, 2014.

[Boj14] M. Bojanczyk. Transducers with origin information. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *ICALP (2)*, vol. 8573 of *Lecture Notes in Computer Science*, pp. 26–37. Springer, 2014.

[CD15] O. Carton and L. Dartois. Aperiodic two-way transducers and fo-transductions. In *Proc. 24th EACSL Annual Conference on Computer Science Logic, CSL 2015*, vol. 41 of *LIPIcs*, pp. 160–174. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[CDF+05] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 2005*, vol. 3653, pp. 66–80. Springer-Verlag, 2005.

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2007.

[Cho77] C. Choffrut. Une Caractérisation des Fonctions Séquentielles et des Fonctions Sous-Séquentielles en tant que Relations Rationnelles. *Theor. Comput. Sci.*, 5(3):325–337, 1977.

[CHP11] K. Chatterjee, T. A. Henzinger, and V. S. Prabhu. Timed parity games: Complexity and robustness. *Logical Methods in Computer Science*, 7(4), 2011.

[CHR02] F. Cassez, T. A. Henzinger, and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In C. Tomlin and M. R. Greenstreet, editors, *Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control (HSCC'02)*, vol. 2289 of *Lecture Notes in Computer Science*, pp. 134–148. Springer, 2002.

[CJ77] M. P. Chytil and V. Jákl. Serial composition of 2-way finite-state transducers and simple programs on strings. In *Automata, languages and programming (Fourth Colloq., Univ. Turku, Turku, 1977)*, pp. 135–137. Lecture Notes in Comput. Sci., Vol. 52. Springer, Berlin, 1977.

[CJ99] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. 10th International Conference on Concurrency Theory (CONCUR'99*, vol. 1664 of *Lecture Notes in Computer Science*, pp. 242–257. Springer, 1999.

[CK87] K. Culik and J. Karhumaki. The equivalence problem for single-valued two-way transducers (on NPDT0L languages) is decidable. *SIAM Journal on Computing*, 16(2):221–230, 1987.

[CL14] V. Carnino and S. Lombardy. On determinism and unambiguity of weighted two-way automata. In *Proceedings 14th International Conference on Automata and Formal Languages, AFL 2014, Szeged, Hungary, May 27-29, 2014.*, vol. 151 of *EPTCS*, pp. 188–200, 2014.

[Cla99] J. Clark. XSL Transformations (XSLT) version 1.0, W3C recommendation, 1999.

[Cou94] B. Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75, 1994.

[dAFH+03] L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *In Proc. 14th International Conference CONCUR 2003 - Concurrency Theory*, vol. 2761 of *Lecture Notes in Computer Science*, pp. 142–156. Springer, 2003.

[DDMR04] M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin. Robustness and implementability of timed automata. In *Proc. Joint Conf. Formal Modelling and Analysis of Timed Systems & Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, vol. 3253 of *LNCS*, pp. 118–133. Springer, 2004.

[DDR05] M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. *Formal Aspects of Computing*, 17(3):319–341, 2005.

[DDSS07] D. D'Aprile, S. Donatelli, A. Sangnier, and J. Sproston. From time Petri nets to timed automata:An untimed approach. In *Proc. of TACAS*, vol. 4424 of *LNCS*, pp. 216–230. Springer, 2007.

[DG08] M. Droste and P. Gastin. On aperiodic and star-free formal power series in partially commuting variables. *Theory Comput. Syst.*, 42(4):608–631, 2008.

[DHS+14] A. Deshpande, F. Herbreteau, B. Srivathsan, T. Tran, and I. Walukiewicz. Fast detection of cycles in timed automata. *CoRR*, abs/1410.4509, 2014.

[DK06] C. Daws and P. Kordy. Symbolic robustness analysis of timed automata. In *Proc. 4th Intl Conf. Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, vol. 4202 of *LNCS*, pp. 143–155. Springer, 2006.

[dS13] R. de Souza. Uniformisation of two-way transducers. In *Language and Automata Theory and Applications - 7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. Proceedings*, vol. 7810, pp. 547–558. Springer, 2013.

[EH91] J. Engelfriet and L. Heyker. The string generating power of context-free hypergraph grammars. *J. Comput. Syst. Sci.*, 43(2):328–360, 1991.

[EH01] J. Engelfriet and H. J. Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Logic*, 2:216–254, 2001.

[EH07] J. Engelfriet and H. J. Hoogeboom. Finitary compositions of two-way finite-state transductions. *Fundam. Inform.*, 80(1-3):111–123, 2007.

[ELTV14] R. Ehlers, S. Lafortune, S. Tripakis, and M. Y. Vardi. Bridging the gap between supervisory control and reactive synthesis: Case of full observation and centralized control. In *12th International Workshop on Discrete Event Systems, WODES 2014, Cachan, France, May 14-16, 2014.*, pp. 222–227. International Federation of Automatic Control, 2014.

[EM65] C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Develop*, 9:47–68, 1965.

[Eng82] J. Engelfriet. Three hierarchies of transducers. *Mathematical Systems Theory*, 15(2):95–125, 1982.

[FGR15] E. Filiot, R. Gentilini, and J.-F. Raskin. Quantitative languages defined by functional automata. *Logical Methods in Computer Science*, 11(3:14):1–32, 2015.

[Fil15] E. Filiot. Logic-automata connections for transformations. In *Proc. 6th Indian Conference on Logic and Its Applications, ICLA 2015*, vol. 8923 of *Lecture Notes in Computer Science*, pp. 30–57. Springer, 2015.

[Fin93] A. Finkel. The minimal coverability graph for Petri nets. In *Proc. ICATPN'91*, vol. 674 of *LNCS*, pp. 210–243. Springer, 1993.

[FKT14] E. Filiot, S. N. Krishna, and A. Trivedi. First-order definable string transformations. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*, vol. 29 of *LIPIcs*, pp. 147–159. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[GHJ97] V. Gupta, Th. A. Henzinger, and R. Jagadeesan. Robust timed automata. In *Proc. International Workshop on Hybrid and Real-Time Systems (HART'97)*, vol. 1201 of *Lecture Notes in Computer Science*, pp. 331–345. Springer, 1997.

[GNT09] O. Gauwin, J. Niehren, and S. Tison. Earliest query answering for deterministic nested word automata. In *17th International Symposium on Fundamentals of Computation Theory, FCT 2009*, vol. 5699 of *LNCS*, pp. 121–132, 2009.

[Gri68] T. V. Griffiths. The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. *Journal of the ACM*, 15(3):409–413, 1968.

[GRVB10] G. Geeraerts, J.-F. Raskin, and L. Van Begin. On the efficient computation of the coverability set for petri nets. *International Journal of Foundations of Computer Science*, 21(2):135–165, 2010.

[Gur82] Gurari. The equivalence problem for deterministic two-way sequential transducers is decidable. *SICOMP: SIAM Journal on Computing*, 11, 1982.

[HHP10] M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In M. V. Hermenegildo and J. Palsberg, editors, *POPL*, pp. 471–482. ACM, 2010.

[HIKS02] T. Harju, O. H. Ibarra, J. Karhumaki, and A. Salomaa. Some decision problems concerning semilinearity and commutation. *Journal of Computer and System Sciences*, 65:278–294, 2002.

[HJR12] W. R. Harris, S. Jha, and T. W. Reps. Secure programming via visibly pushdown safety games. In *24th International Conference on Computer Aided Verification (CAV 2012)*, vol. 7358 of *Lecture Notes in Computer Science*, pp. 581–598. Springer, 2012.

[HL11] M. Hague and A. W. Lin. Model checking recursive programs with numeric data types. In *23rd International Conference on Computer Aided Verification, CAV 2011*, pp. 743–759, 2011.

[HOS14] T. A. Henzinger, J. Otop, and R. Samanta. Lipschitz robustness of finite-state transducers. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014*, vol. 29 of *LIPIcs*, pp. 431–443. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[HOS15] T. A. Henzinger, J. Otop, and R. Samanta. Lipschitz robustness of timed I/O systems. *CoRR*, abs/1506.01233, 2015.

[HRSV02] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 2002.

[Iba78] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978.

[IK86] K. C. II and J. Karhumäki. The equivalence of finite valued transducers (on HDT0L languages) is decidable. *Theor. Comput. Sci.*, 47(3):71–84, 1986.

[Imm87] N. Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987.

[Jau09] R. Jaubert. Aspects quantitatifs dans la réalisation de contrôleurs temps-réels robustes. Mémoire de Master Recherche, Master Informatique Fondamentale, Marseille, 2009.

[JF15] I. Jecker and E. Filiot. Multi-sequential word relations. In *19th International Conference on Developments in Language Theory, DLT 2015*, vol. 9168 of *Lecture Notes in Computer Science*, pp. 288–299. Springer, 2015.

[KLMP14] P. Kordy, R. Langerak, S. Mauw, and J. W. Polderman. A symbolic algorithm for the analysis of robust timed automata. In *19th International Symposium on Formal Methods (FM 2014)*, vol. 8442 of *Lecture Notes in Computer Science*, pp. 351–366. Springer, 2014.

[KM69] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.

[KMV07] V. Kumar, P. Madhusudan, and M. Viswanathan. Visibly pushdown automata for streaming XML. In *16th international conference on World Wide Web, WWW 2007*, pp. 1053–1062, 2007.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[KP05] P. Krcál and R. Pelánek. On sampled semantics of timed systems. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference*, vol. 3821 of *Lecture Notes in Computer Science*, pp. 310–321. Springer, 2005.

[KVW00] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000.

[Led13] J. Ledent. Towards an algebraic characterization of rational word functions. Mémoire de Stage L3 ENS Lyon, LaBRI, University of Bordeaux, 2013.

[Lho15] N. Lhote. Towards an algebraic characterization of rational word functions. Mémoire de Master Recherche, Master Recherche en Informatique, Bordeaux, 2015.

[LLTW14] K. G. Larsen, A. Legay, L. Traonouez, and A. Wasowski. Robust synthesis for real-time systems. *Theor. Comput. Sci.*, 515:96–122, 2014.

[Mer74] P. M. Merlin. *A Study of the Recoverability of Computing Systems.* PhD thesis, University of California, Irvine, CA, USA, 1974.

[MPS95] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In *STACS*, pp. 229–242, 1995.

[MR13] E. Mandrali and G. Rahonis. Characterizations of weighted first-order logics over semirings. In *Algebraic Informatics - 5th International Conference, CAI 2013, Porquerolles, France, September 3-6, 2013. Proceedings*, vol. 8080 of *Lecture Notes in Computer Science*, pp. 247–259. Springer, 2013.

[MS77] A. Mandel and I. Simon. On Finite Semigroups of Matrices. *Theor. Comput. Sci.*, 5(2):101–111, 1977.

[MV09] P. Madhusudan and M. Viswanathan. Query automata for nested words. In *34th International Symposium on Mathematical Foundations of Computer Science, MFCS 2009*, vol. 5734 of *LNCS*, pp. 561–573. Springer Berlin / Heidelberg, 2009.

[NMA+02] P. Niebert, M. Mahfoudh, E. Asarin, M. Bozga, O. Maler, and N. Jain. Verification of timed automata via satisfiability checking. In *Proc. 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 2002)*, vol. 2469 of *Lecture Notes in Computer Science*, pp. 225–244. Springer, 2002.

[OW03] J. Ouaknine and J. B. Worrell. Revisiting digitization, robustness and decidability for timed automata. In *Proc. 18th Ann. Symp. Logic in Computer Science (LICS'03)*. IEEE Comp. Soc. Press, 2003.

[OW05] J. Ouaknine and J. B. Worrell. On the decidability of metric temporal logic. In *Proc. 19th Ann. Symp. Logic in Computer Science (LICS'05)*, pp. 188–197. IEEE Comp. Soc. Press, 2005.

[OW08] J. Ouaknine and J. Worrell. Some recent results in metric temporal logic. In *Proc. 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2008)*, vol. 5215 of *Lecture Notes in Computer Science*, pp. 1–13. Springer, 2008.

[Pla94] W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Second Annual European Symposium on Algorithms, ESA 1994*, pp. 460–470, 1994.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pp. 179–190. ACM Press, 1989.

[PS04] T. Perst and H. Seidl. Macro forest transducers. *IPL*, 89(3):141–149, 2004.

[Pur98] A. Puri. Dynamical properties of timed automata. In *Proc. 5th Intl Symp. Formal techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, vol. 1486 of *LNCS*, pp. 210–227. Springer, 1998.

[Pur00] A. Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.

[Put94] M. L. Putterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, New York, NY, 1994.

[RS59] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[RS91] C. Reutenauer and M. P. Schützenberger. Minimization of rational word functions. *SIAM J. Comput.*, 20(4):669–685, 1991.

[RS08] J.-F. Raskin and F. Servais. Visibly pushdown transducers. In *35th International Colloquium on Automata, Languages and Programming, ICALP 2008*, vol. 5126 of *LNCS*, pp. 386–397, 2008.

[RW87] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, January 1987.

[Saa15] A. Saarela. Systems of word equations, polynomials and linear algebra: A new approach. *Eur. J. Comb.*, 47:1–14, 2015.

[San11] O. Sankur. Untimed language preservation in timed systems. In F. Murlak and P. Sankowski, editors, *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS'11)*, vol. 6907 of *Lecture Notes in Computer Science*, pp. 556–567. Springer, August 2011.

[San13] O. Sankur. *Robustness in timed automata: analysis, synthesis, implementation*. PhD dissertation, ENS Cachan, 2013.

[San15] O. Sankur. Symbolic quantitative robustness analysis of timed automata. In *Proc. 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015)*, vol. 9035 of *Lecture Notes in Computer Science*, pp. 484–498. Springer, 2015.

[SBM14] O. Sankur, P. Bouyer, and N. Markey. Shrinking timed automata. *Information & Computation*, 234:107–132, 2014.

[SDC13] R. Samanta, J. V. Deshmukh, and S. Chaudhuri. Robustness analysis of string transducers. In *Proc. 11th International Symposium on Automated Technology for Verification and Analysis*, vol. 8172 of *Lecture Notes in Computer Science*, pp. 427–441. Springer, 2013.

[SdS08] J. Sakarovitch and R. de Souza. On the decidability of bounded valuedness for transducers. In *33rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2008*, pp. 588–600, 2008.

[SdS10] J. Sakarovitch and R. de Souza. Lexicographic decomposition of k -valued transducers. *Theory of Computing Systems*, 47(3):758–785, 2010.

[Sei94] H. Seidl. Equivalence of finite-valued tree transducers is decidable. *Mathematical Systems Theory*, 27(4):285–346, 1994.

[She59] J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.

[SLLN09] S. Staworko, G. Laurence, A. Lemay, and J. Niehren. Equivalence of deterministic nested word to word transducers. In *17th International Symposium on Fundamentals of Computation Theory, FCT 2009*, vol. 5699 of *LNCS*, pp. 310–322, 2009.

[SS07] L. Segoufin and C. Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *ICDT*, pp. 299–313, 2007.

[Sta12] A. Stainer. Frequencies in forgetful timed automata. In M. Jurdziński and D. Nickovic, editors, *Proceedings of the 11th International Conference on Formal Modeling and Analysis of Timed Systems*, vol. 7595 of *Lecture Notes in Computer Science*, pp. 236–251. Springer, 2012.

[TVY08] A. Thomo, S. Venkatesh, and Y. Y. Ye. Visibly pushdown transducers for approximate validation of streaming XML. In *5th international conference on Foundations of information and knowledge systems, FoIKS 2008*, pp. 219–238, 2008.

[Var89] M. Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, 30(5):261–264, 1989.

[Var95] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, August 27 - September 3, 1995, Proceedings)*, vol. 1043 of *Lecture Notes in Computer Science*, pp. 238–266. Springer, 1995.

[VH14] A. Valmari and H. Hansen. Old and new algorithms for minimal coverability sets. *Fundam. Inform.*, 131(1):1–25, 2014.

[VSS05] K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational horn clauses. In *20th International Conference on Automated Deduction, CADE 2005*, vol. 3632 of *LNCS*, pp. 337–352, 2005.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pp. 332–344. IEEE Computer Society, 1986.

[Web90] A. Weber. On the valuedness of finite transducers. *Acta Inf.*, 27(8):749–780, 1990.

[Web93] A. Weber. Decomposing finite-valued transducers and deciding their equivalence. *SIAM Journal on Computing*, 22(1):175–202, 1993.

[WK95] A. Weber and R. Klemm. Economy of description for single-valued transducers. *Information and Computation*, 118(2):327–340, 1995.

Appendix A

Publications

[1] S. Akshay, L. Hélouet, C. Jard, and P.-A. Reynier. Robustness of time petri nets under guard enlargement. In *Proc. 6th International Workshop on Reachability Problems (RP'12)*, volume 7550 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2012.

[2] S. Akshay, L. Hélouet, C. Jard, and P.-A. Reynier. Robustness of time petri nets under guard enlargement. *Fundamenta Informaticae*, 2015. To appear.

[3] P. Bouyer, N. Markey, and P.-A. Reynier. Robust model-checking of linear-time properties in timed automata. In J. R. Correa, A. Hevia, and M. Kiwi, editors, *Proceedings of the 7th Latin American Symposium on Theoretical INformatics (LATIN'06)*, volume 3887 of *Lecture Notes in Computer Science*, pages 238–249. Springer, 2006.

[4] P. Bouyer, N. Markey, and P.-A. Reynier. Robust analysis of timed automata via channel machines. In *Proc. 11th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'08)*, volume 4962 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2008.

[5] P. Bulychev, F. Cassez, A. David, K. G. Larsen, J.-F. Raskin, and P.-A. Reynier. Controllers with minimal observation power (application to timed systems). In *Proc. 10th International Symposium on Automated Technology for Verification and Analysis (ATVA'12)*, volume 7561 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2012.

[6] M. Caralp, E. Filiot, P.-A. Reynier, F. Servais, and J.-M. Talbot. Expressiveness of visibly pushdown transducers. In *Proc. 2nd International Workshop on Trends in Tree Automata and Tree Transducers (TTATT'13)*, volume 134, pages 17–26. EPTCS, 2013.

[7] M. Caralp, P.-A. Reynier, and J.-M. Talbot. Visibly pushdown automata with multiplicities: Finiteness and k-boundedness. In *Proc. 16th International Conference on Developments in Language Theory (DLT'12)*, volume 7410 of *Lecture Notes in Computer Science*, pages 226–238. Springer, 2012.

[8] M. Caralp, P.-A. Reynier, and J.-M. Talbot. Trimming visibly pushdown automata. In *Proc. 18th International Conference on Implementation and Application of Automata*

(CIAA'13), volume 7982 of *Lecture Notes in Computer Science*, pages 84–96. Springer, 2013.

[9] M. Caralp, P.-A. Reynier, and J.-M. Talbot. Trimming visibly pushdown automata. *Theoretical Computer Science*, 578:13–29, 2015.

[10] F. Cassez, J. J. Jessen, K. G. Larsen, J.-F. Raskin, and P.-A. Reynier. Automatic synthesis of robust and optimal controllers - an industrial case study. In *Proc. 12th International Conference on Hybrid Systems: Computation and Control (HSCC'09)*, volume 5469 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2009.

[11] L. Dartois and P.-A. Reynier. Aperiodic transducers. *CoRR*, abs/1506.04059, 2015.

[12] L. Daviaud, P.-A. Reynier, and J.-M. Talbot. A Generalized Twinning Property for Minimisation of Cost Register Automata. Technical report, HAL-01201704, 2015.

[13] E. Filiot, O. Gauwin, P.-A. Reynier, and F. Servais. Streamability of nested word transductions. In *Proc. 31st Annual International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'11)*, volume 13 of *LIPIcs*, pages 312–324. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[14] E. Filiot, O. Gauwin, P.-A. Reynier, and F. Servais. From two-way to one-way finite state transducers. In *Proc. 28th Annual IEEE Symposium on Logic in Computer Science (LICS'13)*, pages 468–477. IEEE Computer Society, 2013.

[15] E. Filiot, S. Maneth, P.-A. Reynier, and J.-M. Talbot. Decision problems of tree transducers with origin. In *Proc. 42nd International Colloquium on Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015*, volume 9135 of *Lecture Notes in Computer Science*, pages 209–221. Springer, 2015.

[16] E. Filiot, J.-F. Raskin, P.-A. Reynier, F. Servais, and J.-M. Talbot. Properties of visibly pushdown transducers. In *Proc. 35th International Symposium on Mathematical Foundations of Computer Science (MFCS'10)*, volume 6281 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2010.

[17] E. Filiot and P.-A. Reynier. On streaming string transducers and HDT0L systems. *CoRR*, abs/1412.0537, 2014.

[18] R. Jaubert and P.-A. Reynier. Quantitative robustness analysis of flat timed automata. In *Proc. 14th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'11)*, volume 6604 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2011.

[19] J. Malinowski, P. Niebert, and P.-A. Reynier. A hierarchical approach for the synthesis of stabilizing controllers for hybrid systems. In *Proc. 9th International Symposium on Automated Technology for Verification and Analysis (ATVA'11)*, volume 6996 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2011.

[20] O.-L. Nguena-Timo and P.-A. Reynier. On characteristic formulae for event-recording automata. In *Proc. Workshop on Fixpoints In Computer Science (FICS'09)*, pages 70–78, 2009.

[21] O.-L. Nguena-Timo and P.-A. Reynier. On characteristic formulae for event-recording automata. *RAIRO - Theoretical Informatics and Applications*, 47(1):69–96, 2013.

[22] Y. Oualhadj, P.-A. Reynier, and O. Sankur. Probabilistic robust timed games. In *Proc. 25th International Conference on Concurrency Theory (CONCUR'14)*, volume 8704 of *Lecture Notes in Computer Science*, pages 203–217. Springer, 2014.

[23] P.-A. Reynier and A. Sangnier. Weak time petri nets strike back! In *Proc. 20th International Conference on Concurrency Theory (CONCUR'09)*, volume 5710 of *Lecture Notes in Computer Science*, pages 557–571. Springer, 2009.

[24] P.-A. Reynier and F. Servais. Minimal coverability set for petri nets: Karp and miller algorithm with pruning. In *Proc. 32nd International Conference on Application and Theory of Petri Nets and Concurrency (ICATPN'11)*, volume 6709 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.

[25] P.-A. Reynier and F. Servais. Minimal coverability set for petri nets: Karp and miller algorithm with pruning. *Fundamenta Informaticae*, 122(1-2):1–30, 2013.

[26] P.-A. Reynier and J.-M. Talbot. Visibly pushdown transducers with well-nested outputs. In *Proc. 18th International Conference on Developments in Language Theory (DLT'14)*, volume 8633 of *Lecture Notes in Computer Science*, pages 129–141. Springer, 2014.

[27] O. Sankur, P. Bouyer, N. Markey, and P.-A. Reynier. Robust controller synthesis in timed automata. In *Proc. 24th International Conference on Concurrency Theory (CONCUR'13)*, volume 8052 of *Lecture Notes in Computer Science*, pages 546–560. Springer, 2013.