Université libre de Bruxelles, department of Computer Science

PhD thesis

# Contributions to formalisms for the specification and verification of quantitative properties

Nicolas Mazzocchi

This dissertation is submitted on July 14, 2020 in fulfillment of the requirements for the PhD degree of *Docteur en Science* to the jury composed by

| | | |
|---|---|---|
| Emmanuel Filiot | Research Associate, Université libre de Bruxelles | Advisor |
| Thierry Massart | Professor, Université libre de Bruxelles | President |
| Jean-François Raskin | Professor, Université libre de Bruxelles | Advisor |
| Pierre-Alain Reynier | Professor, Aix-Marseille Université | Examiner |
| Sriram Sankaranarayanan | Professor, University Colorado Boulder | Examiner |

# Acknowledgments

My foremost thanks go to my advisors, Emmanuel Filiot and Jean-François Raskin, for accepting me as a PhD student and for guiding me through this thesis. They spent a lot of time to point me out mistakes and weaknesses in my work, but they did it with a flawless patience, precision, and clarity. I feel lucky to have been trained in computer science research by them. I would like now to express my gratitude to the jury members, Emmanuel Filiot, Thierry Massart, Jean-François Raskin, Pierre-Alain Reynier, and Sriram Sankaranarayanan. They questions and reviews allowed me to improve the quality of both my manuscript and my presentation. I address a particular mention to Pierre-Alain for having read deep technicalities of my proofs and came with inspiring comments. I am glad for having worked together with Sriram as well as Ashutosh from Boulder University. I am now looking forward to learning from your knowledge and insights with high curiosity.

During my thesis, I had the opportunity to have fruitful discussions with scientists in conferences and research events, in several counties. My work and these travels have been supported by the Fund for Research training in Industry and Agriculture (FRIA) from the Belgian National Funds for Scientific Research (FNRS). My working days in Brussels were very pleasant thanks to the nice atmosphere created by the verification team. I am happy having met such kind peoples. Guillermo thank you for the references on logarithmic-space shortest path, they were useful to prove the NLogSpace membership for $k$-valuedness of addition systems. Nathan thank you for your help on the undecidability of my expression formalism and for challenging us with coffee-time problems (I have puzzle for you, perhaps for the next Highlight conference). Raphael thank you for your math-jokes, your inexhaustible energy and your incredible sympathy. Mickael thank you for having accepted me as your teaching assistants, it was fun and instructive (after you left, I truly missed your personality). Thank Gilles, Stéphane, Ayrat, Debraj, Leo, I enjoyed a lot our board-game parties, beer evenings and forest walks. Finally, Mrudula and Alexis, I wish you the best of luck with your doctorate and keep in mind that offices 209-217 have an awesome sunset view.

I have now a special thanks to address and I start with the PhD student whom influenced me the most, but not only in the good way, I can confess. The period during I worked with Ismaël was so much satisfying, and even leads us to a publication on Orna's primality problem. I am fascinated by both its brilliant and ironic human-been, although I am not decided on the order. I cannot resist to present one problem that I asked him:

> Given a directed graph with a "source" state $s$ and a "target" state $t$ such that, (1) any pair of run from $s$ to $t$ share at least three states, does (2) there is a "bottleneck" state $q$, i.e. every run from $s$ to $t$ must visit $q$.

I had in mind that (1) is a NP property while (2) is NLogSpace and Ismaël answers me with a penis-shaped counter-example. I challenge you to find it. As an other of my co-author, discussing with Shibashis was scientifically enthralling, and also humanly enriching. Also, I owe him many thanks for his proofread of my introduction. Shibashis has been a great companion, we have invited each other to delicious dinners, in Belgium and in India. All around the world, the verification community is actually quite small. So it can be confusing to distinguish between friends and colleagues. Through the constancy of their support, Luc & Marie were the compass that always showed me the right direction to take. They both did a post-doc at the ULB, left the lab, and now live together. I wish them nothing less than the best. In the last months of my thesis, prisoner of the COVID-19 pandemic, I was struggling to sense the definition of "writing a thesis". At this troublesome moment, Paul & Sarah were the friends I have been waiting for. I have to say that it will never have been nicer to make some redstone contraptions. And, who could predict that the person whom I (awkwardly) asks for an internship, would help me to find motivation during this key period. Sarah & Paul are wonderful people and I am very thankful to them. Finally, I must say that the Marguerit & Rampal families gave me the biggest, the strongest, and the purest happiness of these past years. It hurts me to leave, and that's why I have no doubts to keep your farewell as an indelible memory. The last words go to my family, that I would like to thank as well, in particular to my stepmother Gisèle for her lifestyle choices.

This document has been implemented in a LaTeX environment for which compilation does not necessarily end depending on syntax errors. The design has been mainly done by the use of the awesome packages, *titlesec*, *tikz* and *tcolorbox*. In particular, I made a navigation mechanism, originally made for the writing process, that I decided to leave it active for the readers since I find it cool. Thus, a click on the bottom edge (resp. top) of each page allows going to the next (resp. previous) chapter. And, a click on the right edge (resp. left) of each page allows going to the next (resp. previous) statement. Note that the click position on the sides matter because a page can contain several statements. For a PDF reader in the continuous display mode, the statements can be visited by clicking on the top right/left corners of the screen.

# Table of Contents

# Chapter 1

# Introduction

## 1.1 Formal verification

Computer systems pervaded almost every area of our society. Some of them, called *reactive systems* [HP84], operate as a controller that maintains an interaction with their environment. In particular, they appear in software and hardware critical applications. Think about anti-lock braking system (ABS) of automotive, aircraft autopilots, railway traffic control, intelligent medical devices, or mass production micro-controllers. Any flaw in such systems could lead to an economic disaster or even endanger human lives. Unfortunately, such catastrophic scenarios happens, here are two recent examples:

- Governmental Australian Transport Safety Bureau, aviation safety investigations & report published on December 2011[I]. Page vii: “While the aircraft was in cruise at 37,000 ft, one of the aircraft’s three air data inertial reference units (ADIRUs) started outputting intermittent, incorrect values (spikes) on all flight parameters to other aircraft systems.” Page 54: “Although the unit had transmitted a significant amount of incorrect data to other systems, and was associated with several fault messages, extensive testing did not identify any problems relevant to the occurrence.”
- Governmental US Federal Aviation Administration, airworthiness directives published on May 2015[II]: “a Model 787 airplane that has been powered continuously for 248 days can lose all alternating current (AC) electrical power due to the generator control units (GCUs) simultaneously going into failsafe mode”

Although it is fair to say that aeronautics has made considerable progress in terms of safety thanks to the expertise of the crews and the use of secondary system devices, avoidable errors still exist. In addition, the maintenance task imposed by the US Federal Aviation Administration consisting to reboot periodically each Boeing 787 should never come as a satisfactory answer to a design flaw.

Reactive systems exhibit several characteristics like resources limitation, real-time responsiveness, concurrency that make them difficult to implement correctly. The common techniques in development are based on tests, simulations and declarative assertions. Of course this approach can quickly reveal errors, and thus are crucial for industrial interests. Nevertheless, as noticed by the Australian Transport Safety Bureau report, it cannot show the absence of errors, and that is why it is insufficient for critical applications. To ensure the design of reactive systems that are dependable, safe and efficient, researchers and industrials have advocated the use of so-called *formal methods*, that rely on mathematical models to express precisely and analyze the behaviors of those systems. *Model-checking* is one of the most popular techniques that have proved to be successful for companies like Airbus, Amazon, Facebook, Microsoft, NASA, and have been standardized by industries with the following specification languages: Sugar (IBM), ForSpec (Intel), CBV (Motorola/Freescale), *e* (Verisity/Cadence), OVA (Synopsys/Accelera) as reported in [YPA06]. In contrast to theorem proving assistants where the system and its validation are built together, model-checking algorithms are applied on a preexisting model offering a modular approach.

Let us explain the model-checking approach to formal verification. Given a program together with a set of properties to be verified, called specification, the model-checking problem asks to determine whether all the possible behaviors of the program fulfill the specification. The approach consists of first to abstract the program into a trace-formalism suitable for algorithmic treatment, and also to translate the specification property into a logical language that permits to express properties about such traces. Then, model-checking algorithms check if the set of feasible traces of the model

[I]`https://www.atsb.gov.au/media/3532398/ao2008070.pdf`
[II]`https://www.govinfo.gov/content/pkg/FR-2015-05-01/pdf/2015-10066.pdf`

is included into the set of traces allowed by the specification. In the case of a negative answer, there must exist an erroneous program trace which contradicts the specification, and model-checking algorithm are able then to exhibit a counter-example useful for the designer to correct the bug.

The computer science community acknowledged with ACM Turing awards, three contributions that belongs to the field of the model-checking: In 1976, to Scott and Rabin for their paper on automata theory [RS59]; in 1996 to Pnueli for his work on temporal logic as a seminal specification language for reactive systems [Pnu77]; in 2007 shared by Clarke, Emerson and Sifakis for their contributions to develop model-checking algorithms [CE81, QS82]. Following those theoretical contributions, algorithms and tools have been developed [KVW00, CGP01, BK08] and today, model-checking is a standard approach for program verification routinely used in industry. Most of the past contribution in model-checking has been done in the so-called Boolean setting in which, a trace either satisfies or violates the specification. More recently quantitative extensions of the model-checking to framework have been investigated.

In the next sections, we first review the automata theory approach of the well known Boolean setting and its generalization to the quantitative setting. Then we present how this thesis contributes to expand the quantitative aspects of model-checking.

## 1.2 Automata approach

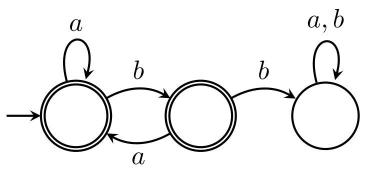
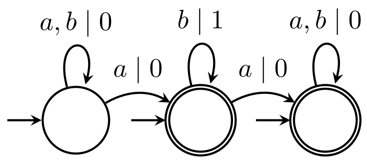
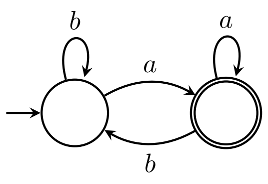
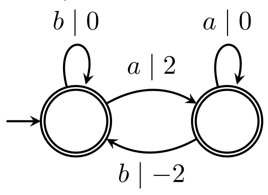
Computer aided-verification aims at using formal methods based on mathematical models to exhaustively and automatically verify the correctness of computer systems, with respect to given specifications. The traditional approach to verification, as proposed by Clarke and Emerson in [CE81], is purely Boolean. Indeed, the specifications express valid behaviors of the systems to be verified and thus induce a partition between correct and incorrect traces. So, the model-checking techniques rely on finite presentations of functions from feasible traces to $\{0, 1\}$ where a trace is denoted by a sequence of events called a word. The formalism that contributed to the rise of model-checking is *finite state automata* well known for its versatileness trough closure properties, a canonical presentation, and equivalences to many other presentations including logics. In the automata theory approach, the model-checking problem reduce to the language *inclusion* as: Does the set of feasible traces defined by the automaton $P$ that models the program is included in the set of traces allowed defined by the automaton $S$ that models the specification. Formally, the language inclusion asks whether $\forall u \;\; u \in L(P) \Rightarrow u \in L(S)$. Simpler problems have also been consider, the language *equivalence*: $\forall u \;\; u \in L(P) \Leftrightarrow u \in L(S)$, the language *universality*: $\forall u \;\; u \in L(P)$, and the language *emptiness*: $\exists u \;\; u \in L(P)$ are the classical decision problems in automata theory.

Quantitative languages of finite words naturally generalize Boolean languages as a more realistic framework in which traces are mapped to a rational value. This approach have applications in modeling resource-consumption (e.g. battery, memory) for embedded systems. Note that, the classical decision problems extend to the quantitative setting in the expected way. The *inclusion* problem asks whether two given quantitative languages $f, g$ satisfy $f \leq g$, i.e. if the domain inclusion $\mathtt{dom}(f) \subseteq \mathtt{dom}(g)$ holds and in that case if $f(u) \leq g(u)$ holds for all $u \in \mathtt{dom}(f)$. The *equivalence* problem asks whether $f(u) \leq g(u)$ and $g(u) \leq f(u)$. The *universality* problem asks whether $\nu \leq f(u)$ for all $u \in \mathtt{dom}(f)$, where $\nu$ is a numerical threshold. And dually, the *emptiness* problem asks whether $\nu \leq f(u)$ for some $u \in \mathtt{dom}(f)$.

The quantitative languages (a.k.a. formal series) have been studied for long in the context of automata theory [Sch61, Cho77a, Ber77]. More recently, what we call now *weighted automata*, received a particular attention from the verification community for their application in modeling system *quality* [DKV09, CDH10b], thus lifting classical Boolean verification problems to a quantitative setting. Nowadays, weighted automata is a classical formalism to define function form word to a numerical value as a generalization of finite automata where transitions are labeled by weights in addition to the input letter. In the general definition, a weighted automaton comes with two operators: (1) one to *aggregate* the weights along a run to provide the quality of a particular trace and (2) another operator to *combine* the values obtained form all accepting runs in order to speak about the whole system. In the realm of the model-checking, the most popular instantiations of operators ((2), (1)) for weighted automata are $(\max, +)$ and dually

$(\min, +)$, denoted respectively $\mathsf{WA}^{\max}_{\text{sum}}$ and $\mathsf{WA}^{\min}_{\text{sum}}$.

In order to highlight similarities and differences between the two settings, the table below shows two implementations (using finite automata and $\mathsf{WA}^{\max}_{\text{sum}}$) where the model-checking problem asks whether every input word for which "bad" letters $b$ are not read twice in a row necessarily ends with a "awesome" letter $a$. We can see that the Boolean setting allows us to handle regular behaviors of event appearance and to count them up to a constant threshold. In addition, the quantitative setting permits to count arbitrary many events. Note that, applications with the need of measuring the performances or consumption of resources, require a model that manipulates unbounded values.

| | Boolean : $\Sigma^* \to \{0,1\}$ finite automata | Quantitative : $\Sigma^* \to \mathbb{Z}$ $\mathsf{WA}^{\max}_{\text{sum}}$ |
|---|---|---|
| Prog. | $P\colon u \mapsto \begin{cases} 0 & \text{if } bb \text{ appears in } u \\ 1 & \text{otherwise} \end{cases}$  | $P\colon u \mapsto \begin{cases} \text{length of the longest} \\ \text{block of } b \text{ in } u \end{cases}$  |
| Spec. | $S\colon u \mapsto \begin{cases} 1 & \text{if } u \text{ ends with } a \\ 0 & \text{otherwise} \end{cases}$  | $S\colon u \mapsto \begin{cases} 2 & \text{if } u \text{ ends with } a \\ 0 & \text{otherwise} \end{cases}$  |
| Verif. | $L(P) \overset{?}{\subseteq} L(S)$ $\left.\begin{array}{l} P(b) = 1 \\ S(b) = 0 \end{array}\right\}$ counter-example | $\forall u \in \Sigma^* \;\; P(u) \overset{?}{\leq} S(u)$ $\left.\begin{array}{l} P(bbba) = 3 \\ S(bbba) = 2 \end{array}\right\}$ counter-example |

## 1.3 Contributions

This section summarizes our contributions. The published versions of the content of this thesis are available on the institutional repository[III] of the *Université libre de Bruxelles* as well as the database[IV] of the computer science bibliography *DBLP*.

### 1.3.1 Weighted formalisms

The $\mathsf{WA}^{\max}_{\text{sum}}$ formalism have been introduced as a generalization of finite automata where a given input word $u$ is associated to the maximal value amongst all sums of weights from accepting runs on $u$. This instance of weighted automata offer few closure under of arithmetical operations due to the "arbitrarily" fixed operators $+$ and max. Moreover, basic and desirable decision problems for $\mathsf{WA}^{\max}_{\text{sum}}$ are already known to be undecidable [Kro94, ABK11]. As a contribution, we introduce a general and as expressive as possible, framework for quantitative verification that retains decidability for model-checking algorithm.

The expression formalism given in [CDE$^+$10] defines quantitative languages by combining the values computed by deterministic sum-automata (i.e. automata that sum integer weighted along its runs) with the operators $+, -, \max$ and $\min$. Consider two deterministic sum-automata, $A$ which counts the occurrence number of $a$ in the input word and $B$ counts the number of $b$. The expression $E = \max\{A, B\}$ defines the function which maps the occurrence number of the most present letter from a given word over

[III]In French, `https://difusion.ulb.ac.be/vufind/Author/Home?author=Mazzocchi,%20Nicolas`
[IV]In English, `https://dblp.uni-trier.de/pers/hd/m/Mazzocchi:Nicolas`

the alphabet $\{a, b\}$. Inspired by this model, we introduce in Chapter 3 an expression formalism that mixes Presburger arithmetic and automata to define quantitative languages i.e. functions from words to integers. In particular, our formalism is closed under all Presburger definable functions and we show as a result that its is decidable for the classical verification problems.

**Monolithic expressions with Presburger combinators**

In Section 3.1, we notice that instead of deterministic weighted automata as in [CDE+10], taking *unambiguous* sum-automata (i.e. automata for which every input has at most one accepting run) does not change the complexity of the classical decision problems but strictly extend the expressive power of the expressions. To combine these atoms, the monolithic expressions that we defined, are allowed to use what we call *Presburger combinators* instead of just fixed operators in [CDE+10]. Presburger combinators are any function definable in existential Presburger arithmetic. For instance, the combinator "greater then else" which takes integers $n, m, x, y$ as inputs and computes `if` $n$ `>` $m$ `then` $x$ `else` $y$ is definable by the formula $\varphi_{gte}(n, m, x, y) = (n > m \wedge r = x) \vee (n \leq m \wedge r = y)$ where $r$ is the result variable. The function $\max\{x, y\}$ is definable by $\varphi_{lte}(x, y, x, y)$ and the absolute value function $\text{abs}(x)$ is definable by $\max\{x, -x\}$. Clearly, Monolithic expressions are strictly more expressive than expressions of [CDE+10],

The largest known class of $\mathsf{WA}^{\max}_{\text{sum}}$ enjoying decidability is that of finitely ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$ (which is expressively equivalent to the class of finite-valued $\mathsf{WA}^{\max}_{\text{sum}}$ for which all the accepting executions over the same input run yields a constant number of different values). Let us precisely state our results. Since any finitely ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$ can be decomposed into a finite union of unambiguous $\mathsf{WA}^{\max}_{\text{sum}}$ [SdS10, FGR14], our formalism captures finitely ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$ automata (by using the Presburger combinator max).

We show that all classical decision problems can be solved in polynomial space, matching the complexity for [CDE+10] provide in [Vel12]. It is important to mention that this complexity result cannot be directly obtained from [Vel12] which is on infinite words with mean-payoff automata as atoms (hence the value of an infinite word is prefix-independent). Instead, we rely on techniques developed in Chapter 2 in the context of Parikh automata [KR03] for which we provide new complexity results.

**Expressions with iterated sum**

The previous expressions are monolithic in the sense that first, some values are computed by weighted automata applied on the whole input word, and once the computation done these values are combined using Presburger combinators. It is not possible to iterate expressions on factors of the input word, and to aggregate all the values computed on these factors, for instance by a sum operation. The basic operator for iteration is that of Kleene star (extended to quantitative languages), which we call more explicitly *iterated-sum.* It has already been defined in [DKV09], and its unambiguous version considered in [AFR14] to obtain an expression formalism equivalent to unambiguous weighted automata.

We investigate in Section 3.2 the extension of monolithic expressions with unambiguous iterated-sum, which we just call iterated-sum in the paper. The idea is, given an expression $E$ which applies on a domain $D$, the expression $E^{\circledast}$ is defined only on words $u$ that can be uniquely decomposed (hence the name unambiguous) into factors $u = u_1 u_2 \dots u_n$ such that $u_i \in D$, and the value of $u$ is then $\sum_{i=1}^{n} E(u_i)$. For example, the function that takes $u_1 \bullet u_2 \bullet \cdots \bullet u_k \bullet$ and returns in how many factors $u_i$ the number of $a$ is greater than the number of $b$, is definable by the expression $E = \big(\psi_{gte}(A, B, 1, 0)\big)^{\circledast}$ where $A$ and $B$ are sum-automata that read words of the form $u\bullet$ and count the number of $a$ and $b$ respectively. Note that, in this example, the subexpression $\psi_{gte}(A, B, 1, 0)$ that reads words of the form $u\bullet$ is applied arbitrarily many time and induces an unique decomposition of the input word.

Unfortunately, we show that such an extension yields undecidability (if two or more iterated sum operations occur in parallel, i.e. not nested, in the expression). The undecidability is caused by the fact that sub-expressions $E^{\circledast}$ may decompose the input word in different ways. We therefore define the class of so called *synchronized* expressions with iterated-sum, which forbids this behavior. We show that while being expressive (for instance, they can define quantitative languages beyond finitely ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$ or $\mathsf{WA}^{\min}_{\text{sum}}$), decidability is recovered.

The proof goes via a new model automata introduced in Section 3.3, called *weighted* *chop automata*, that "chop" the input word into smaller factors, recursively apply smaller chop automata on the factors to compute their values, which are then aggregated by taking their sum. In their synchronized version, we show decidability for chop automata.

## 1.3.2 Robustness

The model-checking has proved to be successful applications for system verification with respect to a given specification. However, this approach does not provide information about "how much correct or incorrect" is the system with respect to its intended specification. This is an important limitation for applications with error-prone systems (as noisy sensors) and imprecise specifications (as statistical observations). As a contribution, we rely on the quantitative framework to lift the verification algorithms to a version robust against disturbances.

In particular, we specify distances between finite words, using the notion of *cost functions*. A cost function assigns a non-negative rational cost to each pair of words $(u_1, u_2)$, modeling the cost of rewriting $u_1$ into $u_2$. By bounding the costs of rewritings, it models how words can be transformed. As a consequence, a neighborhood can be defined for each word, assuming that the cost of "rewriting" a word $u$ back to itself is 0.

A well known formalism to define word-to-word relations is that of *transducers* which generalizes finite automata over two tapes (expected/perturbed) by allowing them to behave independently [Ber79], i.e. without letter-to-letter synchronization. In order to model cost functions, we use *weighted transducers* with non-negative weights [DKV09] along with an *aggregator* (e.g. sum) that combines the cost of each individual rewriting of the transducer into an overall cost between the input word and its perturbed version. Here, the value associated by the transducer to rewrite a word into another, denotes the cost of noise induced by this rewriting.

**Motivation and objectives**

We now provide motivating examples for the cost functions that can be specified by weighted transducers. Consider the transducer $T$ of Figure 1.1. It allows one to rewrite the letter $a_1$ into $a_1$ (at cost 0), and the letter $a_2$ into either $a_2$ (at cost 0) or $b$ (at cost 1). Additionally, rewritings with cost 1 are possible only in the middle state in order to model "bursty" errors, which is common in many situations.

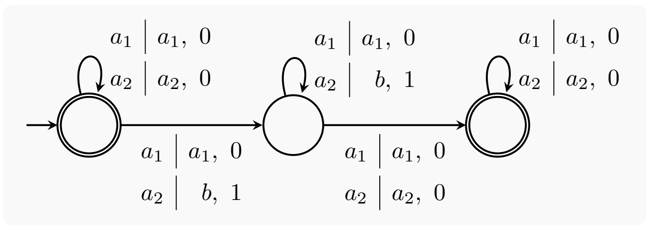


Figure 1.1: A weighted-transducer over alphabet $\Sigma = \{a_1, a_2, b\}$

So, the transducer models all possible words $u'$ that a given input word $u$ can be rewritten into. As an example, the word $u = a_1a_2a_2a_2$ into $u' = a_1bba_2$ through transitions that rewrite the first two occurrences of $a_2$ into $b$. At the same time, the transducer forbids certain rewritings. For instance, the word $u$ above cannot be rewritten into the word $u'' = bba_2a_1$ since the rewrite from an $a_1$ into a $b$ or an $a_2$ into an $a_1$ is clearly disallowed by the transducer $T$ in Figure 1.1. While, the transducer $T$ specifies the cost for individual rewritings through its transitions, we define the cost of rewriting the entire word $u$ into another $u'$ by additionally specifying an *aggregator* function. Since the transducers are not necessarily deterministic, the strategy to choose one value from all possible rewritings in to choose the minimum value. We consider in our work several aggregation functions.

*Discounted sum*: Given a fixed rational *discount factor* $\lambda \in \mathbb{Q} \cap (0, 1)$, the discounted sum aggregator computes the cost of rewriting a word $u$ into another word $u'$ as $\sum_{i=1}^{n} \lambda^{(i-1)} c_i$, wherein $n$ is the size of a run through the transducer and $c_i$ is the cost associated with the $i$th transition.

*Mean*: The mean aggregator computes the average cost: $\frac{1}{n} \sum_{i=1}^{n} c_i$ for $n > 0$.

*Sum* : The sum aggregator computes: $\sum_{i=1}^{n} c_i$ for $n > 0$.

Returning to the example, the sum aggregator of rewriting $a_1a_2a_2a_2$ into $a_1bba_2$ is 2, for the discounted sum it is $\frac{3}{4}$ with discount factor $\frac{1}{2}$, and for the mean it is $\frac{1}{2}$.

This approach considers a transducer model that can allow for insertions of new letters, deletion of letters, transpositions and arbitrary substitutions of one letter by a finite word. As a consequence, many commonly encountered types of distance metrics between words such as the Cantor distance, and the Levenstein (a.k.a. edit) distance, and the average mismatch, can be modeled as weighted transducers.

**Robust verification**

Thanks to our error models, we are able to provide a robust version of the model-checking where not only the set of system traces, but the "neighborhood" of system traces instead, which is a superset obtained by adding some disturbance. Formally, consider a distance $d$ defined by a weighted-transducer with some aggregator function, we define the neighborhood of a language $N$ up to the given error threshold $\nu \in \mathbb{Q}_{\geq 0}$ as $N_\nu = \{u' : \exists u \;\; u \in N \wedge d(u, u') \leq \nu\}$.

For a set of words $N$, a specification $L$, and an error threshold $\nu$, we consider the following robustness problems:

*Robust inclusion*: Does the language robustly satisfies its specification?
Given $N, \nu$ and $L$, check whether $N_\nu \subseteq L$.

*Threshold synthesis*: How large the error can be while keeping the specification satisfy?
Given $N, L$, find the largest threshold $\nu$ such that $N_\nu \subseteq L$.

*Robust kernel synthesis*: For a fixed error bound, what is the robust part of the language that satisfy specification?
Given $N, \nu, L$, synthesize the largest subset $N' \subseteq N$ such that $N'_\nu \subseteq L$. Note that the language $N'$ is not necessary regular.

Consider the transducer of Figure 1.1 using the sum as aggregator and let $\Sigma = \{a_1, a_2, b\}$ its alphabet. We take $L$ as the set of words which do not have *bbb* as a sub-word. Now, any word of the form $(a_1a_2)^*$ is $\nu$-robust for any threshold $\nu$ since the letter $a_1$ is not rewritten by the transducer $T$. Such questions are tackled using the robust inclusion problem. On the other hand, let us choose a word $u \in a_2a_2a_2(a_1^*)$. It is $\nu$-robust for all the thresholds $\nu \leq 2$ but not for $\nu \geq 3$. This is determined using the threshold synthesis problem. For all $\nu \geq 3$, the set of $\nu$-robust words in $N = \Sigma^*$ is $(a_1 + a_1a_2 + a_1a_2a_2)^*$, and for $\nu \leq 2$, any word in $\Sigma^*$ is $\nu$-robust. Such questions are solved using the robust kernel synthesis problem.

As a result, we show that the robust inclusion problem $N_\nu \subseteq L$ is solvable in polynomial time when the language $N$ is given by a non-deterministic finite automaton, and $L$ is given by a deterministic finite automaton, and the weighted-transducer defining the noise is also given as input. To obtain this result, we show that we can effectively compute in polynomial time the largest threshold $\nu$ as defined before, thus solving the threshold synthesis problem. This result holds for the three measures `Sum`, `DSum` and `Mean`. For `Sum`, we show that the robust kernel is regular and computable and testing its non-emptiness can be done with a polynomial space memory. For `Mean`, we show that the robust kernel is not regular in general, and its non-emptiness is undecidable. For `DSum`, we leave those questions partially open. We conjecture that the robust kernel is non-regular in general. Finally, we provide sufficient conditions under which the robust kernel becomes regular for both `Mean` and `DSum` measures.

**Case Studies**

The algorithms to synthesize robustness thresholds have been implemented by the researchers Sankaranarayanan and Trivedi from the university Colorado Boulder. They reported some experiments on two application cases including modeling human control failures and approximate recognition of type-1 diabetes from blood sugar level swings.[V]

Properties are often specified by engineers based on nominal or expected outcomes of a system. However, in engineering practice, it is quite common to state properties that are conservative, i.e. a violation of a property may not always be problematic. Consider for instance a common time reactivity property for an automotive transmission system: Whenever the engine revolutions per minute (RPM) goes above 9000 RPMs and

[V]Publicly available dataset of blood glucose values for people with type-1 diabetes the vehicle is in gear $G_i$ where $i \in \{1, 2, 3, 4\}$, the vehicle will shift to the next higher gear $G_{i+1}$ within 25 cycles. The property has "magic numbers" that include the 9000 RPM and the 25 cycle limits. These are often conservative guesses derived by engineers to account for a large number of unknown and unmodeled factors that may affect the possibility of short term and long term engine damage. As a consequence, engineers may accept a design that violates this property but satisfies a weaker property wherein whenever the engine RPM goes above 9050, the design will shift to the next higher gear within 30 cycles in the worst case.

In other situations, the property is well-established and has little leeway due to the inherent noise of the data obtained from sensors. In such scenarios, engineers would like a built-in *tolerance* wherein the system must actually satisfy a stronger property. Consider for instance the following clinical property on insulin pump: If the blood glucose (BG) levels of a patient fall below 3.9 `mmol/L`, the pump must be shutdown within 10 minutes. In such an example, this requirement is absolutely critical for patient's safety and well-being. As a consequence, engineers would prefer a design that shuts down a pump within 5 minutes rather than a design that shuts it down in 8 minutes. Also, engineers may raise the shutdown limit to 4.2 `mmol/L` rather than 3.9 `mmol/L`.

These examples highlight that not all violations of a specification should be treated equally. The flexibility in specifications and the approximate nature of the models, lead us to seek quantitative and robust versions of model-checking techniques.

### 1.3.3 Structural properties

The definition of automata subclasses with particular properties or with better algorithmic tractability is an important aspect of automata theory. For example, the language inclusion problem can be decided in polynomial time for any finite automata with a constant ambiguity [SI85] while this complexity cannot be achieved in the general non-deterministic case [SM73, Koz77]. As contribution, we introduce a specification logic to define structural properties for popular models of automata.

Here, we mean automata in the general sense of finite state models processing finite words. This includes what we call *automata with outputs*, which may also produce output values in a fixed monoid $(D, \oplus, \mathbb{0})$. In such an automaton, the transitions are extended with an (output) value in $D$, and the value of an accepting path is the sum (for $\oplus$) of all the values occurring along its transitions. Automata over finite words in $\Sigma^*$ and with outputs in $D$ define subsets of $\Sigma^* \times D$ as follows: to any input word $u \in \Sigma^*$, we associate the set of values of all the accepting paths on $u$. For example, transducers are automata with outputs in a free monoid: they process input words and produce output words and therefore define binary relations of finite words [Ber79].

**Logics to express patterns**

We observed that subclasses of automata with outputs are in general characterized by structural patterns and the recognition of such patterns often share similar proof techniques. So, we introduce a generic logic, denoted $\mathsf{PL}[\mathcal{O}]$ for "pattern logic", to express such a properties in a given monoid $(D, \oplus, \mathbb{0})$. In particular, $\mathsf{PL}$ is parameterized by $\mathcal{O}$, which is the set of predicates talking about the output values computed with $\oplus$.

We focus on three particular instances of automata with outputs: finite automata (which can be seen as automata with outputs in a trivial monoid with a single element), transducers (automata with outputs in a free monoid), and sum-automata (automata with outputs in $(\mathbb{Z}, +, 0)$). For each of them, we define particular logics, respectively called $\mathsf{PL}_{\text{nfa}}$, $\mathsf{PL}_{\text{trans}}$ and $\mathsf{PL}_{\text{sum}}$ to express properties of automata with outputs in these particular monoids. Formulas in these logics have the following form:

$$\exists \pi_1 : p_1 \xrightarrow{u_1 | v_1} q_1,\ \ldots,\ \exists \pi_n : p_n \xrightarrow{u_n | v_n} q_n,\ \mathbb{C}$$

where the $\pi_i$ are path variables, the $p_i, q_i$ are state variables, the $u_i$ are (input) word variables and the $v_i$ are output value variables interpreted with respect to corresponding monoid. The sub-formula $\mathbb{C}$ is a quantifier free Boolean combination of predicates talking about states, paths, input words and output values. Such a formula expresses the fact that there exists a path $\pi_1$ from some state $p_1$ to some state $q_1$, over some input word $u_1$, producing some value $v_1$, some path $\pi_2$ and so on, such that they all satisfy the constraints in $\mathbb{C}$. In the three logics, $\mathbb{C}$ can also express path constraints such as path equality, input constraints using the word prefix relation, the length of those words, as well as membership to a finite language. Finally, $\mathbb{C}$ can contain state constraints such as state equality, and whether a state is initial or final.

The predicates we consider for the output values depend on the monoids. For transducers, output words can be compared with the non-prefix relation (and by derivation $\neq$), a predicate which cannot be negated (otherwise model-checking becomes undecidable), and can also be compared with respect to their length, and membership to a finite language can be tested. For sum-automata, the output values can be compared with $\leq$ (and by derivation $=, \neq, <$). As an example, a transducer (resp. sum-automaton) is *not* functional iff it satisfies the following $\mathsf{PL}_{\text{trans}}$-formula (resp. $\mathsf{PL}_{\text{sum}}$-formula):

$$\exists \pi_1 : p_1 \xrightarrow{u|v_1} q_1,\ \exists \pi_2 : p_2 \xrightarrow{u|v_2} q_2,\ v_1 \neq v_2 \wedge \begin{cases} \mathsf{init}(p_1) \wedge \mathsf{final}(q_1) \\ \mathsf{init}(p_2) \wedge \mathsf{final}(q_2) \end{cases}$$

As a result we show that deciding whether a given automaton satisfies a given formula is PSpace-C for the three logics. When the formula is *fixed*, the model-checking problem becomes NLogSpace-C for $\mathsf{PL}_{\text{nfa}}$ and $\mathsf{PL}_{\text{trans}}$, and NP-C for $\mathsf{PL}_{\text{sum}}$. If output values can only be compared via disequality $\neq$ (which cannot be negated), then $\mathsf{PL}_{\text{sum}}$ admits NLogSpace-C model-checking as well.

### Subclasses defined by pattern

Many decidable results of finite automata do not carry over to transducers and weighted automata. In order to recover decidability or just to define subclasses relevant to particular applications, some restrictions have been defined in the literature.

For transducers, the inclusion problem is undecidable [Gri68], but decidable for *finite-valued* transducers [Web93], that is transducers which associate a bounded number of outputs for any input. Another well-known subclass is that of the *determinizable* transducers [BCPS03], defining sequential functions of words. Finite-valuedness and determinizability are two properties decidable in polynomial time.

For addition systems, i.e. automata that sum integer weights along its runs, which define relations from words to integers, properties such as functionality, determinizability, and valuedness bounded by some fixed natural $k$ are decidable in polynomial time [FGR14, FGR15, DJRV17].

In our experience, it is quite often the case that deciding a subclass goes in two steps: (1) define a characterization of the subclass through a "simple" pattern, (2) show how to decide the existence of a such a pattern. For instance, the determinizable transducers have been characterized via the so called *twinning property* [BC02, Cho77b, WK95] which, loosely speaking, asks that the output words produced by any two different paths on input words of the form $uv^n$ cannot differ boundlessly when $n$ grows, with a suitable definition of "differ". Quite often, the most difficult part is step (1) and step (2) is technical but less difficult to achieve, as long as we do not seek for optimal complexity bounds (by this we mean that polynomial time is good enough, and obtaining the best polynomial degree is not the objective). We even noticed that in transducer theory, even though step (2) share common techniques (reduction to emptiness of reversal-bounded counter machines for instance), the algorithms are often ad-hoc to the particular subclass considered.

The last contribution of this thesis introduce logics tailored to particular monoids, as a common tool for step (2). More precisely, the $\mathsf{PL}$ allows us to express subclass characterized through a structural pattern in a generic way and provides automatically a decision procedure for the subclass membership. Here is a non-exhaustive list of criteria each expressible in our logics and for which the NLogSpace complexity of deciding the subclasses membership comes as a direct consequence of our results:

*For finite automata*: ambiguity bounded by a given natural $k$, finite ambiguity [WS91], polynomial ambiguity [WS91], exponential ambiguity.

*For transducer*: determinizability [AM03, BC02, BCPS03, Cho77b, CS86, WK95], functionality [BC02, BCPS03], degree of sequentiality bounded by a given natural $k$ [DJRV17], multi-sequentiality [CS86, JF18], valuedness bounded by a given natural $k$ [GI83], finite valuedness [SdS08, Web93].

*For addition sytems*: functionality [FGR15], valuedness bounded by a given natural $k$ [FGR14], degree sequentiality bounded by a given natural $k$ [DJRV17].

## 1.4 Preliminaries

In this section we introduce the basics.

**Sets, monoids and numbers**

Let $I, J$ be two sets. We write $I \cup J$, $I \uplus J$, $I \cap J$, $I \setminus J$, $I \times J$ and $\overline{I}$ respectively the union, the disjoint union, the intersection, the difference, the Cartesian product of $I, J$ and the complement of $I$. We denote by $\texttt{card}(I)$ the cardinal of $I$ i.e. the number of elements which belongs to it. In particular, we refer to $\varnothing$ as the empty set.

The sets of numerical values $\mathbb{Z}, \mathbb{N}, \mathbb{Q}$ denote respectively the set of integers, the set of non-negative integers and the set of rational numbers. We write $\mathbb{Z}_{\neq 0}, \mathbb{N}_{\neq 0}, \mathbb{Q}_{\neq 0}$ to refer the same sets from which 0 has been removed. We use the standard symbols $+, -, \times, \div$ and $\leq$ to refer respectively to the sum, minus, product, Euclidean division operations and the order predicate over numerical values. For $i, j \in \mathbb{Q}$ the interval $\{i, \ldots, j\}$ refer to the set of values $\{k \in \mathbb{Q} : i \leq k \leq j\}$. The rest of the Euclidean division $i \mod j$ is defined as $r = i - mj$ for some $m \in \mathbb{Z}$. The absolute value of $n \in \mathbb{Q}$, denoted $\text{abs}(n)$, is defined as $n$ if $n \leq 0$ otherwise it is $-n$.

A *monoid* is a tuple $(D, \odot, \mathbb{1})$ where $D$ is a set of elements, the binary function $\odot \colon D \times D \to D$ is an associative operation for which $\mathbb{1} \in D$ is neutral. Formally, $(x \odot y) \odot z = x \odot (y \odot z)$ and $\mathbb{1} \odot x = x \odot \mathbb{1} = x$ for all $x, y, z \in D$. The *rational* subsets of a monoid $(D, \odot, \mathbb{1})$ is the smallest set that contains every finite subset of $D$ and is closed under union, $\odot$ and (Kleene) starring, i.e. for all rational sets $A, B$ we have $A \cup B$, $A \odot B = \{a \odot b : a \in A \wedge b \in B\}$, and $A^* = \bigcup_{i=0}^{+\infty} \bigodot_{j=1}^{i} A$ rational as well.

A *semi-ring* is a tuple $(D, \oplus, \odot, \mathbb{0}, \mathbb{1})$ where $(D, \odot, \mathbb{1})$ and $(D, \oplus, \mathbb{0})$ are monoids, the function $\oplus$ is commutative, $\odot$ distribute over $\oplus$ and $\mathbb{0}$ is absorbing for $\odot$. Formally, for all $x, y, z \in D$ we have the commutativity $x \oplus y = y \oplus x$, the left distributivity $x \odot (y \oplus z) = (x \odot y) \oplus (x \odot z)$ and the right one $(x \oplus y) \odot z = (x \odot z) \oplus (y \odot z)$, finally the absorption $\mathbb{0} \odot x = x \odot \mathbb{0} = \mathbb{0}$.

### 1.4.1 Words, languages and decision problems

An *alphabet* is a set of elements called *symbol* or *letter*. A *word* over an alphabet $\Sigma$, is a (partial) function from $\mathbb{N}_{\neq 0}$ to $\Sigma$. We write $a_1 a_2 \ldots a_n$ for defining the word $u \colon i \mapsto a_i$ where $i$ is called a *position* belonging to $\{1, \ldots, n\}$ that is, the domain of $u$. The set of all finite words over $\Sigma$ is denoted $\Sigma^*$ and we refer the *empty word* by $\varepsilon$ which is the function of domain $\varnothing$. The *length* of a word $u$, denoted $|u|$, is the cardinal of its domain. For all $a \in \Sigma$, we also write $|u|_a$ for the number of occurrences of the letter $a$ in $u$ i.e. the number of position that have $a$ as image by $u$. In particular $|\varepsilon| = 0$. The *concatenation* of two words $a_1 \ldots a_n$ and $b_1 \ldots b_m$ is the word $a_1 \ldots a_n b_1 \ldots b_m$. Also, $u^n \in \Sigma^*$ is $n^{\text{th}}$ iterated concatenation of the word $u$, inductively defined by $u^0 = \varepsilon$ and $u^n = uu^{n-1}$. The *prefix* binary predicate $\sqsubseteq$ holds for two word $u_1, u_2 \in \Sigma^*$ if $u_2 = u_1 u$ for some $u \in \Sigma^*$.

A (Boolean) *language* is a set of words. In particular, the empty language corresponds to the empty set $\varnothing$. The word concatenation can be naturally extended to languages as $L_1 \cdot L_2 = \{u_1 \cdot u_2 : u_1 \in L_1 \wedge u_2 \in L_2\}$. As for words, we write $L^n$ to refer the $n^{\text{th}}$ iterated concatenation of $L$. The *star* (unary) operator is defined as the upper closure $L^* = \bigcup_{i=0}^{\infty} L^n$. Note the consistency with the set of all words over $\Sigma$ denoted $\Sigma^*$. The set of languages over $\Sigma$ that are said to be *regular* is the smallest set that contains the empty-language $\varnothing$, the language singleton $\{a\}$ for each $a \in \Sigma$ and it is closed under union, concatenation and star operations. Note that, regular languages over $\Sigma$ coincide with the rational subsets of the free monoid $(\Sigma^*, \cdot, \varepsilon)$. A *quantitative* language[VI] is a partial function $f \colon \Sigma^* \to \mathbb{Z}$, whose effective domain is denoted by $\texttt{dom}(f)$. E.g. consider the quantitative language mapping any word $u \in \Sigma^*$ to the number of occurrences $|u|_a$ of some symbol $a \in \Sigma$ in $u$. Note that, a Boolean language $L$ can be seen as a total function $f \colon \Sigma^* \to \mathbb{Z}$ such that $f(u) = 1$ if $u \in L$ otherwise $f(u) = 0$.

The classical language decision problems are the following:

- The *inclusion* problem asks whether two given $f, g \colon \Sigma^* \to \mathbb{Z}$ satisfy $f \leq g$, i.e. whether $\texttt{dom}(f) \subseteq \texttt{dom}(g)$ and in that case whether $f(u) \leq g(u)$ for all $u \in \texttt{dom}(f)$.
- The *equivalence* problem asks whether $f \leq g$ and $g \leq f$.
- The *universality* problem asks, given some threshold value $\nu \in \mathbb{Z}$ and a language $f \colon \Sigma^* \to \mathbb{Z}$, whether $\nu \leq f(u)$ for all $u \in \texttt{dom}(f)$.

[VI]Also called *formal series* in [DKV09]

- The *emptiness* problem asks whether $\nu \leq f(u)$ for some $u \in \texttt{dom}(f)$.

For the case of Boolean languages, the only relevant threshold value is $\nu = 1$.

### 1.4.2 Automata formalisms

We introduce a general formalism, with transitions labeled by a given monoid, from which all our automata models are derived. Transition systems are parameterized by a monoid and any execution is associated with an element of the monoid, obtained by aggregating (thanks to $\odot$) the values met along the transitions.

**Definition – labeled transition system**

> A *transition system* $G$ is a tuple $(Q, Q_I, Q_F, \Delta)$ where $Q$ is the set of states, $Q_I \subseteq Q$ the set of initial states, $Q_F \subseteq Q$ the set of final states and $\Delta \subseteq Q \times Q$ is the transition relation. A *labeling* of $G$ over a monoid $(D, \odot, \mathbb{1})$ is a relation $\lambda \subseteq \Delta \times D$.

A *run* $\varrho$ is a sequence of transitions $t_1 \dots t_n \in \Delta^*$ for which there exists $q_0, \dots, q_n \in Q$ called the *visited states* of $\varrho$ such that $t_i = (q_{i-1}, q_i)$ for all $i \in \{1, \dots, n\}$. We refer to the *starting state* $q_0$ as $\varrho^{\lhd}$ and its *ending state* $q_n$ by $\varrho^{\rhd}$. A *cycle* is a run for which if $\varrho^{\lhd} = \varrho^{\rhd}$. If no state is visited twice i.e. $i \neq j$ implies that $q_i \neq q_j$ then $\varrho$ is said to be a *simple* or equivalently *cycle free*. If in addition the run $\varrho = t_1 \dots t_n$ is a cycle and $t_2 \dots t_n$ is cycle free, then $\varrho$ is said to be a *simple cycle*. Note that the empty run $\varepsilon$ does not have starting and ending states, and thus it is not a cycle.

The transition relation $\lambda$ can be inductively extended to support runs by $\lambda\colon \varepsilon \mapsto \{\mathbb{1}\}$ and $\lambda\colon t\varrho \mapsto \{\ell_t \odot \ell_\varrho : (t, \ell_t) \in \lambda \wedge (\varrho, \ell_\varrho) \in \lambda\}$. For convenience, we denote by $S_I \xrightarrow{u} S_F$ the set of runs from some state of $S_I \subseteq Q$ to some state of $S_F \subseteq Q$ and labeled by $u \in D$.

$$S_I \xrightarrow{u} S_F \overset{\text{def}}{=} \{\varrho \in \Delta^+ : \varrho^{\lhd} \in S_I \wedge \varrho^{\rhd} \in S_F \wedge u \in \lambda(\varrho)\} \cup \{\varepsilon : S_I \cap S_F \neq \varnothing \wedge u = \mathbb{1}\}$$

We may forget the label constraint and write $S_I \rightarrow S_F$ to refer to $\bigcup_{u \in D} S_I \xrightarrow{u} S_F$. The run $\varrho$ is said to be *initial* if $\varrho \in Q_I \rightarrow Q$, *final* if $\varrho \in Q \rightarrow Q_F$ and *accepting* if it is both. We denote by $\texttt{AccRun}_G$ the set of accepting runs of $G$ and $\texttt{AccRun}_G(u)$ the set of accepting runs of $G$ on $u$.

A state $q \in Q$ is said to be *reachable* from $q' \in Q$ if $\{q'\} \rightarrow \{q\}$ is not empty. A state $q \in Q$ is said to be *accessible* if there exists an initial run which visits it, i.e. if $Q_I \rightarrow \{q\}$ is not empty. Dually, a state $q \in Q$ is said to be *co-accessible* if there exists a final run which starts from it, i.e. if $\{q\} \rightarrow Q_F$ is not empty. A transition system is said to be *trim* if all states are both accessible and co-accessible i.e. $(Q_I \rightarrow \{q\}) \neq \varnothing \wedge (\{q\} \rightarrow Q_F) \neq \varnothing$ for all $q \in Q$.

Since the representation size of $G$ depends on its monoid, we prefer to define it once instantiated[VII].

**Definition – regular automata**

> A finite *regular automaton* $A$ (NFA for short) over the alphabet $\Sigma$ is defined as a transition system labeled by the free monoid $(\Sigma^*, \cdot, \varepsilon)$ such that $\lambda \subseteq \Delta \times (\Sigma \cup \{\varepsilon\})$. It is represented by the tuple $(Q, Q_I, Q_F, \Delta)$ where the relation $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ denotes the set of transitions labeled over $\Sigma$.

Semantically, an automaton $A$ denotes the language $L(A)$ defined as $\{u \in \Sigma^* : \texttt{AccRun}_A(u) \neq \varnothing\}$. The representation size of $A$ is defined by $|A| = \texttt{card}(Q)^3\texttt{card}(\Sigma)$. We say that $A$ is *deterministic* (DFA for short) if $Q_I$ is a singleton and $\Delta$ is a function from $Q \times (\Sigma \cup \{\varepsilon\})$ to $Q$. For $k \in \mathbb{N}_{\neq 0}$, $A$ is said to be *k-unambiguous* (k-UFA for short) if for every input word there exists at most $k$ accepting runs, formally $\texttt{card}(\texttt{AccRun}_A(u)) \leq k$ for all $u \in \Sigma^*$. We prefer to call an automaton *unambiguous* (UFA for short) when it is 1-ambiguous and *finitely unambiguous* when it is $k$-ambiguous for some $k$. Note that, the ambiguity is a weaker restriction than determinism, that asks for uniform bound on the degree of non-determinism.

[VII]Basically the representation size will be $\texttt{card}(Q) + \texttt{card}(\Delta) \times \mu$ where $\mu$ is the "biggest" possible labeling

**Definition – regular automata with outputs**

> An automaton with outputs $A$ over a monoid $(D, \odot, \mathbb{1})$ is a tuple $(Q, Q_I, Q_F, \Delta, \lambda)$ where $(Q, Q_I, Q_F, \Delta)$ is an NFA called the *underlying* automaton, the function $\lambda\colon \Delta \to D$ maps each transition to an output element.[VIII]

Given a run $\varrho = t_1 \dots t_n \in \Delta^*$ of $A$, the output of $\varrho$ is defined by $\lambda(\varrho) = \bigodot_{i=1}^{n} \lambda(t_i)$ if $n > 0$, and by $\mathbb{1}$ if $n = 0$. We write $\texttt{AccRun}_A(u, v)$ to refer to the set of accepting runs on $u$ outputting $v$ i.e. $\{\varrho : \varrho \in Q_I \xrightarrow{u} Q_F \wedge \lambda(\varrho) = v\}$.

Semantically, an automaton with outputs $A$ denotes an input/output relation $[\![A]\!]$ defined as $\{(u, v) : \texttt{AccRun}_A(u, v) \neq \varnothing\}$. The domain and the range of $A$ are respectively the projection $\texttt{dom}(A) = \{u : (u, v) \in [\![A]\!]\}$ on the inputs and $R(A) = \{v : (u, v) \in [\![A]\!]\}$ on the outputs. We define the notion of valuedness that refer to a bound on the cardinal of the sets associated to any input word. An automaton with outputs $A$ is said to be *$k$-valued* if for all input word $u \in \texttt{dom}(A)$, the set $\{v : (u, v) \in [\![A]\!]\}$ has cardinality at most $k$. Note that it does not imply that there are at most $k$ accepting runs since, two distinct runs can output the same value. However, any $k$-ambiguous automaton with output is $k$-valued. We call an automaton *functional* when it is 1-valued[IX] and *finitely valued* when it is $k$-valued for some $k$. Let $\Gamma = \{\lambda(t) \in D : t \in \Delta\}$. The representation size of an automaton with outputs is $|A| = \texttt{card}(Q)^3 \texttt{card}(\Sigma) \ell \mu$ where $\ell = \texttt{card}(\Gamma)$ is the number of distinct output labels and $\mu = \max\{|\gamma| : \gamma \in \Gamma\}$ is the maximal size of labels of $A$[X].

In this thesis, we mostly consider three instances of automata with outputs. First, *transducers* are automata with outputs in the free monoid $(\Gamma^*, \cdot, \varepsilon)$, they define relations from $\Sigma^*$ to $\Gamma^*$ and the size of an output word is its length. We refer the reader for instance to [BCPS03] for a definition of finite transducers. Secondly, *sum-automata* of dimension $d \in \mathbb{N}_{\neq 0}$, are automata with outputs in the monoid $(\mathbb{Z}^d, +_{\mathbb{Z}^d}, 0_{\mathbb{Z}^d})$ and define relations from $\Sigma^*$ to $\mathbb{Z}^d$. We consider two representation size of an integer value $v$, with a binary encoding the size is $\log_2(v+1)$ while in unary it is $v+1$. Finally, remark that NFA compiles in the definition of automata with outputs in a trivial monoid. In particular, the synchronized product is also well defined for NFA.

**Product construction**

The *product construction* of $n$ automata $A_i = (Q_i, I_i, F_i, \Delta_i, \lambda_i)$ with outputs in $(D, \odot, \mathbb{1})$, denoted $A_1 \times \cdots \times A_n$, is the automaton $A = (Q, I, F, \Delta, \lambda)$ over the monoid $(D^n, \odot_{D^n}, \mathbb{1}_{D^n})$, where $\odot_{D^n}$ is the component-wise $\odot$. The three sets of states are sets of tuples respectively $Q = Q_1 \times \cdots \times Q_n$ and $I = I_1 \times \cdots \times I_n$ as well as $F = F_1 \times \cdots \times F_n$. The transition relation is defined by $\Delta = \{(q, a, p) \in Q \times (\Sigma \cup \{\varepsilon\} \times Q : \bigwedge_{i=1}^{n} (q[i], a, p[i]) \in \Delta_i\}$ and $\lambda\colon (q_1, \dots, q_n), a, (q'_1, \dots, q'_n) \mapsto \lambda_1(q_1, a, q'_1), \dots, \lambda_n(q_n, a, q'_n) \in D^n$ for each transition $(q_i, a, q'_i)$ of $A_i$.

**Definition – weighted automata**

> A *weighted automaton* $W$ over the alphabet $\Sigma$ and with outputs in the semi-ring $(D, \oplus, \odot, \mathbb{0}, \mathbb{1})$ is defined as an automaton with outputs over the monoid $(D, \odot, \mathbb{1})$.

A weighted automaton differ from automaton with outputs in its semantics, because it aggregates all values associated to an input thanks to its (associative) second operator $\oplus$. In particular, a weighted automaton $W$ denotes a (partial) function[XI] which is the quantitative language $[\![W]\!]\colon \texttt{dom}(A) \to D$ defined by $u \mapsto \bigoplus\{v \in R(u)\}$.[XII] Note that, the semantics of automata with outputs and weighted automata coincide for they unambiguous fragment i.e. when automata admit at most one accepting run on any input

[VIII]Sometimes, initial and final weight functions are considered in the literature to assign values distinct from $\mathbb{1}$ to $\varepsilon$, which can also be done by directly providing the set of outputs assigned to $\varepsilon$ and considering it as a special case.

[IX]If $A$ is functional when $[\![A]\!]$ is a function

[X]For implementation matter, some of our algorithms require a distinct memorization of outputs and transitions where each transition only carry a key to refer its corresponding outputs in a hash table that memorize all distinct outputs once.

[XI]Sometimes in the literature, the semantics of weighted automata are lifted to total functions by assigning $\mathbb{0}$ for all input out of the domain.

[XII]Since $\oplus$ is associative, the order in which the values from accepting runs are combined does not matter.

word. Indeed, the $\oplus$ operation is not used to compute the output value of a word which, in this case, corresponds exactly to the value of the unique accepting run. Formally, let $(D, \oplus, \odot, \mathbb{0}, \mathbb{1})$ be a semi-ring and consider $A$ as an automaton with outputs in $(D, \odot, \mathbb{1})$. If $A$ is unambiguous, then $[\![A]\!]$ is a function and thus $[\![A]\!]\colon u \mapsto R(u) = \bigoplus\{v \in R(u)\}$.

In this thesis, we only consider the tropical semi-rings as instances of weighted automata. Let the two pairs of operators (min, $+$) and (max, $+$) define the classes of weighted automata $\mathsf{WA}^{\mathrm{min}}_{\mathrm{sum}}$ and $\mathsf{WA}^{\mathrm{max}}_{\mathrm{sum}}$ respectively. We will be clear when the domain of weights is $\mathbb{N}$ and when it is $\mathbb{Z}$. We refer the reader for instance to [DKV09] for a definition of finite weighted automata.

# I

# Weighted formalisms

# Chapter 2

# Semi-linearity

Model-checking belongs to the main techniques in program verification and its classical approach is elegantly and efficiently supported by regular automata-theoretic methods [CGP01, KVW00]. However, its extension to the quantitative setting requires to capture situations beyond regularity. In this chapter, we consider a model of automata that allows us to deal with unbounded numerical values and so, to provide us with a tool that manipulates an infinite-state space.

Semi-linear sets are the rational subsets of the monoid $(\mathbb{Z}^d, +, 0)$ with $d \in \mathbb{N}_{\neq 0}$ as the considered dimension. Parikh investigated these sets to highlight properties on context-free[1] languages [Par66], todays known as *Parikh's theorem*, and in turn brings decidability results for the emptiness problem. Furthermore, semi-linearity is supported by logical formalisms, in particular, [GS66] proved that the semi-linear sets coincide with the relations definable in the Presburger arithmetic [Pre29].

We first review some results on semi-linearity as the foundation of our technical contributions and then we present the formalism of Parikh automata [KR03], which extends regular automata by counting the use of each transition and constraining this occurrence values to belong to a given semi-linear set. Note that, we provide the descriptive complexity of the emptiness problem for Parikh automata as key results for proofs in Chapters 3 and 7.

## 2.1 Representations of semi-linear sets

Semi-linear sets of numbers are finite unions of ultimately periodic sets and can be presented by a finite set of vectors. In [GS64], authors prove that the class of semi-linear sets is exactly the class of relations definable in the Presburger arithmetic. In this section we present some representations of semi-linear sets and their properties.

**Definition – linear & semi-linear sets**

Consider $d \in \mathbb{N}$ to be the dimension. A set $S \subseteq \mathbb{Z}^d$ is *linear* if there exists $b \in \mathbb{Z}^d$ called the *base* vector and $p_1, \ldots, p_n \in \mathbb{Z}^d$ called the *period* vectors such that $S = \{b + \sum_{i=1}^{n} x_i p_i : x_1, \ldots, x_n \in \mathbb{N}\}$. A set is *semi-linear* if it is a finite union of linear sets.

The most natural representation of linear sets is by providing directly its base and its periods. Given $d \in \mathbb{N}$, we denote by $\mathrm{LinSet}\,(b \mid P)$ the linear set of vector base $b \in \mathbb{Z}^d$ and with vector periods belonging to $P \subset \mathbb{Z}^d$. We define $||S||$ as the maximal absolute value appearing in dimensions of vectors of $P$ and $b$. The representation size of a linear set $S$ of dimension $d$ is defined by $|S| = (\mathtt{card}(P) + 1) \times d \times \log_2(||S|| + 1)$.

Let $A, B$ be two semi-linear sets. The sum $A+B$ is defined by $\{x+y : x \in A \wedge y \in B\}$. The sum of $A$ iterated $n$ times is inductively defined by $0_{\mathbb{Z}^d}$ if $n = 0$ and $\sum_{i=1}^{n} A$. The starring of $A$, denoted $A^*$, is the union of all finite iteration of $A$ defined by $\bigcup_{i=0}^{+\infty} \sum_{j=1}^{i} A$.

**Fact 2.1.1 – Closures of semi-linear sets**

The class of semi-linear sets are close under union, sum and starring.

**Proof** The closure under union is trivial from the definition. Let $d \in \mathbb{N}_{\neq 0}$ be the dimensions for our semi-linear sets. Consider $A, B \subseteq \mathbb{Z}^d$ defined by $A = \bigcup_{i=0}^{k} \mathrm{LinSet}\,(a_{0,i} \mid \{a_{1,i}, \ldots, a_{n,i}\})$ and $B = \bigcup_{j=0}^{\ell} \mathrm{LinSet}\,(b_{0,j} \mid \{b_{1,j}, \ldots, b_{m,j}\})$. Then

[1] Context-free languages are definable by regular automata with a stack pushdown, raising a more expressive class of languages than the regular one.

the sum $A + B$ can be presented as follows.

$$A + B = \bigcup_{i=0}^{k} \bigcup_{j=0}^{\ell} \mathrm{LinSet}\left(a_{0,i} + b_{0,j} \mid \{a_{1,i}, \dots, a_{n,i}\} \cup \{b_{1,j}, \dots, b_{m,j}\}\right)$$

Finally, for the starring operation, we note that $(A \cup B)^* = A^* + B^*$ for all $A, B \subseteq \mathbb{Z}$ by commutativity of the sum and union. So, we only need to provide the closure under starring for linear sets which consists to the following.

$$\mathrm{LinSet}\left(a_{0,i} \mid \{a_{1,i}, \dots, a_{n,i}\}\right)^* = \{0_{\mathbb{Z}^d}\} \cup \mathrm{LinSet}\left(a_{0,i} \mid \{a_{0,i}\} \cup \{a_{1,i}, \dots, a_{n,i}\}\right)$$

♦

An important property of linear sets is that they admit a bound on the number of periods as formalized in the following statement.

**Claim 2.1.2 – Carathéodory's property from [ES06]**

Let $L = \mathrm{LinSet}(b \mid P)$ be a linear set with $k = \texttt{card}(P)$ as cardinal of $P$ and $\mu = ||P||$ as the maximal absolute value appearing in dimensions of vectors of $P$. Then there exists $P_1, \dots, P_m$ such that $L = \bigcup_{i=1}^{m} \mathrm{LinSet}(b \mid P_i)$ where $m \leq 2^k$ and $P_i \subseteq P$ and $\texttt{card}(P_i) \leq d \log_2(2\mu k + 1)$.

## 2.1.1 Presburger formula

The Presburger arithmetic have been introduced in [Pre29] where the author proves the decidability of this theory using quantifier removing method. Originally, Presburger formula were defined as first order theory over naturals augmented with the ordering predicate $\leq$ and the sum function $+$. Here, for simplification of proofs, variables are interpreted over integers and we only consider the existential fragment. Both of those restrictions are w.l.o.g. since integers can be encoded with naturals and universal quantifiers can be removed as in [Pre29].

**Definition – Existential Presburger formula**

An *existential Presburger* formula is built over terms on the signature $\{0, 1, +\} \cup X$ where $X$ is a countable set of variables that take value in $\mathbb{Z}$. So, it is a term generated by the following grammar:

$$\Phi ::= t \leq t \mid \exists x\ \Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi$$

where the variable $x \in X$ is quantified at most once[II]. A formula is said to be *weak* if the predicate $\neq$ is used instead of $\leq$:

$$\Phi ::= t \neq t \mid \exists x\ \Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi$$

For readability, we introduce the common macros. First, we extend the definition of existential Presburger formula with syntactic sugar by the strict order predicate $(t_1 < t_2) \equiv (t_1 + 1 \leq t_2)$, the disequality predicate $(t_1 \neq t_2) \equiv (t_1 < t_2 \vee t_2 < t_1)$, the difference of terms $(t_1 \leq t_2 - t) \equiv (t_1 + t \leq t_2)$ and the fixed multiplication of terms $(t_1 \leq t_2 + \times_c(t)) \equiv (\exists x\ t_1 \leq t_2 + \sum_{i=1}^{c} x \wedge x = t)$. Similarly, we extend the weak fragment with the difference of terms $(t_1 \neq t_2 - t) \equiv (t_1 + t \neq t_2)$ and the fixed multiplication of terms $(t_1 \neq t_2 + \times_c(t)) \equiv (\exists x\ t_1 \neq t_2 + \sum_{i=1}^{c} x \wedge x = t)$.

Note that, in opposition to the equality $ct = t + \cdots + t$ which can create an exponential blow-up, our macro for fixed multiplication introduces an extra variable existentially quantified which create only a polynomial blow-up. We can also highlight the equivalence $(t_1 \leq t_2) \equiv (\exists x\ t_1 = t_2 + x \wedge 0 \leq x)$ which shows that the ordering predicate is not necessary when the variables are interpreted over naturals but here, they are interpreted over integers and in particular, the sign predicate cannot be expressed.

Let $\Phi$ be an existential Presburger formula over the set of variables $X$. We define representation of $|\Phi|$ as the number of symbol to write it, i.e. the size of its syntactic tree. We denote by $\mathrm{Free}(\Phi) = \{x_1, \dots, x_d\}$ the linearly ordered set of variables that appear in

[II]This assumption is not necessary but permits to simplify formula rewriting

$\Phi$ but are not quantified. We call $d = \mathtt{card}(\mathrm{Free}(\Phi))$ the *dimension* of $\Phi$. A *valuation* $\nu$ is a function from $X$ to $\mathbb{Z}$ which can be inductively extended to terms by $\nu(0) = 0$, $\nu(1) = 1$ and $\nu(t_1 + t_2) = \nu(t_1) + \nu(t_2)$. The formula $\Phi$ defines a set of integer vectors denoted $[\![\Phi]\!]$ such that:

$$[\![t_1 \leq t_2]\!] = \{\nu : \nu(t_1) \leq \nu(t_2)\} \qquad [\![\Phi_1 \wedge \Phi_2]\!] = [\![\Phi_1]\!] \cap [\![\Phi_2]\!]$$
$$[\![t_1 \neq t_2]\!] = \{\nu : \nu(t_1) \neq \nu(t_2)\} \qquad [\![\Phi_1 \vee \Phi_2]\!] = [\![\Phi_1]\!] \cup [\![\Phi_2]\!]$$
$$[\![\exists x_i\ \Psi]\!] = [\![\Psi]\!]$$

A formula $\Phi$ is said to be *satisfiable* if $[\![\Phi]\!] \neq \varnothing$. Given $v \in \mathbb{Z}^d$, we define $\Phi(v)$ as the formula $\Phi$ where the free variable have been instantiated by $v$. We write $v \models \Phi$ instead of $[\![\Phi(v)]\!] \neq \varnothing$ i.e. to express that $\Phi$ holds when the vector of free variables is interpreted with $v$. Scarpellini shows in Theorem 6.A from [Sca84] that satisfiability requires to interpret variables with values at most exponential in $|\Phi|$.

**Fact 2.1.3 – Scarpellini's Theorem from [Sca84]**

Let $\Phi$ be an existential Presburger of dimension $d$. If $[\![\Phi]\!] \neq \varnothing$ then there exists $v \in \mathbb{Z}^d$ such that $v \models \Phi$ and $||v||$ is at most exponential in $|\Phi|$.

### 2.1.2 Parikh image of regular languages

Semi-linear sets were originally introduced in [Par66] to provide a characterization to the appearance of letters in words of a given language. This abstraction, called Parikh image, preserves the emptiness of a languages.

**Definition – Parikh image**

Let $\Lambda = \{a_1, \ldots, a_n\}$ be a totally ordered alphabet and $u \in \Lambda^*$. The *Parikh image*[III]of $u$ with respect to $\Lambda$, denoted $\mathfrak{P}_\Lambda(u)$, is defined as the vector $(|u|_{a_1}, \ldots, |u|_{a_n})$. The Parikh image of the language $L \subseteq \Lambda^*$ is $\mathfrak{P}_\Lambda(L) = \{\mathfrak{P}_\Lambda(u) : u \in L\}$.

Any regular language has a semi-linear Parikh image, this result is known as Parikh's Theorem [Par66][IV]. The representation size of the semi-linear set which denotes the Parikh image of the language given by a $\mathsf{NFA}$ has been provided in [Lin10]. The proof of Theorem 7.3.1. of [Lin10] contains many technicalities which raise a long and complex method. The following statement presents a new simple approach but provides a coarser bound.

**Lemma 2.1.4**

Let $A$ be an $\mathsf{NFA}$ over $\Sigma$. The Parikh image over $\Sigma$ of $L(A)$ is semi-linear. It can be represented a semi-linear set $L = \bigcup_{i=1}^m \mathrm{LinSet}\,(b_i \mid P_i)$ where $m$ is doubly exponential[V]in $|A|$ and $||b_i||, \mathtt{card}(P_i), ||P_i||$ are polynomial in $|A|$.

**Proof** Let $A = (Q, Q_I, Q_F, \Delta)$ be an $\mathsf{NFA}$ over $\Sigma$. The proof goes as follow: $(i)$ we provide a semi-linear set $S$ which denotes exactly the Parikh image of $L(A)$, then $(ii)$ we use the Carathéodory property to get the bounds of the statement.
$(i)$ We show here the equality $\mathfrak{P}_\Sigma(L(A)) = S$ with $S$ defined below where for all run $\varrho$ labeled by $u$, we denote by $\mathfrak{P}_\Sigma(\varrho)$ the Parikh image of $u$.

$$S = \bigcup_{\substack{\varrho \in \mathtt{AccRun}(A) \\ |\varrho| \leq \mathtt{card}(\Delta)^2}} \mathrm{LinSet}\left(\mathfrak{P}_\Sigma(\varrho) \mid \left\{\mathfrak{P}_\Sigma(c) : \bigwedge \begin{cases} c \in \mathtt{SimpleCycles}(A) \\ \forall t \in \Delta\ |c|_t > 0 \implies |\varrho|_t > 0 \end{cases}\right\}\right)$$

We prove first that the Parikh image $\mathfrak{P}_\Sigma(L(A))$ belongs to the set $S$. Let us take $v \in S$ and show that there exists a word $u \in L(A)$ such that $\mathfrak{P}_\Sigma(u) = v$. This is trivial since

[III]Also called commutative image in the literature.
[IV]In fact, Parikh's Theorem is more general and proves that context-free languages have a semi-linear Parikh image.
[V]The doubly exponential upper bound for $m$ is not optimal (see [Lin10])

$v$ is generated from an accepting run $\varrho$ of $A$ and the iteration of some cycles that use at least one transition visited by $\varrho$. In fact, any transition of the iterated cycles is visited by $\varrho$.

Now, we prove that the set $S$ belongs to the Parikh image $\mathfrak{P}_\Sigma(L(A))$. Consider $u \in L(A)$. Hence, there exists an accepting run $\varrho$ of $A$ such that $\mathfrak{P}_\Sigma(u) = \mathfrak{P}_\Sigma(\varrho)$. Let $\Delta_\varrho = \{t_1, \dots, t_m\} \subseteq \Delta$ be the transitions visited by $\varrho$ and $C = \mathtt{SimpleCycle}(A) \cap \Delta_\varrho^*$ be the set of simple cycles over transitions used by $\varrho$. Hence $\varrho$ is of the form $\varrho_1 t_1 \dots \varrho_m t_m$. For all $1 \leq i \leq m$, we denote by $n_i$ be the number of, non-necessarily simple, cycles in $\varrho_i$. We also define $n = \sum_{i=1}^m n_i$. By induction on $n$, we show that there exists an accepting run $\hat\varrho$ such that the two following properties hold:

1. $\mathfrak{P}_\Sigma(\varrho) \subseteq \mathfrak{P}_\Sigma(\hat\varrho) + \mathfrak{P}_\Sigma(C)^*$
2. $\mathfrak{P}_\Sigma(\hat\varrho) + \mathfrak{P}_\Sigma(C)^* \subseteq S$

If $n = 0$ we trivially set $\hat\varrho = \varrho$. In fact $|\varrho| \leq \mathtt{card}(\Delta)^2$ since $m \leq \mathtt{card}(\Delta)$ and $|\varrho_i t_i| \leq \mathtt{card}(\Delta)$ due to the absence of cycle in all $\varrho_i$. In addition $C \subseteq \{c \in \mathtt{SimpleCycles}(A) : \forall t \in \Delta \ |c|_t > 0 \implies |\varrho|_t > 0\}$ because for all $t \in \Delta_\varrho$ we have $|\varrho|_t \geq 1$. Thus $\mathfrak{P}_\Sigma(\varrho) + \mathfrak{P}_\Sigma(C)^* \subseteq S$.

Otherwise $n > 0$ i.e. there is a cycle in some $\varrho_i$. Note that, if a run admits a cycle then it admits a simple one. Let $\varrho_i = \varrho_i' c \varrho_i''$ where $c$ is a simple cycle. In fact $c \in C$ since $\varrho \in \Delta_\varrho^*$. We compute $\hat\varrho$ by applying the induction hypothesis on $\varrho_1 t_1 \dots \varrho_i' \varrho_i'' t_i \dots \varrho_m t_m$.

$$\begin{aligned}
\mathfrak{P}_\Sigma(\varrho) &= \mathfrak{P}_\Sigma(\varrho_1 t_1 \dots \varrho_i' \varrho_i'' t_i \dots \varrho_m t_m) + \mathfrak{P}_\Sigma(c) & \\
&\subseteq \mathfrak{P}_\Sigma(\hat\varrho) + \mathfrak{P}_\Sigma(C)^* + \mathfrak{P}_\Sigma(c) & \text{by hypothesis (1)} \\
&\subseteq \mathfrak{P}_\Sigma(\hat\varrho) + \mathfrak{P}_\Sigma(C)^* & \text{since } c \in C \\
&\subseteq S & \text{by hypothesis (2)}
\end{aligned}$$

Finally $\mathfrak{P}_\Sigma(u) \in S$ since $\mathfrak{P}_\Sigma(u) = \mathfrak{P}_\Sigma(\varrho)$. This prove the semi-linearity of the Parikh image of $L(A)$.

*(ii)* In order to get the statement, we apply Claim 2.1.2 on linear sets that define $S$. Let $\ell = \mathtt{card}(\Sigma)$ and $m = \mathtt{card}(\Delta)$. Note that $\mathtt{card}(\mathfrak{P}_\Sigma(\mathtt{SimpleCycles}(A))) \leq (m+1)^\ell$. We obtain an union of $s \leq m^{m^2} 2^{(m+1)^\ell}$ linear sets of the form $L_i = \mathrm{LinSet}\,(b_i \mid P_i)$ where $||b_i|| \leq m^2$ and $||P_i|| \leq m$ and $\mathtt{card}(P_i) \leq \ell \log_2(2m(m+1)^\ell + 1) \leq 2\ell^2 \log_2(2m+3)$. ♦

## 2.2 Parikh automata

Parikh automata, introduced in [KR03], extend finite automata with integer counters that can only be incremented, decremented but never tested for zero. For acceptance, the accumulated values are constrained to belong to a semi-linear set, given by an existential Presburger formula. We also consider a fragment where the formula is weak, yielding the class of weak Parikh automata.

**Definition – Parikh automata**

> A *Parikh automaton* (NPA for short) of dimension $d \in \mathbb{N}$ over some alphabet $\Sigma$ is a tuple $(A, \lambda, \varPhi)$ where $A = (Q, Q_I, Q_F, \Delta)$ is an NFA called the underlying NFA, its transitions are labeled by $\lambda\colon \Delta \to \mathbb{Z}^d$ over the monoid $(\mathbb{Z}^d, +_{\mathbb{Z}^d}, 0_{\mathbb{Z}^d})$ and $\varPhi$ is an existential Presburger formula with $d$ free variables called acceptance constraint.

Let $P = (A, \lambda, \varPhi)$ be an NPA where $A = (Q, Q_I, Q_F, \Delta)$ and $\lambda\colon \Delta \mapsto \mathbb{Z}^d$. The language defined by $P$ is the set of words which admit an accepting run in $A$ associated, by $\lambda$, to a weight vector that satisfies the acceptance constraint $\varPhi$. Formally $L(P) = \{u \in \Sigma^* : \exists v \in \mathbb{Z}^d\ \mathtt{AccRun}_P(u,v) \wedge v \models \varPhi\}$ and $R(P) = \{v \in \mathbb{Z}^d : \exists u \in \Sigma^*\ \mathtt{AccRun}_P(u,v) \wedge v \models \varPhi\}$. Let $\Gamma = \{\lambda(t) \in \mathbb{Z}^d : t \in \Delta\}$ be the set of weight vectors. The representation size of $P$ is defined as $|P| = |A| d \ell \log_2(\mu+1) |\varPhi|$ where $\ell = \mathtt{card}(\Gamma)$ is the number of distinct weight vectors and $\mu = ||\Gamma||$ is the maximal absolute value appearing on weight vectors. If numeric values of weight vectors are given in unary, the parameter $\mu$ becomes fixed and we have that $|P| = |A| d \ell |\varPhi|$. A NPA is said to be *weak* when its acceptance constraint is a weak existential Presburger formula.

### 2.2.1 Intersection and non-emptiness problems

The language of Parikh automata can be abstracted with Parikh image in order to decide the emptiness problem which asks whether the language is empty. The NP membership of the non-emptiness problem is proved, in Proposition III.2 of [FL15]. Given an NPA, the approach consists of constructing in linear time a Presburger formula that denotes the Parikh image of the language of the underlying NFA [VSS05] from which, we are able to represent the set of values that the acceptance constraint can deal with. Finally, the problem reduces to check for the satisfiability of both conditions, being feasible by the underlying NFA and being accepted by the constraint. In addition, we highlight a fragment for which the non-emptiness is NLogSpace.

**Theorem 2.2.1 – NPA non-emptiness**

The non-emptiness problem for NPA is in NP. It is in NLogSpace when the set of weight vectors and the acceptance constraint[VI] are fixed (in particular, for unary encoding of numeric values).

In fact, we are able to obtain a similar statement, but where the formula is logarithmic in the number of transitions. In the following, we use the same proof techniques as used in [FL15].

**Lemma 2.2.2**

Let $P = (A, \lambda, \Phi)$ be a Parikh automaton of dimension $d$, with $m$ transitions and $\ell$ distinct weight vectors which belong to $\{-\mu, \dots, -1, 0, 1, \dots, \mu\}^d$. One can construct an existential Presburger formula $\Psi$ of the form $\bigvee_{i=1}^{s} \Psi_i$ for which there exists $v \in \mathbb{N}$ such that $v \models \Psi$ iff there exists $u \in L(P)$ with $|u| = v$. Furthermore $|\Psi_i|$ is polynomial in $\ell, d, |\Phi|$ and logarithmic in $m, \mu$ for all $1 \le i \le s$.

**Proof** Let $P = (A, \lambda, \Phi)$ be a NPA of dimension $d$ where $A = (Q, Q_I, Q_F, \Delta)$. We define the data set $D = \{a_1, \dots, a_\ell\}$ as the linearly ordered set of weight vectors of $P$ i.e. $D = \{\lambda(t) \in \mathbb{Z}^d : t \in \Delta\}$. This proof highlights an existential Presburger formula $\Psi(x)$ which holds iff $L(P) \neq \varnothing$. The construction relies on $\mathfrak{P}_D(R(P)) \subseteq \mathbb{N}^\ell$ the Parikh image of $R(P)$ with respect to $D$. We recall that $\tau \in \mathfrak{P}_D(R(P))$ iff there exists an accepting run $\varrho$ of $P$ which visits exactly $\tau[i]$ transitions weighted by $a_i$ for all $1 \le i \le \ell$. Intuitively, the language non-emptiness $L(P) \neq \varnothing$ comes as a consequence of the range non-emptiness $R(P) \neq \varnothing$ and the fact that Presburger arithmetic permits to recover the weighted vector of an accepting run from $\tau$. In the sequel, $(i)$ we describe how to construct an existential Presburger formula $\Upsilon$ which defines $\mathfrak{P}_\Gamma(R(P))$ and $(ii)$ from $\Upsilon$ we define the existential Presburger formula $\Psi(x)$ which holds for $v \in \mathbb{N}$ iff there exists an accepting run of $P$ of length $v$.
$(i)$ By Lemma 2.1.4 there exist $s \in \mathbb{N}$ linear sets $L_i = \mathrm{LinSet}(b_i \mid P_i) \subseteq \mathbb{Z}^\ell$ such that $\mathfrak{P}_\Gamma(R(P)) = \bigcup_{i=1}^{s} L_i$. Each set $L_i$ can be represented by the following existential Presburger formula:

$$\Upsilon_i(\tau) = (\exists x_p)_{p \in P_i} \left[ \tau = b_i + \sum_{p \in P_i} \times_p(x_p) \right]$$

Note that $b_i$ and $P_i$ depends on $P$ only. Also Lemma 2.1.4 ensures that for all $1 \le i \le s$ we have $||b_i|| \le m^2$ and $||P_i|| \le m$ and $\mathtt{card}(P_i) \le 2\ell^2 \log_2(2m+3)$. Thanks to the operator twice function $\times_2$, constants can be encoded in binary. So $|\Upsilon_i| = O(4\ell\mathtt{card}(P_i))$ is polynomial in $\ell$ and logarithmic in $m$. Let $\Upsilon(\tau) = \bigvee_{i=1}^{s} \Upsilon_i(\tau)$ then for all $v \in \mathbb{N}^\ell$ we have that $v \models \Upsilon$ iff $v \in \mathfrak{P}_\Gamma(R(P))$.
$(ii)$ Now, we explain how $\Upsilon$ and the acceptance constraint of $P$ are glued together. Recall that $\Gamma = \{a_1, \dots, a_\ell\}$ where each $a_i \in \mathbb{Z}^d$ is a weight vectors of $P$. The value of each dimension $1 \le k \le d$ at the end of a run can be computed from $\tau \in \mathbb{N}^\ell$ by $c_k = \sum_{j=1}^{\ell} \tau[j] \times \mathtt{proj}_k(a_j)$ and the total number of transitions taken is $x = \sum_{j=1}^{\ell} \tau[j]$. Then, we define the formula $\Psi_i$ as follows:

$$\Psi_i(x) = \exists \tau, \exists c \left[ \bigwedge \begin{cases} \Upsilon_i(\tau) \wedge \Phi(c) \wedge x = \sum_{j=1}^{\ell} \tau[j] \\ \bigwedge_{k=1}^{d} c[k] = \sum_{j=1}^{\ell} \times_{a_j[k]}(\tau[j]) \end{cases} \right]$$

[VI] A fixed acceptance constraint implies by definition a fixed dimension.

We have that $|\Psi_i| = O(|\Upsilon_i| + |\Phi| + \ell + d\ell\log_2(||\Gamma||))$. Note that $\mu = ||\Gamma||$ is the maximal absolute value appearing on weight vectors of $P$, it can be encoded in binary thanks to the twice function $\times_2$. Thus $|\Psi_i|$ is polynomial in $\ell, d, |\Phi|$ and logarithmic in $m, \mu$. To conclude $\Psi(x) = \bigvee_{i=1}^{s} \Psi_i(x)$. ♦

From the previous lemma, we are able to provide an upper bound on the complexity to decide the language non-emptiness of the intersection of $n$ NPA. It was shown to be PSpace-C in [FL15] but here we have a more precise statement used in the sequel.

**Lemma 2.2.3**

Let $P_1, \ldots, P_k$ be $k$ NPA over $\Sigma$ each of dimension $d$ with at most $m$ transitions, at most $\ell$ weight vectors that belong to $\{-\mu, \ldots, -1, 0, 1, \ldots, \mu\}^d$ and with an acceptance constraint of size at most $c$. The non-emptiness of $\bigcap_{i=1}^{k} L(P_i)$ can be decided within space polynomial in $k, \ell, d, c$ and logarithmic in $m, \mu$.

**Proof** For all $1 \leq i \leq k$, let $P_i = (A_i, \Phi_i)$ be a Parikh automaton of dimension $d$ over $\Sigma$ with at most $m$ transitions, where $A_i = (Q_i, I_i, F_i, \Delta_i)$ is labeled by $\lambda_i$, admits at most $\ell$ weight vectors belonging to $\{-\mu, \ldots, -1, 0, 1, \ldots, \mu\}^d$ and $|\Phi_i| \leq c$.
We construct $P$ an NPA such that $L(P) = \varnothing$ iff $\bigcap_{i=1}^{k} L(P_i) \neq \varnothing$ which simulates each $P_i$ by performing their transitions sequentially to keep a set of weight vectors of polynomial size. Note that, due to this sequentiality $P$ is not build in the way to have $\bigcap_{i=1}^{k} L(P_i)$ as language, only the non-emptiness matter here. Formally, its set of states is $Q = \{1, \ldots, k\} \times Q_1 \times \cdots \times Q_k$, its set of initial states is $I = \{1\} \times I_1 \times \cdots \times I_k$, its set of final states is $F = \{1\} \times F_1 \times \cdots \times F_k$. Given $a \in \Sigma$, consider for each $A_i$ a transition $q_i \xrightarrow{a} q_i' \in \Delta_i$ with $v_i = \lambda_i(p_i, a, q_i) \in \mathbb{Z}^d$. The transition relation $\Delta$ of $P$ contains $t_1 = (1, q_1, \ldots, q_k) \to (2, q_1', q_2, \ldots, q_k)$ and $\ldots$ and $t_k = (k, q_1', \ldots, q_{k-1}', q_k) \to (1, q_1', \ldots, q_k')$ with $\lambda(t_i) = \{0\}^{d\times(i-1)} \times \{v_i\} \times \{0\}^{d\times(k-i)}$. The input letter on transitions of $P$ does not matter here. Finally, its acceptance constraint $\Phi$ is defined by $\Phi(x_1, \ldots, x_k) = \bigwedge_{i=1}^{k} \Phi_i(x_i)$. Note that $P$ has a dimension $kd$, it admits at most $km^3$ transitions, at most $k\ell$ distinct weight vectors which all belong to $\{-\mu, \ldots, -1, 0, 1, \ldots, \mu\}^{kd}$ and $|\Phi| \leq ck$.
In order to determine whether $L(P) = \varnothing$, we consider the Presburger formula $\Psi$ using Lemma 2.2.2 on $P$. We get $\Psi(x) = \bigvee_{i=1}^{s} \Psi_i(x)$ where each $|\Psi_i|$ is also is polynomial in $k, \ell, d, c$ and logarithmic in $m, \mu$. Recall that for all $v \in \mathbb{N}$, there exists $u \in L(P)$ with $|u| = v$ iff $v \models \Psi$. By Fact 2.1.3, there exists a bound $n$ exponential in $|\Psi_i|$ such that $\Psi_i$ is satisfiable iff it is satisfiable for some value in $\{0, \ldots, n\} \subset \mathbb{N}$. Note that $n$ is exponential in $k, \ell, d, c$ and polynomial in $m, \mu$. Hence, the language $L(P)$ is non-empty iff it accepts a witness word of length at most $n$.
Now we describe an algorithm which decides the emptiness of $L(P)$ using a working space logarithmic in $n$. The algorithm does not construct explicitly $P$ but guesses non-deterministically on-the-fly a witness of length at most $n$ and controls its length using a binary counter. The end of the run is also non-deterministically guessed and its acceptance constraint holds if $\Phi$ (explicitly constructed) is satisfied by the accumulated weights. We show that, given $v = (v_1, \ldots, v_{dk})$ such that $v_i \leq \mu n$, deciding whether $v \models \Phi$ can be done using a space logarithmic in $n$. Let $\Phi'(x_1, \ldots, x_{dk}) = \Phi(x_1, \ldots, x_k) \wedge \bigwedge_{i=1}^{k} v_i = x_i$. The satisfiability of $\Phi'$ in NP thanks to [Sca84]. Since $|\Phi'| \leq |\Phi| + k + \sum_{i=1}^{k} \log(\mu n)$ due to binary encoding of numbers in the formula thanks to the twice function $\times_2$, we get an algorithm logarithmic in $n$. ♦

From Lemma 2.2.3, we get the NLogSpace membership for the restricted non-emptiness of NPA when the set of weight vectors and the acceptance constraint as stated by Theorem 2.2.1. The following result, already given by Proposition III.3 of [FL15], also comes as a direct consequence.

**Corollary 2.2.4 – NPA intersection problem**

Given $k$ NPA $P_1, \ldots, P_k$, deciding whether $\bigcap_{i=1}^{k} L(P_i) = \varnothing$ is in PSpace.

A reader familiar with results on the Presburger logic could see that requiring a constant acceptance constraint in Theorem 2.2.1 is not a necessary condition. In fact, Scarpellini shows in Theorem 6.B of [Sca84] that satisfiability of existential Presburger formula with a fixed number of quantifiers (and thus without succinct constant multiplication $\times_c$) can be decided in NLogSpace. We conjecture that "non-emptiness problem for NPA where number of used variables and set of weight vectors are fixed is in NLogSpace".

In the next subsection, we show that the weakness restriction of NPA on the acceptance constraint permits to relax the condition on the set of weight vectors while keeping an NLogSpace complexity for the non-emptiness.

### 2.2.2 Weak Parikh automata

The emptiness problem for Parikh automata have been previously treated. In particular, Theorem 2.2.1 provides an NLogSpace algorithm for deciding the non-emptiness problem for automata with a constant set of weight vectors and a constant acceptance constraint. In fact, to obtain NLogSpace the assumption on the set of weight vectors is not necessary in the case of weak Parikh automata. Here, we provide another NLogSpace algorithm for non-emptiness of weak Parikh automata (over an arbitrary set of weight vectors) with a constant acceptance constraint.

The restriction to use only predicates $\neq$ in the acceptance constraint is major in the way a run can be evaluated. Intuitively, checking the distinctness of two integers is easier than checking the ordering in the sense that it does not require the entire knowledge of the values. This fact is formalized by the following lemma.

**Lemma 2.2.5**

Let $N \in \mathbb{N}$. For all $x, y \in \mathbb{N}$ such that $-N \leq x, y \leq N$ we have $x \neq y$ iff there exists $0 \leq z \leq \big(2\ln(2N)\big)^2$ such that $x \not\equiv y \mod z$.

**Proof** In this proof we denote by $p_n$ the $n$th prime number. Note that $x \neq y$ iff $x + N \neq y + N$ and $0 \leq x + N, y + N \leq 2N$. Since $p_1 \times \cdots \times p_{\log_2(2N)} \geq 2N$, the Chinese Remainder Theorem states that any integers $x \in \{0, \ldots, 2N\}$ is uniquely determined by the tuple $(x_1, \ldots, x_{\log_2(2N)})$ where $x \equiv x_i \mod p_i$ for each $i$. Similarly, with $y$ and $(y_1, \ldots, y_{\log_2(2N)})$ such that $y \equiv y_i \mod p_i$ for each $i$. So, $x \neq y$ iff there exists $\ell \leq \log_2(2N)$ such that $x_\ell \not\equiv y_\ell \mod p_\ell$. Thanks to Rosser's theorem [Ros39], $p_\ell < \ell\ln(\ell) + 2\ell\ln\ln(\ell)$ for $\ell > 3$. To conclude, $p_\ell \leq \big(2\ln(2N)\big)^2$. ♦

By the use of Lemma 2.2.5 and the Chinese Remainder Theorem, we are now able to check the acceptance constraint of weak NPA by first guessing for all literals which remainder will differ.

**Theorem 2.2.6 – weak NPA non-emptiness**

The non-emptiness problem for weak NPA is in NP. It is in NLogSpace when the acceptance constraint[VII]is constant.

**Proof** Let $P = (A, \Phi)$ be a weak Parikh automaton of constant dimension $d$ where $A = (Q, I, F, \Delta, \lambda)$ have weight vectors in $\{-\mu, \ldots, 0, \ldots, \mu\}^d \subseteq \mathbb{Z}^d$ and $\Phi$ is an acceptance constraint of constant size. We provide an algorithm which decides whether $L(P) \neq \varnothing$ holds in NLogSpace. To do so, $(i)$ we first make some assumptions on $P$ to simplify the proof and the notations. Then $(ii)$ we show a small witness property on $P$ which states that the non-emptiness of $L(P)$ implies the existence of an accepting run with a polynomial length and which satisfies the acceptance constraint. Finally $(iii)$ we describe a non-deterministic procedure that uses a logarithmic space.

$(i)$ In this proof, we assume that the acceptance constraint $\Phi$ is of the form $\bigwedge_{i=1}^{d} x_i \neq 0$ where $d$ is the dimension of $P$ (it is constant as $\Phi$ has constant size at least $d$). We prove now that this assumption is without loss of generality. Since the acceptance constraint $\Phi$ is fixed, it can be transformed in disjunctive normal form and its quantifiers can be removed[VIII]in constant space. In addition, the treated disjunct can be non-deterministically guessed at the beginning for the NLogSpace procedure. So, the acceptance constraint can be transformed into the form $\bigwedge_{i=1}^{k} \alpha_{i,1}x_1 + \cdots + \alpha_{i,d}x_d \neq 0$ where $k$ and all $\alpha_{i,j}$ are constant integers. To obtain the desired form, we need to change the weight vectors of $P$. This modification of the labeling does not change the language

[VII] A constant acceptance constant implies a constant dimension.

[VIII] An algorithm for the quantifier removal of Presburger formula is given in [Coo72] for instance.

of $P$ nor the length of the minimal accepting run. Given a transition $t \in \Delta$ such that $\lambda(t) = (v_1, \ldots, v_d)$, we define $\lambda'(t) = (v'_1, \ldots, v'_k)$ where $v'_i = \alpha_{i,1}v_1 + \cdots + \alpha_{i,d}v_d$. For all $1 \leq i \leq k$ the function $f_i\colon (x_1, \ldots, x_d) \mapsto \alpha_{i,1}x_1 + \cdots + \alpha_{i,d}x_d$ has constant size since the acceptance constraint is fixed. However, applying $f_i$ on any tuple $(v_1, \ldots, v_d)$ of the transitions of $P$ requires polynomial space. We explain in (*iii*) how it can be partially computed to stay within logarithmic space.

(*ii*) In this paragraph, we show that if $P$ (assumed to have an acceptance constraint of the later form) admits an accepting run which satisfies the acceptance constraint, then it admits such a run of polynomial length. Let $\varrho$ be an accepting run of $P$ such that $\mathrm{out}(\varrho) \models \Phi$. Consider $\Delta_\varrho \subseteq \Delta$ be the set of transitions visited by $\varrho$ and $C = \mathrm{SimpleCycles}(A) \cap \Delta_\varrho^*$ be the set of simple cycles over transitions used by $\varrho$.The run $\varrho$ can be decomposed into $t_1\varrho_1 \ldots t_m\varrho_m$ where $\{t_1, \ldots, t_m\} = \Delta_\varrho$ and $\varrho_i$ are intermediate runs. Such decomposition ensures the possibility to attach any cycle of $C$ to $\varrho$. We then construct a shorter run from $\varrho$ by iteratively removing simple cycles of all $\varrho_i$. Formally, from any run $\tau$ we define the sequence $\tau_0 = \tau$ and for all $i > 0$, $\tau_i$ is equal to $\tau_{i-1}$ in which the first visited simple cycle is removed (the choice of the removed cycle is not important but we set it to be the first to make this choice canonical). We let $\hat{\tau}$ the limit of this sequence (which is reached in at most $|\tau|$ iterations). We now let $\hat{\varrho} = t_1\hat{\varrho}_1 \ldots t_m\hat{\varrho}_m$.
By construction, $\hat{\varrho}$ have a length at most $\mathtt{card}(\Delta)^2$ and it is accepting since $\varrho$ is so. However, $\hat{\varrho}$ may not satisfy the acceptance constraint $\Phi$, i.e. , $\mathtt{proj}_i(\hat{\varrho}) = 0$ for some $i$, where $\mathtt{proj}_i$ denotes the projection on the $i$th component. We claim that the satisfaction of $\Phi$ can be restored by attaching $d$ simple cycles of $C$ to $\hat{\varrho}$.

- For all $1 \leq i \leq d$ for which $\mathtt{proj}_i(\mathrm{out}(\hat{\varrho})) = 0$, there exists a cycle denoted $c_i \in C$ with $\mathtt{proj}_i(\mathrm{out}(c_i)) \neq 0$. Indeed, if $\mathtt{proj}_i(\mathrm{out}(c)) = 0$ for all $c \in C$ then in particular the cycles removed from $\varrho$ to obtain $\hat{\varrho}$ have all value zero on the $i$th component. As a direct consequence $\mathtt{proj}_i(\hat{\varrho}) = \mathtt{proj}_i(\varrho) = 0$ implying that $\mathrm{out}(\varrho) \not\models \Phi$ which contradicts the assumptions that $\varrho \models \Phi$. Thus $c_i$ exists.
- For all $1 \leq i \leq d$ for which $\mathtt{proj}_i(\mathrm{out}(\hat{\varrho})) \neq 0$, we pick some cycle $c_i \in C$ arbitrarily.

So, we get $d$ simple cycles $c_1, \ldots, c_d \in C$ that not necessarily appear in $\varrho$ but can be attached to $\hat{\varrho}$ and iterated. For instance, if $c_1$ start with some transition $t_j$, say $c_1 = t_j t'_1 \ldots t'_k t_j$, then it can be iterated in $\hat{\varrho}$, i.e. for all $\ell \in \mathbb{N}$ the sequence of transition $t_1\hat{\varrho}_1 \ldots \hat{\varrho}_{j-1}(c_j)^k\hat{\varrho}_j \ldots t_m\hat{\varrho}_m$ is a accepting run of $P$.
Now, we have to determine how many times each cycle have to be iterated in order to satisfy $\Phi$. This is done by solving the system of disequations $\bigwedge_{j=1}^{d} \mathtt{proj}_j(\mathrm{out}(\hat{\varrho}) + y_1\mathrm{out}(c_1) + \cdots + y_d\mathrm{out}(c_d)) \neq 0$. Using a pigeon hole argument, we can prove that a solution exists by interpreting all $y_i$ in $\{0, \ldots, d\}$. Formally, we show by induction on $0 \leq j \leq d$ how to determine the value $\beta_j \in \{0, \ldots, d\}$ of the variable $y_j$ such that the following holds.

1. $\mathtt{proj}_{j'}(\mathrm{out}(\hat{\varrho}) + \sum_{\ell=1}^{j} \beta_\ell \mathrm{out}(c_\ell)) \neq 0$ for all $1 \leq j' \leq j$
2. $\mathtt{proj}_i(\mathrm{out}(\hat{\varrho})) \neq 0 \Rightarrow \mathtt{proj}_i(\mathrm{out}(\hat{\varrho}) + \sum_{\ell=1}^{j} \beta_\ell \mathrm{out}(c_\ell)) \neq 0$ for all $1 \leq i \leq d$

For $j = 0$ both properties hold trivially. Assume by induction hypothesis that (1) and (2) hold for some $0 \leq j < d$. If $\mathtt{proj}_{j+1}(\mathrm{out}(\hat{\varrho})) \neq 0$ then (1) and (2) trivially extends to $j + 1$ by taking $\beta_{j+1} = 0$. Otherwise, let $v = \mathrm{out}(\hat{\varrho}) + \sum_{\ell=1}^{j} \beta_\ell \mathrm{out}(c_\ell) \in \mathbb{Z}^d$.

- For each $1 \leq i \leq d$ such that $\mathtt{proj}_i(\mathrm{out}(c_{j+1})) \neq 0$ the equation $\mathtt{proj}_i(v + y\mathrm{out}(c_{j+1})) = 0$ admits a unique non-negative solution for $y$. We consider at most $d$ equations so, there exists $\beta_{j+1} \in \{0, \ldots, d\}$ that dissatisfies them all.
- For each $1 \leq i \leq d$ such that $\mathtt{proj}_i(\mathrm{out}(c_{j+1})) = 0$ we have that $\mathtt{proj}_i(v + \beta_{j+1}\mathrm{out}(c_{j+1})) = \mathtt{proj}_i(v)$ and then (1) and (2) trivially extends to $j + 1$.

This construction ensures the existence of an accepting run that satisfies $\Phi$ and have a length bounded by $N = (2d\mathtt{card}(\Delta))^2$ since $|\hat{\varrho}| \leq \mathtt{card}(\Delta)^2$, $|c_i| \leq \mathtt{card}(\Delta)$ and $\beta_i \leq d$ for all $1 \leq i \leq d$.

(*iii*) As for Theorem 2.2.1, we provide an algorithm that guesses a witness in $L(P)$ of length at most $N$. Since $N$ is polynomial in $|P|$, the length can be memorized in binary using a logarithmic space. However, the value of accumulated weights along such witness grows with the maximal absolute value $\mu$ and thus can be $\mu N$, which is exponential in $|P|$. Thanks to Lemma 2.2.5, for all $1 \leq i \leq d$ the constraint $x_i \neq 0$ holds iff there exists $0 \leq r_i \leq \big(2\ln(2\mu N)\big)^2$ such that $x_i \not\equiv 0 \mod r_i$. Using a binary encoding, the numerical value $\big(2\ln(2\mu N)\big)^2$ requires $2\log_2\big(2\ln(2\mu N)\big)$ bits which is logarithmic in $|P|$. Hence our NLogSpace algorithm has $d$ counters, and starts by guessing each rest $r_i$. Then it searches a witness non-deterministically on-the-fly by controlling is length with a binary counter, performing transitions on-demand and updating each counter values modulo the corresponding rest $r_i$. More precisely, each function $f_i$ defined in paragraph $(i)$ can be partially computed by taking its arguments modulo $r_i$ and returning its result modulo $r_i$ as well. Finally, for acceptance, it checks the satisfiability of $\Phi$ within logarithmic space. ♦

In the previous subsection, we made the conjecture that only the number of used variables in the acceptance constraint and vector weights need to be fixed to obtain NLogSpace membership for the non-emptiness of NPA. This argument does not seem to be sufficient in the case for weak NPA with unbounded vector weights. Formally, we conjecture that "non-emptiness problem for weak NPA with number of used variables fixed is hard for PTime".

## 2.3 Future and related works

We investigated complexity results of the Parikh automata formalism, which is, an extension of regular automata which semi-linearly constrains the number of times transitions occur.

**Future works**

However, we have not investigated the complexity of inclusion problem for Parikh automata. Klaedtke and Rueß shown that the universality is undecidable implying the same results for equivalence and inclusion [KR03]. As proved by Cadhilac et al. in [CFM13], the unambiguous fragment of Parikh automata has decidable inclusion problem. It is known that the class of $k$-valued sum-automata, which associate at most $k$ distinct values to each input word, coincide with the class of $k$-unambiguous one, which have at most $k$ accepting run for each input word [SdS10]. As a matter of fact, for a fixed $k \in \mathbb{N}$, we conjecture that the language inclusion problem for Parikh automata represented by $k$-valued sum-automata with existential Presburger formula as acceptance constraint is PSpace-C.

**Related works**

We considered a fragment of Parikh automata where the acceptance constraint is restricted to weak existential Presburger formula, or equivalently $\exists\mathsf{FO}(\mathbb{Z}, \neq, +)$. Disequality tests have been considered in the context of vector addition systems with states (VASS) where the authors investigate the one-counter case [ACP$^+$19]. Note that, counter values of VASS range over $\mathbb{N}$. Now, consider the (unary) $\mathsf{neg}$ predicate defined for all $v \in \mathbb{Z}$ such that $\mathsf{neg}(v)$ holds iff $v < 0$. Since our setting, $\exists\mathsf{FO}(\mathbb{Z}, \neq, +)$, is not expressive enough to define the predicate $\mathsf{neg}$, the two formalisms are incomparable.

We recall that, the emptiness problem of Parikh automata with acceptance constraint in $\exists\mathsf{FO}(\mathbb{Z}, \leq, +)$ i.e. existential Presburger formula is in NP and it is in NLogSpace if the acceptance constraint is in $\exists\mathsf{FO}(\mathbb{Z}, \neq, +)$ i.e. weak existential Presburger formula. The study of the emptiness problem of Parikh automata with acceptance constraint in $\exists\mathsf{FO}(\mathbb{Z}, \neq, +, \mathsf{neg})$ may rely on proof techniques from [ACP$^+$19].

Extensions of Parikh automata have been studied in the literature. On one hand, the model of the underlying automata can be generalized. For instance, affine Parikh automata [CFM12] generalize computation of the values of runs. Instead of taking an automaton weighted by a vector of integers, authors considered an automaton weighted by linear functions. So, an update does not add a constant vector but applies a linear transformation. The downside is that this extension turns out to be quickly intractable since emptiness is undecidable for deterministic affine Parikh automata. The one-counter fragment seems to recover the decidability for emptiness, in fact, even when the updates are computed by a polynomial [FGH13]. Another extension that generalizes the model of the underlying automata is two-wayness, i.e. regular automata that are also allowed to read the input word backward. In [FGM19] we prove that two-way Parikh automata are undecidable for the emptiness problem, then we consider several sufficient conditions under which decidability is recovered and we investigate how they change the complexity. On the other hand, systems of quadratic Diophantine equation are decidable for satisfiability [GS81] and generalize the model from semi-linear acceptance constraint to non-linear one. This result has been used in automata theory to show decidability results of ratio-automata, i.e. two counters sum-automata that aggregate its values with a Euclidean division [FGR15].

# Chapter 3

# Weighted expressions with semi-linear combinators

A popular formalism to define quantitative languages over integers is that of weighted automata over the tropical semi-ring $(\mathbb{Z}, \max, +)$ as defined in Section 1.4, called $\mathsf{WA}^{\max}_{\text{sum}}$ here after. We recall that, given an input word $u$, the image of $u$ by a $\mathsf{WA}^{\max}_{\text{sum}}$ is the maximal value amongst all sums of weights from accepting runs on $u$. In particular, the semantics of sum-automata and $\mathsf{WA}^{\max}_{\text{sum}}$ coincide for their unambiguous fragment i.e. for automata which admit at most one accepting run on any input word. However, $\mathsf{WA}^{\max}_{\text{sum}}$ have undecidable[IX] inclusion, equivalence and universality problems [Kro94], even if they are linearly ambiguous i.e. for automata which have at most a linear number of accepting runs in the length of its input word [DGM17]. The most expressive known class of $\mathsf{WA}^{\max}_{\text{sum}}$ enjoying decidability for inclusion is that of finitely ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$ [FGR14]. Moreover, $\mathsf{WA}^{\max}_{\text{sum}}$ are not closed under simple operations such as minimum or minus [KLMP04]. In particular, basic functions such as $u \mapsto \min(|u|_a, |u|_b)$ and as a consequence $u \mapsto \text{abs}(f(u) - g(u))$ are not definable by $\mathsf{WA}^{\max}_{\text{sum}}$, even if $f, g$ are.

To cope with the expressiveness and undecidability issues, we introduce new weighted formalisms which retain decidability for inclusion while being strictly more expressive than finitely ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$. The previous chapter presented the model of Parikh automata that enrich the acceptance of a regular language with a Presburger constraint. Now, we use the expressiveness of Presburger arithmetic to attach a mechanism that combines outputs computed by a regular language in order to define quantitative languages over integers.

## 3.1 Weighted expressions without iteration

We start our study of weighted expressions by a definition directly inspired by [CDE+10] where deterministic sum-automata are used as building blocks of quantitative expressions, called mean-payoff expressions[X] that can be inductively composed with functions such as min, max, addition and minus. The universality, emptiness, inclusion and equivalence problems for mean-payoff expressions are PSpace-C [Vel12]. We cast here mean-payoff expressions to finite words, this gives what we call simple expressions. We prove some results for simple expressions that motivate the need to have a more expressive formalism.

**Definition – Simple expressions**

A *simple expression* is a term generated by the following grammar, where $D$ range over deterministic sum-automata.

$$E ::= D \mid \min(E_1, E_2) \mid \max(E_1, E_2) \mid E_1 + E_2 \mid E_1 - E_2$$

Any simple expression $E$ defines a quantitative language $[\![E]\!] \colon \Sigma^* \to \mathbb{Z}$ on a domain $\texttt{dom}(E)$ is inductively defined as: If $E$ is defined as a deterministic sum-automaton $D$ then $\texttt{dom}(E) = \texttt{dom}(D)$ and for all $u \in L(D)$ we have $[\![E]\!](w) = [\![D]\!](w)$, where the semantics of sum-automata has been defined in Section 1.4. Otherwise if $E$ is of the form $\min(E_1, E_2)$ then $\texttt{dom}(E) = \texttt{dom}(E_1) \cap \texttt{dom}(E_2)$ and $[\![E]\!](w) = \min([\![E_1]\!](u), [\![E_2]\!](u))$ holds for all $u \in \texttt{dom}(E)$. The semantics of max, + and − are defined similarly. We say that two simple expressions $E_1, E_2$ are equivalent if $[\![E_1]\!] = [\![E_2]\!]$, in particular $\texttt{dom}(E_1) = \texttt{dom}(E_2)$.

[IX] The quantitative language emptiness problem is decidable for $\mathsf{WA}^{\max}_{\text{sum}}$.

[X] Chatterjee et al. studied quantitative expressions on infinite words and the automata that they consider are deterministic mean-payoff automata

Now, we provide a criterion which shows that taking deterministic sum-automata as atoms for simple expression is strictly less expressive than taking an unambiguous sum-automata. A quantitative language $f$ denotes a *Lipschitz-continuous* function if there exists $k \in \mathbb{N}$ such that for all words $u, v \in \Sigma^*$, $\mathrm{abs}(f(u) - f(v)) \leq k \times (|u| + |v| - 2|u \sqcap v|)$ where $u \sqcap v$ denotes the longest common prefix of $u$ and $v$.

**Proposition 3.1.1**

Any simple expression defines a Lipschitz continuous quantitative language.

**Proof** To prove that simple expressions define Lipschitz continuous functions, we need to show that for all simple expression $E$, there exists $k \in \mathbb{N}$ such that for all words $u, v \in \Sigma^*$:

$$\mathrm{abs}(E(u) - E(v)) \leq k\,(|u| + |v| - |u \sqcap v|) \tag{$\star$}$$

We reason by induction on the structure of the simple expressions.
First, let us consider the base case where $E = D$. As $D$ is deterministic, the partial sum on $u = w \cdot u'$ and $v = w \cdot v'$ on their common prefix $w = u \sqcap v$ is equal in the two cases to some value $s_w$ then on the two different suffixes $u'$ and $v'$, their sum may differ but at most by the following amount: $|u'| \times \mu + |v'| \times \mu$ where $\mu$ is the maximum of the set of absolute value of weights appearing in the automaton $D$. It is clear that the inequality $(\star)$ is true when we take $k = \mu$.
Second, we consider the operation min for the inductive case, i.e. $E = \min(E_1, E_2)$. The other operators are treated similarly. By induction hypothesis, $E_1$ and $E_2$ defines Lipschitz continuous functions, and we note $k_1$ and $k_2$ their respective Lipschitz constants. We claim that $k = \max(k_1, k_2)$ is an adequate constant to show the Lipschitz continuity of $E$ that is:

$$\forall u, v \in \Sigma^* \qquad \mathrm{abs}(\min(E_1, E_2)(u) - \min(E_1, E_2)(v)) \leq k\,(|u| + |v| - |u \sqcap v|)$$

Let $d = (|u| + |v| - |u \sqcap v|)$. Assuming that $\min(E_1, E_2)(u) = E_1(u)$, $\min(E_1, E_2)(v) = E_2(v)$, and that $E_1(u) \geq E_2(v)$. Thus $\mathrm{abs}(E_1(u) - E_2(v)) = E_1(u) - E_2(v)$. Furthermore we have $E_1(u) - E_2(v) \leq E_1(u) - E_1(v) \leq k_1 d \leq \max(k_1, k_2)d$. So finally, $\max(k_1, k_2)d = kd$. All the other cases are treated similarly. ♦

Unambiguous sum-automata can define functions that are not Lipschitz continuous, as for example the function "last block" which maps any word over the form $a^{n_k} b a^{n_{k-1}} b \ldots b a^{n_0}$ to $n_0$. Hence by the previous proposition, this function is not definable by a simple expression. On the other hand, $u \mapsto \min\{|u|_a, |u|_b\}$ is definable by a simple expression while it is not definable by any $\mathsf{WA}^{\max}_{\text{sum}}$ [KLMP04], and then neither by the subclass of unambiguous sum-automata. To summarize:

**Proposition 3.1.2**

There are quantitative languages that are definable by unambiguous sum-automata and not by simple expressions. There are quantitative languages that are definable by simple expressions but not by an unambiguous sum-automata.

## 3.1.1 Monolithic Expressions

To unleash the expressive power of simple expressions, we introduce monolithic expressions as a generalization. First, instead of deterministic sum-automata, we consider unambiguous sum-automata as atoms. This extends their expressiveness beyond finite valued $\mathsf{WA}^{\max}_{\text{sum}}$. Second, instead of considering a fixed (and arbitrary) set of operators, we consider instead any Presburger combinator. We show that all the decision problems are PSPACE-C for monolithic expressions.

Any binary operation $\boxplus \colon \mathbb{Z}^2 \to \mathbb{Z}$ is extended to quantitative languages by $f_1 \boxplus f_2(u) = f_1(u) \boxplus f_2(u)$ if $u \in \mathtt{dom}(f_1) \cap \mathtt{dom}(f_2)$, otherwise it is undefined. Let $\varphi$ be a existential Presburger formula of dimension $d \in \mathbb{N}_{\neq 0}$. We say that $\varphi$ is *functional*[XI] if for all $v_1, \ldots, v_{d-1} \in \mathbb{Z}$, there exists a unique $v_d \in \mathbb{Z}$ such that $v_1, \ldots, v_d \models \varphi$. Hence, $\varphi$ defines a total function from $\mathbb{Z}^{d-1}$ to $\mathbb{Z}$ that we denote $[\![\varphi]\!]$. For convenience, we

[XI] For readability, we use uppercase like $\Phi, \Psi$ for existential Presburger formula and lowercase like $\varphi, \psi$ for function defined by existential Presburger formula call $\alpha = d-1$ the arity of $\varphi$ and write $\varphi(x_1,\dots,x_\alpha)$ to denote the unique $z$ such that $\varphi(x_1,\dots,x_\alpha,z)$ holds. We say that a function $f\colon \mathbb{Z}^\alpha \to \mathbb{Z}$ is a Presburger *combinator* if there exists a functional existential Presburger formula $\varphi$ such that $f = [\![\varphi]\!]$.

**Example 3.1.3**

The following examples are Presburger combinators.

- The operator max which returns the maximum of values $x_1,\dots,x_n \in \mathbb{Z}$ is definable by:

$$\varphi_{\max} \equiv (\bigwedge_{i=1}^{n} x_i \leq z) \wedge (\bigvee_{i=1}^{n} x_i = z)$$

- The absolute value of $x$ is defined by:

$$\varphi_{abs} \equiv (x < 0 \implies z = -x) \wedge (x \geq 0 \implies z = x)$$

- the 1-norm distance $\sum_{i=1}^{n} \mathrm{abs}(x_i - y_i)$ between $(x_1,\dots,x_n)$ and $(y_1,\dots,y_n)$ is defined by:

$$\varphi_1 \equiv \exists z_1 \dots \exists z_n, \bigwedge_{i=1}^{n} \varphi_{abs}(x_i - y_i, z_i) \wedge z = \sum_{i=1}^{n} z_i$$

**Definition – Monolithic expressions**

A *monolithic expression* is a term $E$ generated by the following grammar, where $W$ range over unambiguous sum-automata and $\varphi$ range over Presburger combinators of some arity $n \in \mathbb{N}$.

$$E ::= W \mid \varphi(E_1,\dots,E_n)$$

The semantics $[\![E]\!]\colon \Sigma^* \to \mathbb{Z}$ of a monolithic expression $E$ is defined inductively and similarly as for simple expressions. For $E$ of the form $\varphi(E_1,\dots,E_n)$ we define $\mathtt{dom}(E) = \bigcap_{i=1}^{n} \mathtt{dom}(E_i)$ and $[\![E]\!](u) = [\![\varphi]\!]([\![E_1]\!](u),\dots,[\![E_n]\!](u))$ for all $u \in \mathtt{dom}(E)$. The representation size $|E|$ of a monolithic expression $E$ is inductively defined as: If $E$ is defined as an unambiguous sum-automaton $W$ then $|E| = |W|$ otherwise if $E$ is of the form $\varphi(E_1,\dots,E_n)$ then $|E| = |\varphi| + \sum_{i=1}^{n} |E_i|$.

**Example**

As seen in Example 3.1.3, max is Presburger-definable by a formula $\varphi_{\max}$, it is also the case for $\min(E_1,\dots,E_n)$, $E_1 + E_2$, $E_1 - E_2$ and the unary operation $-E$. For monolithic expressions $E_1, E_2$, the distance $\mathrm{abs}(E_1 - E_2)\colon u \in \mathtt{dom}(E_1) \cap \mathtt{dom}(E_2) \mapsto \mathrm{abs}(E_1(u) - E_2(u))$ is defined by the monolithic expression $\max(E_1 - E_2, E_2 - E_1)$. This function is not definable by a $\mathsf{WA}^{\max}_{\text{sum}}$ even if $E_1, E_2$ are given by unambiguous $\mathsf{WA}^{\max}_{\text{sum}}$. As a consequence of non-expressibility by $\mathsf{WA}^{\max}_{\text{sum}}$ of $\min\{|u|_a, |u|_b\} = \mathrm{abs}(0 - \max\{-|u|_a, -|u|_b\})$ for all $u$ [KLMP04].

**Proposition 3.1.4**

Monolithic expressions are strictly more expressive than finite valued $\mathsf{WA}^{\max}_{\text{sum}}$. There are functions definable by monolithic expressions and not by a $\mathsf{WA}^{\max}_{\text{sum}}$.

**Proof** We show that monolithic expressions can express any quantitative language definable by a $k$-valued $\mathsf{WA}^{\max}_{\text{sum}}$. It is known that any $k$-valued $\mathsf{WA}^{\max}_{\text{sum}}$ $A$ can be decomposed into a disjoint union of $k$ unambiguous sum-automata $A_i$ [FGR14, KLMP04]. It is tempting to think that $A$ is equivalent to the monolithic expression $\max(A_1,\dots,A_k)$. However, this latter expression is defined only on $\bigcap_i \mathtt{dom}(A_i)$, which may be strictly included in $\mathtt{dom}(A)$. Hence, we first complete any automaton $A_i$ into some $B_i$ such that $\mathtt{dom}(B_i) = \mathtt{dom}(A)$ and for all $u \in \mathtt{dom}(B_i) \setminus \mathtt{dom}(A_i)$, $[\![B_i]\!](u) \leq [\![A]\!](u)$. Let $\mu$ be the smallest value occurring on the transitions of all the automata $A_i$. We construct $B_i$ as the disjoint union of $A_i$ and some deterministic sum-automaton $A_i^c$ such that $\mathtt{dom}(A_i^c) = \mathtt{dom}(A) \setminus \mathtt{dom}(A_i)$ and $[\![A_i^c]\!](u) = \mu|u|$. In fact, $A_i^c$ can be easily constructed from any $\mathsf{DFA}$ recognizing $\mathtt{dom}(A) \setminus \mathtt{dom}(A_i)$ and weight function associating $\mu$ to any transitions. Then, $A$ is equivalent to the monolithic expression $\max(B_1, \dots, B_k)$. For the second statement, it is already the case for simple expressions by Proposition 3.1.2. ♦

Our goal is now to show the decidability of the classical decision problems for quantitative languages defined by monolithic expressions. In fact every problem reduces (in PTime) to the $>0$-emptiness problem.

**Proposition 3.1.5**

The quantitative emptiness, universality, equivalence and inclusion decision problems for monolithic expressions reduce to the >0-emptiness.

**Proof** Monolithic expressions contains the constant quantitative languages $c_i\colon u \mapsto v$ for all $v \in \mathbb{Z}$, it is closed under regular domain restriction, minus and enjoy decidable domain inclusion, the emptiness, universality, inclusion and equivalence problems. So, the reduction to the >0-emptiness problem goes as follows:

- to establish $f(w) \geq \nu$ for all $w \in \mathtt{dom}(f)$, it suffices to check that it is not the case that $\exists w \in \mathtt{dom}(f), -(f(w) - \nu) > 0$
- to establish $\mathtt{dom}(f_2) \subseteq \mathtt{dom}(f_1)$ and $f_1(w) \geq f_2(w)$ for all $w \in \mathtt{dom}(f_2)$, when the first assertion succeeds it suffices to check that it is not the case that $\exists w \in \mathtt{dom}(f_2), -(f_1(w) - f_2(w)) > 0$

Finally, the strict inequalities variants reduces to ≥0-emptiness which in turns reduces to >0-emptiness by adding the constant function $c_1$. Note also with similar arguments, we can show that the >0-emptiness problem can be reduced to the universality and inclusion problems. ♦

Thus, we only have to provide a decision procedure for >0-emptiness, and to do so, we first give a normal form on monolithic expressions that allows us to rely on Parikh automata defined in Section 2.2.

**Lemma 3.1.6 – normal form**

From any monolithic expression $E$ whose atoms are unambiguous sum-automata $W_1, \dots, W_n$, one can construct in linear-time an equivalent monolithic expression $\varphi(W_1, \dots, W_n)$, for some Presburger combinator $\varphi$.

**Proof** We construct $E'$ the normal form of $E$ such that $[\![E]\!] = [\![E']\!]$ and $|E'| = O(|E|)$, by structural induction on $E$. If $E$ is defined as an unambiguous sum-automaton $W$ then $E' = \varphi_{\text{id}}(W)$ where $\varphi_{\text{id}}$ denote the identity function. If $E$ is of the form $\psi(E_1, \dots, E_n)$ then for each $i \in \{1, \dots, n\}$ we can construct by induction hypothesis $E'_i = \psi_i(W_{i,1}, \dots, W_{i,m_i})$ equivalent to $E_i$. Finally, we define $E' = \varphi(W_{1,1}, \dots, W_{1,m_1}, \dots, W_{n,1}, \dots, W_{n,m_n})$ where $\varphi$ is constructed as follows:

$$\varphi(x_{1,1}, \dots, x_{1,m_1}, \dots, x_{n,1}, \dots, x_{n,m_n}, r) = \exists r_1, \dots, r_n \quad \bigwedge \begin{cases} \psi(r_1, \dots, r_n, r) \\ \bigwedge_{i=1}^{n} \psi_i(x_{i,1}, \dots, x_{i,m_i}, r_i) \end{cases}$$

♦

**Theorem 3.1.7**

The emptiness, universality, inclusion and equivalence problems for monolithic expressions are PSpace-C.

**Proof** We start by proving the upper bound. By Proposition 3.1.5 and since PSpace = coPSpace we must only show that the quantitative >0-emptiness problem is in PSpace. Let $E$ be a monolithic expression, we can assume w.l.o.g. that $E$ is of the form $\varphi(W_1, \dots, W_n)$ applying Lemma 3.1.6. Then the proof goes as for Lemma 2.2.3 where $W_i$ are seen as UPA with a trivial acceptance constraint. The formula $\Phi$, in the proof of Lemma 2.2.3 is now defined as $\Phi(x_1, \dots, x_k) = \varphi(x_1, \dots, x_k)$. Note that, by taking $c = |\varphi|$ as parameter we have that $|\Phi| \leq ck$.

We now show the lower bound, by reduction from the *finite automaton intersection problem.* Let $A_1, \dots, A_n$ be $n$ deterministic regular automata with a common alphabet

$\Sigma$, determining whether $\{w \in \Sigma^* \mid w \in \bigcap_{i=1}^n L(A_i)\} \neq \varnothing$ is PSpace hard. Moreover, since PSpace = coPSpace we also have that $\{w \in \Sigma^* \mid w \in \bigcap_{i=1}^n L(A_i)\} = \varnothing$ is PSpace hard. So, for each $1 \leq i \leq n$, we construct in linear time the deterministic sum-automaton $W_i$ such that for all $w \in \Sigma^*$ if $w \in L(A_i)$ then $W_i(w) = 1$ otherwise $W_i(w) = 0$.
We show that the finite automaton intersection problem can be reduced to the quantitative $>0$-emptiness problem and its negation can be reduced to the quantitative equivalence problem:

$$\begin{aligned}
\{w \in \Sigma^* \mid w \in \textstyle\bigcap_{i=1}^n L(A_i)\} = \varnothing \quad &\Leftrightarrow \forall w \in \Sigma^*, \exists 1 \leq i \leq n, w \notin L(A_i) \\
&\Leftrightarrow \forall w \in \Sigma^*, \min(W_1(w), \dots, W_n(w)) = 0 \\
\{w \in \Sigma^* \mid w \in \textstyle\bigcap_{i=1}^n L(A_i)\} \neq \varnothing \quad &\Leftrightarrow \exists w \in \Sigma^*, \forall 1 \leq i \leq n, w \in L(A_i) \\
&\Leftrightarrow \exists w \in \Sigma^*, \min(W_1(w), \dots, W_n(w)) > 0
\end{aligned}$$

Finally, the PSpace hardness of the quantitative equivalence implies trivially the PSpace hardness of the quantitative inclusion. ♦

## 3.2 Weighted expressions with iterated sum

As shown by Proposition 3.1.4 monolithic expressions are strictly more expressive than finite valued $\mathsf{WA}^{\max}_{\text{sum}}$. However, in this model, combinator cannot be apply at run time i.e. there is no operation iterable on factors of the input word. For instance, the following quantitative languages are not definable with monolithic expressions.

**Example 3.2.1**

Let $\Sigma = \{a, b, c, d\}$ and $S = \{\circ, \bullet\}$ two separator symbols. Let $f, g, h$ be quantitative languages defined, for all $n \geq 1$ and $u_1, \dots, u_n \in \Sigma^*$, all $s_1, \dots, s_n \in S$ and all $v_1, \dots, v_n \in (\Sigma \cup \{\bullet\})^*$ by:

$$\begin{aligned}
f(u_1 s_1 \dots u_n s_n) &= \textstyle\sum_{i=1}^n \max\{\#_a(u_i), \#_b(u_i)\} \\
g(u_1 s_1 \dots u_n s_n) &= \textstyle\sum_{i=1}^n \max\{\#_c(u_i), \#_d(u_i)\} \\
h(v_1 \circ \dots \circ v_n \circ) &= \textstyle\sum_{i=1}^n \max\{f(v_i \circ), g(v_i \circ)\}
\end{aligned}$$

Any atom of a monolithic expression applies on the whole input word, while here one needs to apply them on factors of the input word. We conjecture that $f$ cannot be defined with a monolithic expression. Note that $h$ needs two levels of iterations, as $f$ and $g$ themselves iterate over factors of the sub-words $v_i$.

Let us now formally define what we mean by iterated-sum. It is an unambiguous quantitative version of the Kleene star, which was already defined in [AFR14].

### Iterated sum

We start by defining the language of uniquely decomposable words with respect to a language $L \subseteq \Sigma^*$. It is denoted by $L^{\oplus} \subseteq L^*$ and define as the set of words $w$ such that there exists $n \geq 1$ and at most one tuple $(w_1, \dots, w_n) \in (L \setminus \{\varepsilon\})^n$ where $w = w_1 \dots w_n$. Conventionally we consider also $L^{\circledast} = L^{\oplus} \cup \{\varepsilon\}$. Given $f \colon \Sigma^* \to \mathbb{Z}$ a quantitative language, the iterated-sum of $f$ denoted by $f^{\circledast}$, is defined by $f^{\circledast}(\varepsilon) = 0$, and for all $w \in \mathtt{dom}(f)^{\oplus}$, by $f^{\circledast}(w) = \sum_{i=1}^n f(w_i)$, where $(w_1, \dots, w_n)$ is the unique decomposition of $w$. Note that $\mathtt{dom}(f^{\circledast}) = \mathtt{dom}(f)^{\circledast}$ for any $f$.

**Proposition 3.2.2**

Given a regular language $L \subseteq \Sigma^*$, its unambiguous iteration $L^{\circledast}$ is regular.

**Proof** Let $A = (Q, q_I, Q_F, \Delta)$ be the DFA recognizing $L \setminus \{\varepsilon\}$. First one can define the non-deterministic automaton $B$ (with $\varepsilon$-transitions) from $A$ as $B = (Q, \{q_I, q_I'\}, Q_F \cup \{q_0'\}, \Delta \cup \{(q_f, \varepsilon, q_I) \mid q_f \in Q_F\})$ which accepts $\varepsilon$ and all words that are decomposable into non-empty factors of $L$. Then, by taking the product of $B$ with itself, and by adding a bit of memory in this product to remember whether some $\varepsilon$-transition was fired in parallel of a non-$\varepsilon$ one (which implies in that case, if the two simulated runs of $B$ terminates, that there are two distinct decompositions), one obtains an automaton which accepts all words which can be non-uniquely decomposed. It suffices then to complement this automaton, concluding the proof. ♦

### 3.2.1 Definition and undecidability of expressions with iterated-sum

We now define the extension of monolithic expressions with iterated-sum and how it leads to undecidability.

**Definition**

A *iterated-sum expression* $E$ is a term generated by the following grammar, where $W$ range over unambiguous sum-automata and $\varphi$ range over Presburger combinators of some arity $n \in \mathbb{N}$.

$$E ::= W \mid \varphi(E_1, \dots, E_n) \mid E^{\circledast}$$

As for monolithic expressions, the semantics of any iterated-sum expression $E$ is a quantitative language $[\![E]\!] \colon \Sigma^* \to \mathbb{Z}$ inductively defined on the structure of the expression, in particular $[\![E^{\circledast}]\!] = [\![E]\!]^{\circledast}$. The domain $\texttt{dom}(E)$ of an iterated-sum expression is $\texttt{dom}([\![E]\!])$. We extend the representation size of a monolithic expression to an iterated-sum expression by $|E^{\circledast}| = |E| + 1$. We also introduce the syntactic sugar $+, -, \max, \min$ which are Presburger-definable, as well as the regular domain restriction[XII] $E|_L$ of a given expression $E$ for any regular language $L$.

**Example 3.2.3**

The quantitative language $f$ of Example 3.2.1 can be defined by the expression $F = \max(W_a, W_b)^{\circledast}$, where $W_\sigma$, for $\sigma \in \{a, b, c, d\}$, is a deterministic 2-state sum-automaton counting the number of occurrences of $\sigma$ in words of the form $us$, $u \in \{a, b, c, d\}^*$ and $s \in S$ (otherwise it is undefined). Similarly, $g$ is defined by $G = \max(W_c, W_d)^{\circledast}$.
The quantitative language $h$ is defined by $H = \max(F|_L, G|_L)^{\circledast}$, where $L$ is defined by the rational expression $(\Sigma^* \bullet)^* \Sigma^* \circ$. Note that in $H$, iterated-sum operators are nested, and it is necessary, since the max operator does not distribute over the sum operator. The example $H$ can be generalized to any nesting level $n$ of iterated-sum operators, by considering $n$ different separators.

As a positive result, we first show that the domain of any iterated-sum expression $E$ is effectively regular.

**Proposition 3.2.4**

The domain of any iterated-sum expression is regular.

**Proof** The domain of a sum-automaton is regular, and defined by its underlying finite automaton. For an expression $\varphi(E_1, \dots, E_n)$, by induction $\texttt{dom}(E_i)$ is regular for all $i$, and by definition, $\texttt{dom}(\varphi(E_1, \dots, E_n)) = \bigcap_i \texttt{dom}(E_i)$ which is regular since regular languages are closed under intersection. Consider now the case of an expression of the form $E^{\circledast}$. By induction hypothesis, $\texttt{dom}(E)$ is regular, and since $\texttt{dom}(E^{\circledast}) = \texttt{dom}(E)^{\circledast}$, by Proposition 3.2.2 we get the result. ♦

By reducing the halting problem for 2-counter machines, it turns out that all the decision problems are undecidable for iterated-sum expressions, even without nesting iteration operators.

**Theorem 3.2.5**

Emptiness, universality, inclusion and equivalence for iterated-sum expressions are undecidable problems, even if only monolithic expressions are iterated.

[XII]The expression $E|_L$ can be effectively constructed by $E + C_L$ where $C_L$ is an unambiguous sum-automaton such that $\texttt{dom}(C_L) = L$ and for all $w \in \texttt{dom}(C_L)$ we have that $[\![C_L]\!](w) = 0$.

**Proof** The proof of this theorem, inspired by the proof of [DGM17] for the undecidability of $\mathsf{WA}^{\max}_{\text{sum}}$ universality, consists of a reduction from the 2-counter machine halting problem to the $\geq 0$-emptiness problem of iterated-sum expressions. In particular, we construct an expression $E$ such that $E(w) \leq 0$ for all $w \in \mathtt{dom}(E)$, and $E(w) = 0$ iff $w$ encodes an halting computation of the counter machine. This establishes undecidability for the other decision problems by Proposition 3.1.5.

*Sketch of proof* We first explain the main ideas of the proof. A configuration of a 2-counter machine is defined as a tuple $(q, \nu)$ where $q$ is a state and $\nu\colon \{x, y\} \to \mathbb{Z}$ a counter valuation. In this reduction, a transition between two successive configurations $...(q_1, (x \mapsto c_1, y \mapsto d_1))\delta(q_2, (x \mapsto c_2, y \mapsto d_2))...$, where $\delta$ is a transition of the machine, is coded by a factor of word of the form: $... \vdash q_1 a^{c_1} b^{d_1} \lhd \delta \rhd q_2 a^{c_2} b^{d_2} \dashv\vdash q_2 a^{c_2} b^{d_2} \lhd ....$

We show that such a word encodes an halting computation if it respects a list of simple requirements that are all are regular but two: one that expresses that increments and decrements of variables are correctly executed, and one that imposes that, from one transition encoding to the next, the current configuration is copied correctly. In our example above, under the hypothesis that $x$ is incremented in $\delta$, this amounts to check that the number of $a$ occurrences before $\delta$ is equal to the number of occurrences of $a$ after $\delta$ minus one. This property can be verified by simple expression on the factor between the $\vdash$ and $\dashv$ that returns 0 if it is the case and a negative value otherwise. The second property amounts to check that the number of occurrences of $a$ between the first $\rhd$ and $\dashv$ and the number of $a$ between the second $\vdash$ and second $\lhd$ are equal. Again, it is easy to see that this can be done with a simple expression that returns 0 if it is the case and a negative value otherwise. Then, with iterated-sum expressions we decompose the word into factors that are between the markers $\vdash$ and $\dashv$, and other factors that are between the markers $\rhd$ and $\lhd$, and we iterate the application of the simple expressions mentioned above. The sum of all the values computed on the factors is equal to 0 if the requirements are met and negative otherwise.

*Formal proof* We now formally define the reduction from the halting problem of 2-counter machines. Let $M = (\Sigma, \{x, y\}, Q, q_{init}, F, \Delta, \tau, \lambda)$ be a deterministic 2-counter machine, with set of states $Q$, initial state $q_{init}$, accepting states $F$, transitions $\Delta : Q \times \Sigma \to Q$, guards $\tau : \Delta \to \{0, \neq 0\}^2$ (tests to 0 for both counters), and updates $\lambda : \Delta \to \{-1, 0, 1\}^2$. Let $\nu_0$ be such that $\nu_0(x) = 0$ and $\nu_0(y) = 0$, and w.l.o.g., let us assume that the states in $F$ are the halting states of $M$. We reduce the problem of deciding if the unique[XIII]computation of $M$ that starts from configuration $(q_{init}, \nu_0)$ reaches or not an accepting state (from which it halts) to the problem of deciding if for some effectively constructible iterated-sum expression $E$, there exists a word $w \in L(E)$ such that $E(w) \geq 0$.

Before defining $E$, we first explain how we encode computations of $M$ into words over the alphabet $\Gamma = Q \cup \{\vdash, \dashv, \rhd, \lhd, a, b\} \cup \Delta$. Let $\varrho =$

$$(q_0, v_0)\delta_0(q_1, v_1)\delta_1 \dots (q_{n-1}, v_{n-1})\delta_n(q_n, v_n)$$

be a computation of $M$, we encode it by the following word over $\Gamma$:

$$\vdash q_0 a^{v_0(x)} b^{v_0(y)} \lhd \delta_0 \rhd q_1 a^{v_1(x)} b^{v_1(y)} \dashv\vdash q_1 a^{v_1(x)} b^{v_1(y)} \lhd \delta_1 \rhd q_2 a^{v_2(x)} b^{v_2(y)} \dashv \dots$$
$$\vdash q_{n-1} a^{v_{n-1}(x)} b^{v_{n-1}(y)} \lhd \delta_{n-1} \rhd q_n a^{v_n(x)} b^{v_n(y)} \dashv$$

So a word $w \in \Gamma^*$ encodes of the halting computation of $M$ from $\nu_0$ if the following conditions holds:

1. the word $w$ must be in the language defined by the following regular expression $(\vdash Qa^*b^* \lhd \Delta \rhd Qa^*b^* \dashv)^*$
2. the first element of $Q$ in $w$ is equal to $q_{init}$, i.e. the computation is starting in the initial state of $M$
3. the last element of $Q$ in $w$ belongs to set $F$, i.e. the computation reaches an accepting state of $M$
4. the first element of $Q$ in $w$ is directly followed by an element in $\Delta$, i.e. the computation starts from the valuation $\nu_0$
5. for each factor of the form $\vdash q_1 a^{n_1} b^{m_1} \lhd \delta \rhd q_2 a^{n_2} b^{m_2} \dashv$:
   (a) $\delta$ is a transition from $q_1$ to $q_2$
   (b) $(n_1, m_1) \models \tau(\delta)$, i.e. the guard of $\delta$ is satisfied
   (c) $((n_1, m_1), (n_2, m_2)) \models \lambda(\delta)$, i.e. the updates of $\delta$ are correctly realized
6. for each factor of the form $\rhd q_1 a^{n_1} b^{m_1} \vdash\dashv q_2 a^{n_2} b^{m_2} \lhd$, it is the case that:

(a) $q_1 = q_2$, i.e. the control state is preserved from one configuration encoding to the next one

(b) $n_1 = n_2$ and $m_1 = m_2$, i.e. valuations of counters are preserved from one configuration encoding to the next one

Let us now explain how we can construct an expression $E$ that maps a word $w$ to value 0 if and only if this word is the encoding of a halting computation of $M$ from valuation $\nu_0$, and to a negative value otherwise.

First, we note that this can be done by providing for each of the conditions an expression which returns 0 when the condition is satisfied and a negative value otherwise. Then we simply need to combine those expressions with the $+$ operator: the $+$ expression will be equal to 0 only if all the expressions are equal to 0, and it will be negative otherwise.

Second, we note that all the constraints in the list above are regular constraints with the exception of $5(c)$ and $6(b)$. Being regular, all the other constraints can be directly encoded as deterministic sum-automaton and so trivially as iterated-sum expressions. We concentrate here on the constraints that require the use of iteration, and we detail the construction for constraint $5(c)$ as the construction for $6(b)$ is similar and simpler.

For constraints $5(c)$, we construct an iterated-sum expression $E_{5(c)}$ that decomposes the word uniquely as factors of the form $\vdash q_1 a^{n_1} b^{m_1} \delta q_2 a^{n_2} b^{m_2} \dashv$. On each factor, we evaluate an simple expression whose value is non-negative if and only if the update defined by $\delta$ is correctly realized in the encoding. To show how to achieve this, assume for the illustration that $\delta$ is incrementing the counter $x$ and let us show how this can be checked. The expression that we construct in this case computes the minimum of $1 + n_1 - n_2$ and $-1 - n_1 + n_2$. It should be clear that this minimum is equal to 0 if and only if $n_2 = n_1 + 1$ (i.e. when the increment is correctly realized). In turn, it is a simple exercise to construct a deterministic weighted automaton to compute $n_1 + 1 - n_2$ and one to compute $n_2 - n_1 - 1$. All the different updates can be treated similarly and thus using only simple expressions. Now, the iterated-sum expression $E_{5(c)}$ simply take the sum of all the values obtained locally on all the factors of the decomposition. This sum is non-negative if and only if all the values computed locally are non-negative. ♦

Remark that, another option would be to define the semantics of $E^{\circledast}$ as an iteration of max, i.e. $\mathtt{dom}(E^{\circledast})$ is still the set of words $u$ that are uniquely decomposed into $u_1 \dots u_n$ with $u_i \in \mathtt{dom}(E)$, but $[\![E]\!](u) = \max\{[\![E]\!](u_i) \mid i = 1, \dots, n\}$. This variant is again undecidable with respect to classical quantitative decision problems. Indeed, a careful inspection of the proof above show that in constraint $E_{5(c)}$ and $E_{6(b)}$, we can replace the iteration of sum by iteration of min, or equivalently, if we first reverse the sign of all expression, by the iteration of max. In that case the max will be non-positive if and only if the 2-counter machine admits a halting computation.

## 3.2.2 Synchronized expressions

A close inspection of the proof of Theorem 3.2.5, reveals that the undecidability stems from the asynchronism between parallel star operators, and in the way they decompose the input word (decomposition based on $\vdash \cdots \dashv$ or $\triangleright \cdots \triangleleft$). The two overlapping decompositions are needed. By disallowing this, decidability is recovered: sub-expressions $F^{\circledast}$ and $G^{\circledast}$ at the same star depth must decompose words in exactly the same way.

Let us formalize the notion of star depth. Given an iterated-sum expression $E$, its syntax tree $T(E)$ is a tree labeled by Presburger combinators $\varphi$, star operators $^{\circledast}$, or unambiguous sum-automata $A$. Any node $p$ of $T(E)$ defines a sub-expression $E|_p$ of $E$. The *star depth* of node $p$ is the number of star operators occurring above it, i.e. the number of nodes $q$ on the path from the root of $T(E)$ to $p$ (excluded) labeled by a star operator. E.g. in the expression $\varphi(W_1^{\circledast}, \varphi(W_2^{\circledast}))^{\circledast}$, the sub-expression $W_1^{\circledast}$ has star depth 1 while $W_1$ has star depth 2. The star depth of the root of any expression is 0.

**Definition – synchronization of iterated-sum expressions**

A set of iterated-sum expressions $\mathbb{E}$ is *synchronized*, denoted by the predicate $\mathrm{Sync}(\mathbb{E})$, if for all $F, G \in \mathbb{E}$ (not necessarily distinct), all nodes $p$ of $T(F)$ and nodes $q$ of $T(G)$ at the same star depth, if $F|_p = H^{\circledast}$ and $G|_q = I^{\circledast}$, then $\mathtt{dom}(H) = \mathtt{dom}(I)$. An iterated-sum expression $E$ is synchronized if $\{E\}$ is synchronized.

[XIII] The computation is unique as $M$ is deterministic.

By Proposition 3.2.4, this property is decidable. Asking that $H$ and $I$ have the same domain enforces that any word $u$ is decomposed in the same way by $H^{\circledast}$ and $I^{\circledast}$.

**Example**

The expressions $F, G, H$ of Example 3.2.3 are synchronized. It is obvious for $F$ and $G$ since they contain only one star operation. Recall that $H = \max(F|_L, G|_L)^{\circledast}$ with $F|_L = \max(W_a, W_b)^{\circledast} + C_L$ and $G|_L = \max(W_c, W_d)^{\circledast} + C_L$. Hence $H$ is synchronized iff $\mathtt{dom}(\max(W_a, W_b)^{\circledast}) = \mathtt{dom}(\max(W_c, W_d)^{\circledast})$, which is the case as $\mathtt{dom}(W_a) = \mathtt{dom}(W_b) = \mathtt{dom}(W_c) = \mathtt{dom}(W_d)$.

Finitely ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$ is the largest known class of $\mathsf{WA}^{\max}_{\text{sum}}$ for which emptiness, universality, inclusion and equivalence are decidable [FGR14]. Already for linearly ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$, universality and equivalence problems are undecidable [DGM17]. Example 3.2.1 is realizable by a synchronized expression (Example 3.2.3) or a $\mathsf{WA}^{\max}_{\text{sum}}$ which non-deterministically guesses, for each factor $u_i$, whether it should count the number of $a$ or $b$. However, as shown in Section 3.5 of [KLMP04], it is not realizable by any finitely ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$. As a consequence:

**Proposition 3.2.6**

There is a quantitative language $f$ such that $f$ is definable by a synchronized expression or a $\mathsf{WA}^{\max}_{\text{sum}}$, but not by a finitely ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$.

As a direct consequence of the definition of iterated-sum expressions and synchronization, synchronized expressions are closed under Presburger combinators and unambiguous iterated-sum in the following sense:

**Proposition 3.2.7**

Let $E_1, \ldots, E_n, E$ be iterated-sum expressions and $\varphi$ a Presburger combinator of arity $n$. If $\text{Sync}\,(E_1, \ldots, E_n)$ then $\varphi(E_1, \ldots, E_n)$ is synchronized, and if $E$ is synchronized, so is $E^{\circledast}$.

Despite the fact that synchronized expressions can express quantitative languages that are beyond finitely ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$, decidability holds. It is done by converting synchronized expressions into a new model of weighted automata, which, with a suitable notion of synchronization, are decidable with respect to all decision problems considered in this paper. The next two sections are devoted to the definition of the automata model and its decidability in its synchronized restriction.

**Theorem 3.2.8**

The emptiness and universality problems are decidable for synchronized expressions. The inclusion and equivalence problems for iterated-sum expressions $E_1, E_2$ such that $\text{Sync}\,(E_1, E_2)$ holds are decidable.

## 3.3 Weighted Chop Automata

In this section, we introduce a new weighted automata model, called *weighted chop automata* (WCA), into which iterated-sum expressions can be compiled. As a consequence, the undecidability of iterated-sum expressions (Theorem 3.2.5) carries over to WCA. We therefore introduce the class of synchronized WCA, into which synchronized expressions can be compiled in thus decidability is recovered, proving Theorem 3.2.8. The intuitive behavior of a WCA is as follows. A hierarchical NFA (whose transitions are not reading single letters but words in some regular language) “chop” the input word into factors, on which expressions of the form $\varphi(C_1, \ldots, C_n)$, where $C_i$ are smaller WCA, are applied to obtain intermediate values, which are then summed to obtain the value of the whole input word. In the following, we first introduce hierarchical NFA, then WCA and finally their synchronized restriction.

**Definition – hierarchical regular automata**

A *hierarchical* NFA is a tuple $A = (Q, I, F, \Delta)$ where $Q$ is a set of states, $I$ its initial states and $F$ its final states, and $\Delta$ maps any pair $(p, q) \in Q^2$ to a regular language $\Delta(p, q) \subseteq \Sigma^*$ finitely represented by some (classical) NFA.

A run $\varrho$ of $A$ over a word $u \in \Sigma^*$ is a sequence $q_0 u_1 \dots q_{n-1} u_n q_n$ such that $u = u_1 \dots u_n$ and $u_i \in \Delta(q_{i-1}, q_i)$ for each $i$. It is accepting if $q_0 \in I$ and $q_n \in F$. We say that $A$ is an unambiguous hierarchical NFA (hierarchical UFA for short) if for all $u \in \Sigma^*$, there is at most one accepting run of $A$ on $u$ (and hence its decomposition $u_1 \dots u_n$ is unique). We define the representation size of a hierarchical finite automaton $A = (Q, I, F, \Delta)$ as $|A| = \sum_{p,q \in Q} n_{p,q} + \mathtt{card}(Q) + \mathtt{card}(\Delta)$ where $n_{p,q}$ is the number of states of the NFA recognizing $\Delta(p, q)$.

**Proposition 3.3.1**

We can decide whether a hierarchical NFA is unambiguous in PTime.

**Proof** Let $A = (Q, I, F, \Delta)$ be a hierarchical NFA whose languages $\Delta(q, q')$ are given by NFA $A_{q,q'} = (Q_{q,q'}, I_{q,q'}, F_{q,q'}, \Delta_{q,q'})$. We construct in polynomial time a finite transducer $T = (P, \{p_I\}, \{p_F\}, \Delta')$ as defined in Section 1.4. Given $u \in L(A)$ as input word, $T$ outputs any run of $A$ on $u$. Clearly, $T$ defines a function iff $A$ is unambiguous. In particular, the transition relation $\Delta'$ has type $\Delta' \subseteq P \times \Sigma^* \times \Gamma^* \times P$, where $\Gamma$ is the output alphabet. Transitions are denoted by $p \xrightarrow{u \mid v} q$ where $u$ is the input word and $v$ the output word. In general, a transducer defines a binary relation from input to output words, but deciding whether it denotes a function is decidable in PTime [FGR15].[XIV] To construct $T$, the idea is as follows. We take $\Gamma = Q \cup \Sigma$ has output alphabet. The set of states is defined by $P = \{p_I, p_F\} \uplus \biguplus_{q,q' \in Q} Q_{q,q'}$. The transition function is defined by:

- From its initial state $p_I$, the transducer can only enters into some sub-automaton $A_{q_I,q}$ and outputs the initial state $q_I$ that $A$ should have visit i.e. $p_I \xrightarrow{\varepsilon \mid q_I} s_I$, for all $q_I \in I$ and $s_I \in \bigcup_{q \in Q} I_{q_I,q}$.
- To move from a sub-automaton $A_{q',q}$ to a sub-automaton $A_{q,q''}$, the transducer outputs the state $q$ that $A$ should have visited i.e. $s_F \xrightarrow{\varepsilon \mid q} s$, for all $q \in Q$ and $s_F \in \bigcup_{q' \in Q} F_{q',q}$ and $s_I \in \bigcup_{q'' \in Q} I_{q,q''}$.
- To reach its final state $p_F$, the transducer can only exits some sub-automaton $A_{q,q_F}$ and outputs the final state $q_F$ that $A$ should have visited i.e. $s \xrightarrow{\varepsilon \mid q_F} p_F$, for all $q_F \in F$ and $s_F \in \bigcup_{q \in Q} F_{q,q_F}$.
- Inside a sub-automaton, only symbols from $\Sigma$ are written on the output i.e. $s \xrightarrow{\sigma \mid \sigma} s'$, for all $(s, \sigma, s') \in \bigcup_{q,q' \in Q} \Delta_{q,q'}$. ♦

## 3.3.1 Definition and closure properties

We formally define weighted chop automata, then investigate their closure properties.

**Definition – weighted chop automata**

The class of *weighted chop automata* is inductively defined as follows.

- A 0-weighted chop automaton is an unambiguous sum-automaton.
- For $m > 0$, an $m$-weighted chop automaton ($m$-WCA) is a tuple $C = (A, \lambda)$ where $A$ is a hierarchical UFA and $\lambda$ is a function mapping any pair $(p, q) \in Q^2$ to some term $\varphi(C_1, \dots, C_n)$ where for all $i$, $C_i$ is an $m'$-WCA, with $m' < m$ and $\varphi$ is a Presburger combinator of arity $n$. Moreover, it is required that at least one $C_i$ is an $(m-1)$-WCA.

By definition of $m$-WCA, $m$ is unique and is called the chop level of $C$. A WCA $C$ is an $m$-WCA for some $m$. This definition is generalized to outputs in $\mathbb{Z}^d$ for some $d \in \mathbb{N}$ in Section 3.4.

A WCA $C$ defines a quantitative language $[\![C]\!] \colon \Sigma^* \to \mathbb{Z}$ of domain $\mathtt{dom}(C)$ inductively defined as follows: If $C$ is a 0-WCA, then its semantics is that of unam-

[XIV] Corollary 6.3.5 proves that the functionally of a transducer is in NLogSpace.

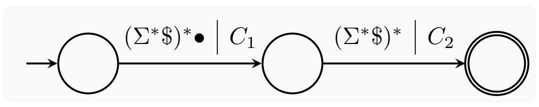
biguous sum-automata defined in Section 1.4. Otherwise $C = (A, \lambda)$, and the set $\mathtt{dom}(C)$ is the set of words $u = u_1 \dots u_n$ on which there exists one (and only one) accepting run $\varrho = q_0 u_1 \dots q_{n-1} u_n q_n$ of $A$ such that for all $1 \leq i \leq n$, let $\lambda(q_{i-1}, q_i)$ be of the form $\varphi(C_1, \dots, C_k)$, then $u_i \in \bigcap_{j=1}^{k} \mathtt{dom}(C_j)$, and in this case we define $v_i = [\![\varphi]\!]\big([\![C_1]\!](u_i), \dots, [\![C_k]\!](u_i)\big)$. The value of $\varrho$ (which also defines the value of $u$) is then $\sum_{i=1}^{n} v_i$. The range of $C$ is set of values that $[\![C]\!]$ can outputs i.e. $R(C) = \{[\![C]\!](u) : u \in \mathtt{dom}(u)\}$.

The unique sequence $\big(u_1, \lambda(q_0, q_1)\big) \dots \big(u_n, \lambda(q_{n-1}, q_n)\big)$ is denoted in the sequel by $\mathrm{dec}_C(u)$. We define the representation size of a WCA $C = (A, \lambda)$ with $A = (Q, I, F, \Delta)$ as $|C| = \sum_{p,q \in Q} n_{p,q} + \mathtt{card}(Q) + \mu\mathtt{card}(\Delta)$ where $n_{p,q}$ is the number of states of the NFA recognizing $\Delta(p, q)$ and $\mu$ is the maximal size of expressions given by $\lambda$ formally, $\mu = \max\{|\varphi| + \sum_{i=1}^{k} |C_i| : p, q \in Q \wedge \lambda(p, q) = \varphi(C_1, \dots, C_k)\}$.

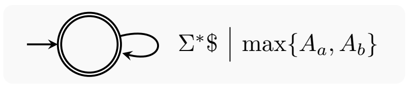
**Example 3.3.2**

Let $\Sigma = \{a, b, c, d\}$ and $\bullet, \$ \notin \Sigma$. The 1-WCA $C$ depicted below realizes the function which maps any word of the form $u_1\$ \dots u_n\$ \bullet v_1\$ \dots v_m\$$, where $u_i, v_i \in \{a, b, c, d\}^*$ to $\sum_{i=1}^{n} \max(|u_i|_a, |u_i|_b) + \sum_{i=1}^{m} \max(|v_i|_c, |v_i|_d)$. Note that, here we write $C_i$ as short-cuts for $\varphi_{id}(C_i)$ where $\varphi_{id}$ defines the identify function. For all $\sigma \in \{a, b, c, d\}$, the automaton $A_\sigma$ is an unambiguous sum-automaton that counts the number of occurrences of $\sigma$ in the input word.

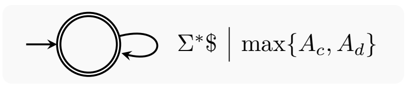
$(C)$



$(C_1)$



$(C_2)$



Here, WCA are unambiguous by definition. Note that, a fully non-deterministic version of WCA can be defined, by using a max aggregator to combine the values of all accepting runs. Such definition would generalize $\mathsf{WA}^{\max}_{\mathrm{sum}}$. But as we will see, the $> 0$-emptiness problem is already undecidable for (unambiguous) WCA, and further synchronization restriction will be necessary to recover decidability.

**Proposition 3.3.3**

The domain of any WCA is regular and constructible.

**Proof** Let $C = (A, \lambda)$ be an $m$-WCA where $A = (Q, I, F, \Delta)$. We show by induction on $m$ that $\mathtt{dom}(C)$ is regular. If $m = 0$ then $C$ is an unambiguous sum-automaton and its domain is regular (given by its underlying NFA). Otherwise, assume by induction hypothesis that for all $m' < m$ the domain of any $m'$-WCA is regular. We construct a hierarchical NFA to recognize $\mathtt{dom}(C)$. Let $p, q \in Q$ and $\lambda(p, q) = v$. We define $\mathtt{dom}(v) = \bigcap_{j=1}^{k} \mathtt{dom}(C_j)$ if $v = \varphi(C_1, \dots, C_k)$. By induction hypothesis each $\mathtt{dom}(v)$ is regular since all $C_j$ is a $m'$-WCA with $m' < m$. By definition of $\mathtt{dom}(C)$, the domain of $C$ is the language of the hierarchical NFA $A' = (Q, I, F, \Delta')$ where for all $q, p \in Q$ we have $\Delta'(p, q) = \Delta(p, q) \cap \mathtt{dom}(v)$. ♦

**Closure properties**

We now investigate the closure properties of WCA. Given two quantitative languages $f_1, f_2$, let us define their *split sum* $f_1 \odot f_2$ as the function mapping any word $u$ which can be uniquely decomposed into $u_1, u_2$ such that $u_i \in \mathtt{dom}(f_i)$ for all $i$, to $f_1(u_1) + f_2(u_2)$ [AFR14]. We also define the conditional choice $f_1 \triangleright f_2$ as the mapping of any word $u \in \mathtt{dom}(f_1)$ to $f_1(u)$, and of any word $u \in \mathtt{dom}(f_2) \setminus \mathtt{dom}(f_1)$ to $f_2(u)$ [AFR14]. These operators may be thought of as (unambiguous) concatenation and disjunction in rational expressions. WCA are closed under these two operations, as well as Presburger combinators, (unambiguous) iterated-sum and regular domain restriction, in the sense given by Proposition 3.3.5.

First, we show a property on the split sum of languages.

**Lemma 3.3.4**

Let $L_1, L_2$ be two regular languages $L_1, L_2$ both realizable by some hierarchical NFA with at most $n$ states. If $L_1 \odot L_2$ denotes the set of words $u$ which can be uniquely decomposed into $u_1u_2$ with $u_i \in L_i$, then there exists $2n$ regular languages $N_i, M_i$ such that $N_i \subseteq L_1$ and $M_i \subseteq L_2$, and $L_1 \odot L_2 = \bigcup_{i=1}^{n} N_iM_i$.

**Proof** Let $A_1, A_2$ be two hierarchical NFA recognizing $L_1, L_2$ respectively. We can suppose w.l.o.g. that $A_1, A_2$ are classical NFA by replacing in linear time a call to a sub-automaton by the automaton itself. We can construct an automaton $A$ that accepts the words in $L_1L_2$ which admits at least two different factorizations with respect to $L_1, L_2$. It suffices to simulate two runs of $A_1$ in parallel, whenever a copy of $A_1$ goes to an accepting state, this copy either stays in $A_1$ or, thanks to an added $\varepsilon$-transition, goes to some initial state of $A_2$. We also add one bit of memory to check that $\varepsilon$-transitions have been taken at two different moments in the two simulated runs. The accepting states are states $(q_2, q_2', 1)$ where $q_2, q_2'$ are accepting states of $A_2$. By complementing $A$, we obtain an automaton, say $B$, recognizing $L_1 \odot L_2$.
Now, we make a product between $B$ and a disjunct union between $A_1$ and $A_2$. If $Q$ are the states of $B$, $Q_1$ of $A_1$, $Q_2$ of $A_2$ (with initial states $I_2$), the set of states of this product is $Q \times (Q_1 \uplus Q_2 \uplus I_2')$ where $I_2'$ is a copy of $I_2$. $B$ initially runs in parallel of $A_1$ and, when $A_1$ enters an accepting state, i.e. the product is in state $(q, q_1)$ where $q_1 \in F_1$, then we add some $\varepsilon$-transition to any state $(q, q_2')$ where $q_2 \in I_2$. Then, from states of this form, the product continues its simulation of $B$ and simulates in parallel $A_2$ (in normal states $Q_2$, so that the copy $I_2'$ is only met once, when the product switches to $A_2$). Let denote by $B \boxtimes (A_1A_2)$ this product. We set its accepting states to be any pair $(q, q_2)$ or $(q, q_2')$ where $q_2$ and $q$ are accepting. For all states $(q, p)$ of $B \boxtimes (A_1A_2)$, we denote by $L_{q,p}$ the set of words that admit a run from some initial state of $B \boxtimes (A_1A_2)$ to the state $(q, p)$, and dually, we denote $R_{q,p}$ the set of words that admit a run from the state $(q, p)$ to some accepting state of $B \boxtimes (A_1A_2)$. We claim that:

$$L_1 \odot L_2 = L(B) = \bigcup_{(q,q_2') \in Q \times I_2'} L_{q,q_2}R_{q,q_2}$$

Clearly, $\bigcup_{(q,q_2') \in Q \times I_2'} L_{q,q_2}R_{q,q_2} \subseteq L(B)$ since the product also checks that the input words are accepted by $B$. Conversely, if $u \in L(B)$, then it is uniquely decomposed into $u_1u_2$ where $u_i \in L(A_i)$. From $\varrho$ an accepting run of $B$ that visits the states $q_1 \dots q_{n+1}p_1 \dots p_{m+1}$ while reading $u$ (where $n = |u_1|$ and $m = |u_2|$), an accepting run $\varrho_1$ that visits the states $\alpha_1 \dots \alpha_{n+1}$ of $A_1$ while reading $u_1$, and a accepting run $\varrho_2$ that visits the states $\beta_1 \dots \beta_{m+1}$ of $A_2$ while reading $u_2$, we can construct the following accepting run of $B \boxtimes (A_1A_2)$:

$$(q_1, \alpha_1) \dots (q_{n+1}, \alpha_{n+1})(p_1, \beta_1')(p_2, \beta_2) \dots (p_{m+1}, \beta_{m+1})$$

♦

We now turn to the closure properties of WCA.

**Proposition 3.3.5 – Closure Properties of WCA**

The class of quantitative languages defined by WCA is closed under split sum, conditional choice, Presburger combinators, iterated-sum and regular domain restriction.
More precisely, let $C, C_1, \dots, C_k$ be WCA, $\varphi$ be a Presburger combinator of arity $k$ and $L \subseteq \Sigma^*$ a regular language. One can construct WCA respectively denoted by $\varphi(C_1, \dots, C_k)$, $C^{\circledast}$, $C_1 \odot C_2$, $C_1 \triangleright C_2$ and $C|_L$ such that

- $\mathtt{dom}(\varphi(C_1, \dots, C_k)) = \bigcap_{i=1}^{k} \mathtt{dom}(C_i)$ and for all $u \in \bigcap_{i=1}^{k} \mathtt{dom}(C_i)$,

$$[\![\varphi(C_1, \dots, C_n)]\!](u) = [\![\varphi]\!]\big([\![C_1]\!](u), \dots, [\![C_n]\!](u)\big)$$

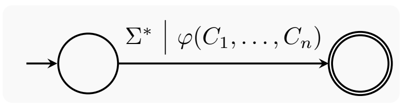
- $[\![C^{\circledast}]\!] = [\![C]\!]^{\circledast}$, $[\![C \odot D]\!] = [\![C]\!] \odot [\![D]\!]$ and $[\![C \triangleright D]\!] = [\![C]\!] \triangleright [\![D]\!]$, $[\![C|_L]\!] = [\![C]\!]|_L$

**Proof** We prove the closure under each operator one by one.

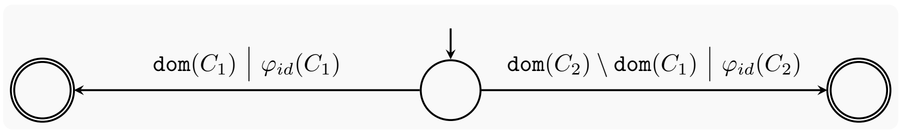
*Closure under star.* Let $C^{\circledast} = (A, \lambda)$ defined as follows. The only difficulty is that it should be unambiguous (because we want decomposition to be unique), hence it is not correct to add some $\varepsilon$-transition from accepting states of $C$ to its initial states. However, it is possible to define an unambiguous NFA $B = (Q, I, F, \Delta)$ with a set of special states $S \subseteq Q$ such that $L(B) = \mathtt{dom}(C)^{\circledast}$ and such that for all $u \in L(B)$, the occurrences of special states in the accepting run of $B$ on $u$ decomposes (uniquely) $u$ into factors that belong to $\mathtt{dom}(C)$. Then, the states of $C^{\circledast}$ are the states $S \cup I$, the initial states $I$, final states $S \cap F$, and $\Delta(s, s')$ for all $s \in S \cup I$ and $s' \in S$, is the set of words on which there is a run of $B$ from $s$ to $s'$ that does not pass by any state of $S$ (except at the end and beginning). This set is easily shown to be regular. Finally, $\lambda(s, s') = \varphi_{id}(C)$ where $\varphi_{id}$ defines the identity function.

*Closure under regular domain restriction.* If $C = (A, \lambda)$, then it suffices to take the product of $A$ with any DFA $B$ such that $L(B) = L$. Since $A$ is a hierarchical NFA, the product is a bit different than the usual automata product. Assume $A = (Q, I, F, \Delta)$ and $B = (P, p_0, F', \delta')$ (a classical DFA). Then $A \times B = (Q \times P, I \times \{p_0\}, F \times F', \Delta \times \delta')$ where for all $(q, p), (q', p') \in Q \times P$ we have that $\Delta \times \delta'\big((q, p), (q', p')\big)$ is the set of words in $\Delta(q, q')$ such that there exists a run of $B$ from state $p$ to state $p'$. This set is effectively regular. The resulting hierarchical NFA is unambiguous since $B$ was taken to be deterministic and $A$ is unambiguous.

*Closure under Presburger combinators.* The WCA $\varphi(C_1, \ldots, C_n)$ is defined as:

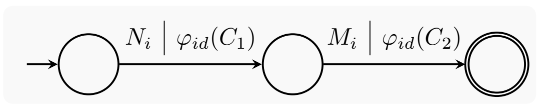


*Closure under conditional choice*The WCA $C_1 \triangleright C_2$ is defined as


$\mathtt{dom}(C_1) \mid \varphi_{id}(C_1)$
$\mathtt{dom}(C_2) \setminus \mathtt{dom}(C_1) \mid \varphi_{id}(C_2)$

Note that $\mathtt{dom}(C_i)$ are regular by Proposition 3.3.3.

*Closure under split sum.* By Lemma 3.3.4, $\mathtt{dom}(C_1) \odot \mathtt{dom}(C_2) = \bigcup_{i=1}^{n} N_i M_i$ for some regular languages $N_i \subseteq \mathtt{dom}(C_1)$ and $M_i \subseteq \mathtt{dom}(C_2)$. For all $i = 1, \ldots, n$, we define the WCA $(C_1 \odot C_2)|_{N_i M_i}$ as depicted below:



Note that it is unambiguous since $N_i M_i \subseteq \mathtt{dom}(C_1) \odot \mathtt{dom}(C_2)$, $N_i \in \mathtt{dom}(C_1)$ and $M_i \in \mathtt{dom}(C_2)$. Furthermore, for all $u \in N_i M_i$ such that $u = u_1 u_2$ with $u_1 \in \mathtt{dom}(C_1)$ and $u_2 \in \mathtt{dom}(C_2)$ we have $[\![(C_1 \odot C_2)|_{N_i M_i}]\!](u) = [\![C_1]\!](u_1) + [\![C_2]\!](u_2)$. Finally, we let $C_1 \odot C_2 = \rhd_{i=1}^{n} (C_1 \odot C_2)|_{N_i M_i}$ (the way this expression is parenthesized, as well as the order in which the index $i$ are taken, does not change the semantics). ♦

A direct consequence of the closure properties of WCA is that any iterated-sum expression can be encoded into an equivalent WCA.

**Corollary 3.3.6**

Given an iterated-sum expression $E$ we can construct a WCA $C$ such that $[\![E]\!] = [\![C]\!]$.

The undecidability of WCA is implied by Theorem 3.2.5 and Corollary 3.3.6. Furthermore, the iterated-sum expression of Theorem 3.2.5 can be encoded by a 1-WCA.

**Corollary 3.3.7**

Emptiness, universality, inclusion and equivalence for WCA are undecidable problems, even for 1-WCA.

## 3.3.2 Synchronized weighted chop automata

As for iterated-sum expressions, we present a subclass of WCA with a notion of synchronization into which synchronized expressions can be compiled. Then, in the next section, we show the decidability of synchronized WCA.

**Definition**

The notion of synchronization of WCA is inductively defined:

- Two terms $\varphi_1(C_1, \dots, C_k)$ and $\varphi_2(C'_1, \dots, C'_{k'})$ are synchronized if $C_i$ is synchronized with $C'_j$ for all $1 \le i \le k$ and all $1 \le j \le k'$.
- Two WCA $C_1, C_2$ are synchronized, denoted by $\text{Sync}(C_1, C_2)$, if they are either both 0-WCA, or $C_1 = (A_1, \lambda_1)$ and $C_2 = (A_2, \lambda_2)$, and the following holds: for all $u \in L(A_1) \cap L(A_2)$, if $\text{dec}_{C_1}(u) = (u_1, E_1), \dots, (u_n, E_n)$ and $\text{dec}_{C_2}(u) = (v_1, F_1), \dots, (v_m, F_m)$, then $n = m$ and for all $1 \le i \le n$, we have $u_i = v_i$ and $E_i$ is synchronized with $F_i$.

We write $\text{Sync}(C_1, \dots, C_n)$ if $\text{Sync}(C_i, C_j)$ for all $i, j \in \{1, \dots, n\}$. Now, a WCA $C$ is synchronized if it is an unambiguous sum-automaton, or it is of the form $(A, \lambda)$, and any expression $\varphi(C_1, \dots, C_k)$ in the range of $\lambda$ satisfies $\text{Sync}(C_1, \dots, C_k)$.

Note that, the base case synchronizes sum-automaton with sum-automata only. This induces that the synchronization holds with respects to the chop level. Such property is formalized by the following statement.

**Proposition 3.3.8**

Let $C_1$ and $C_2$ be two WCA. As a consequence of the definition of synchronization, if $\text{Sync}(C_1, C_2)$, then both $C_1$ and $C_2$ are $m$-WCA for the same $m$.

In Example 3.3.2 the WCA $C$ have a chop level 1 and cannot be synchronized with $C_1$ or $C_2$ which have a both chop level 0. However, $C$ is synchronized since any Presburger combinators of $C$ have an arity 1 and $C_1, C_2$ are unambiguous sum-automata satisfying $\text{Sync}(C_1, C_2)$ trivially.

**Proposition 3.3.9**

Let $C_1, C_2$ be two WCA. Deciding whether $\text{Sync}(C_1, C_2)$ holds can be done in PTime.

**Proof** We provide a recursive algorithm which takes two chop automata $C_1, C_2$ and checks whether $\text{Sync}(C_1, C_2)$ in PTime. If $C_1$ and $C_2$ are two unambiguous sum-automaton, then the algorithm returns true. If only one of them is an unambiguous sum-automaton and the other not, then the algorithm returns false.
Now, consider the case where we have chop automata $C_1 = (A_1, \lambda_1)$ and $C_2 = (A_2, \lambda_2)$. We first show how to decide the following (weaker) property: for all $u \in L(A_1) \cap L(A_2)$, if $\text{dec}_{C_1}(u) = (u_1, E_1), \dots, (u_n, E_n)$ and $\text{dec}_{C_2}(u) = (v_1, F_1), \dots, (v_m, F_m)$, then $n = m$ and for all $i \in \{1, \dots, n\}$, we have $u_i = v_i$.
As in Proposition 3.3.1, the idea is to construct a transducer $T$, which defines a function from $\Sigma^*$ to $2^{\Sigma^*}$, whose domain is $L(A_1) \cap L(A_2)$. Given $u \in \texttt{dom}(T)$, if $\text{dec}_{C_1}(u) = (u_1, E_1), \dots, (u_n, E_n)$ and $\text{dec}_{C_2}(u) = (v_1, F_1), \dots, (v_m, F_m)$, then $T$ returns the set of words $\{u_1\#u_2\# \dots \#u_n, v_1\#v_2\# \dots \#v_n\}$. By definition of a WCA decomposition $u_1 \dots u_n = v_1 \dots v_n = u$, then the latter set is a singleton i.e. $u_1\#u_2\# \dots \#u_n = v_1\#v_2\# \dots \#v_n$ iff the two decompositions are equal. Hence, it suffices to decide whether $T$ defines a function (i.e. is functional), which can be done in PTime in the size of $T$ (see for instance [BCPS03]).

It remains to show how to construct $T$. $T$ is the disjoint union of two transducers $T_1$ and $T_2$, which respectively output $u_1\#u_2\#\dots\#u_n$ and $v_1\#v_2\#\dots\#v_n$. Basically $T_i$ can be seen as a copy of $A_i$ where each transitional NFA are replaced by transducer which output the input with $\#$ as an ending maker. Consider $T_1$. Let $A_1 = (Q, q_0, F, \Delta)$ and $(A_{p,q})_{p,q\in Q}$ be NFA that recognize $\Delta(p,q)$. Whenever a transition $(\alpha,\sigma,\beta)$ of $A_{q,q'}$ is fired, $T_1$ makes several choices. Either $q'$ is not final in $A_{q,q'}$ and $T_1$ moves to state $\beta$ and write $\sigma$ on the output. Either $q'$ is final, in that case $T_1$ may move to state $\beta$ while writing $\sigma$ on the output, or move to the initial state of some automaton $A_{q',q''}$ for some non-deterministically chosen state $q'' \in Q$, and write $\sigma\#$ on the output. Clearly, $T_1$ has a polynomial size in the size of $C_1$.
In order to decide the synchronization between $C_1$ and $C_2$, we also need to check that $\mathrm{Sync}\,(E_i, F_i)$ for all sub-expressions $E_i, F_i$ that occur at the same position in some decomposition. Formally, let $S$ be the set of expressions $E, F$ such that there exists $u \in L(A_1) \cap L(A_2)$, such that $\mathrm{dec}_{C_1}(u) = (u_1, E_1), \dots, (u_n, E_n)$ and $\mathrm{dec}_{C_2}(u) = (v_1, F_1), \dots, (v_m, F_m)$ and there exists $i$ such that $E_i = E$ and $F_i = F$. We will show that $S$ can be computed in PTIME. Once $S$ has been computed, for every pair $\big(\varphi(C_1,\dots,C_n), \varphi'(C'_1,\dots,C'_m)\big) \in S$, it suffices to call this algorithm on each pair $(C_i, C'_j)$ for all $i, j$.
It remains to show that $S$ can be computed in polynomial time. Again, for all expressions $E, F$ occurring in the range of $\lambda_1$ and $\lambda_2$ respectively, one could define some automaton $A_{E,F}$ (of polynomial size) which accepts a word $u \in L(A_1)\cap L(A_2)$ iff $E, F$ occurs together in the respective decomposition of $u$, i.e. $E \in \mathrm{dec}_{C_1}(u)$ and $F \in \mathrm{dec}_{C_2}(u)$. Then, for all these pairs, if $L(A_{E,F}) \neq \varnothing$ (which can be checked in PTIME), then we add $(E, F)$ to $S$. To construct $A_{E,F}$, the idea is to simulate, via a product construction, an execution of $C_1$ and an execution of $C_2$ in parallel (by also simulating the smaller automata defining the regular languages on the transitions of $C_1$ and $C_2$). In this product construction, one bit of memory is used to remember whether a pair of states $(p_1, p_2)$ of $C_1$ and $C_2$ respectively, such that $E = \lambda_1(p_1)$ and $F = \lambda_2(p_2)$, was reached. The automaton accepts if such a pair was found, and the simulation of the two runs accept (meaning that $u \in L(A_1) \cap L(A_2)$).
◆

Since a WCA is synchronized iff any of its Presburger combinators are parameterized by a synchronized set of WCA, Proposition 3.3.9 implies the following.

**Corollary**

Deciding whether a WCA is synchronized is in PTIME.

## 3.4 Decidability by synchronization

In this section, we prove that emptiness, universality, inclusion and equivalence problems are decidable for synchronized WCA. Then, we show that synchronized iterated-sum expressions can be effectively converted into synchronized WCA, thus proving their decidability (Theorem 3.2.8).

### Overview of the proof of synchronized WCA decidability

The cornerstone of the proof is proving the semi-linearity of the range of synchronized WCA. Let us give the intuitive ideas on how this can be shown. Assume for the time being that we consider a synchronized WCA $C = (A, \lambda)$ with a single state $q$, and hence a single transition from $q$ to $q$ on any word of $\Delta(q,q)$. Assume that $\lambda(q,q) = \varphi(C_1,\dots,C_n)$. Then, in order to prove that the range $R(C)$ of the function $[\![C]\!]$ is semi-linear, we have to show that $\{\big([\![C_1]\!](u),\dots,[\![C_n]\!](u)\big) : u \in \Delta(q,q) \cap \bigcap_{i=1}^n \mathtt{dom}(C_i)\}$ is a semi-linear set. For a general synchronized WCA, if one wants to prove the result by induction on its structure, a stronger statement is needed, namely to consider tuples of WCA instead of a single one. In particular, we show (Lemma 3.4.2) the following result: For all $C_1,\dots,C_n$ WCA such that $\mathrm{Sync}\,(C_1,\dots,C_n)$, the following set is semi-linear and constructible:

$$\left\{ \big([\![C_1]\!](u),\dots,[\![C_n]\!](u)\big) : u \in \bigcap_{i=1}^{n} \mathtt{dom}(C_i) \right\}$$

While product construction naturally extends unambiguous sum-automata to higher dimension for the outputs, such construction requires synchronization for WCA. So, under the condition that any word is decomposed in the same way, we define the product of $n$ given $m$-WCA, for some $m$, as a WCA weighted in $\mathbb{Z}^n$. Then, we show semi-linearity of the range of synchronized WCA, by induction on its chop level (level 0 corresponds to unambiguous sum-automata with values in $\mathbb{Z}^n$). First, we generalize the definition of WCA.

**Definition – generalized weighted chop automata**

Let $d \in \mathbb{N}$. The class of *generalized WCA* of dimension $d$ extends the class of WCA as follows. A generalized 0-WCA is an unambiguous sum-automaton with values in $\mathbb{Z}^d$. For $m > 0$, a generalized $m$-WCA is defined as a WCA $(A, \lambda)$, except that $\lambda$ returns $d$-tuples of terms of the form $\varphi(C_1, \dots, C_k)$, where $C_i$ are generalized $m'$-WCA with $m' < m$ and $\varphi$ is a Presburger combinator of arity $d \times k$ that returns a $d$-tuple of values.

Hence, a generalized WCA of dimension $d$ denotes a quantitative language $[\![C]\!] \colon \Sigma^* \to \mathbb{Z}^d$ of domain $\mathtt{dom}(C)$. The semantics carry over from (non-generalized) WCA, in particular, the range of $C$ is the set of values $R(C) = \{[\![C]\!](u) : u \in \mathtt{dom}(C)\}$. We define the representation size of a generalized WCA $C = (A, \lambda)$ of dimension $d$ with $A = (Q, I, F, \Delta)$ as $|C| = \sum_{p,q \in Q} n_{p,q} + \mathtt{card}(Q) + d\mu\mathtt{card}(\Delta)d$ where $n_{p,q}$ is the number of states of the NFA recognizing $\Delta(p, q)$ and $\mu$ is the maximal size of expressions given by $\lambda$ formally, $\max\{|\varphi| + \sum_{i=1}^{k} |C_i| : p, q \in Q \wedge \lambda(p, q) = \varphi(C_1, \dots, C_k)\}$. The notion of synchronization is defined the same way as for WCA. Just to make it clear, $C_1 = (A_1, \lambda_1)$ and $C_2 = (A_2, \lambda_2)$ are synchronized if for all $u \in \mathtt{dom}(C_1) \cap \mathtt{dom}(C_2)$, we let:

$$\begin{aligned}
\mathrm{dec}_{C_1}(u) &= (u_1, (E_{1,1}, \dots, E_{1,d})), \dots, (u_n, (E_{n,1}, \dots, E_{n,d})) \\
\mathrm{dec}_{C_2}(u) &= (u'_1, (E'_{1,1}, \dots, E'_{1,d'})), \dots, (u'_n, (E'_{n',1}, \dots, E'_{n',d'}))
\end{aligned}$$

then $n = n'$, for each $i \in \{1, \dots, n\}$, $j \in \{1, \dots, d\}$ and $j' \in \{1, \dots, d'\}$ we have $u_i = u'_i$ and if $E_{i,j}$ and $E'_{i,j'}$ are of the respective form $\varphi(C_1, \dots, C_k)$ and $\varphi'(C'_1, \dots, C'_{k'})$, then $\mathrm{Sync}\,(C_\ell, C'_{\ell'})$ for all $\ell \in \{1, \dots, k\}$ and $\ell' \in \{1, \dots, k'\}$.

**Product construction**

We now define the product of $C_1, C_2$ two generalized $m$-WCA. If $m = 0$ then $C_1, C_2$ are unambiguous sum-automata with values in $\mathbb{Z}^{d_1}$ and $\mathbb{Z}^{d_2}$ respectively and the product construction is a classical state product construction, whose transitions are valued in $\mathbb{Z}^{d_1+d_2}$. Else if $m > 0$ such that $C_i = (Q_i, I_i, F_i, \Delta_i, \lambda_i)$ for each $i$, we define $C_1 \times C_2 = (Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_2, \Delta, \lambda)$ where $\Delta\big((p_1, p_2), (q_1, q_2)\big) = \Delta_1(p_1, q_1) \cap \Delta_2(p_2, q_2)$ and $\lambda\big((p_1, p_2), (q_1, q_2)\big) = \big(\lambda_1(p_1, q_1), \lambda_2(p_2, q_2)\big)$ for all $p_1, q_1 \in Q_1$ and $p_2, q_2 \in Q_2$.

As a direct consequence of the construction and the definition of synchronization, we have:

**Proposition 3.4.1**

Let $C_1, C_2$ be two generalized WCA. If $\mathrm{Sync}\,(C_1, C_2)$ holds then $\mathtt{dom}(C_1 \times C_2) = \mathtt{dom}(C_1) \cap \mathtt{dom}(C_2)$ and $[\![C_1 \times C_2]\!](u) = \big([\![C_1]\!](u), [\![C_2]\!](u)\big)$ for all $u \in \mathtt{dom}(C_1) \cap \mathtt{dom}(C_2)$.

## 3.4.1 Decidability of synchronized WCA

We show, by induction on its chop level, that any generalized WCA which is synchronized have a semi-linearity range.

**Lemma 3.4.2**

Let $C_1, \dots, C_n$ be $n$ generalized WCA. If $\mathrm{Sync}\,(C_1, \dots, C_n)$ holds then $\{\big([\![C_1]\!](u), \dots, [\![C_n]\!](u)\big) : u \in \bigcap_{i=1}^{n} L(C_i)\}$ is semi-linear and constructible.

**Proof** Let us apply Proposition 3.4.1 to construct the product $P = C_1 \times \cdots \times C_n$ with $\{t_1, \dots, t_k\}$ is its set of transitions and $\lambda$ its weight function. We recall that if $\mathrm{Sync}\,(C_1, \dots, C_n)$ then the $C_i$ are all generalized $m$-WCA for some $m$ by Proposition 3.3.8. So, the proof goes by induction on $m$.

If $m = 0$ then the $C_i$ are all unambiguous sum-automata whose transitions are valued by tuples of integers. We show that $R(P)$ is semi-linear. As stated by Lemma 2.1.4, there exists an existential Presburger formula $\Phi$ for which $(v_1, \dots, v_k) \models \Phi$ iff there exists an accepting run in $P$ such that for all $i$, it triggers the transition $t_i$ exactly $v_i$ times, is semi-linear and constructible. The range can be denoted from $\Phi$ by the following existential Presburger formula:

$$\Psi(x) = \exists v_1, \dots, v_k \ \Phi(v_1, \dots, v_k) \wedge x = \sum_{i=1}^{k} \lambda(t_i) v_i$$

Now, suppose that $m > 0$. By Proposition 3.4.1 we have $\mathtt{dom}(C) = \bigcap_i \mathtt{dom}(C_i)$ and for all $u \in \mathtt{dom}(C)$, $[\![C]\!](u) = \big([\![C_1]\!](u), \dots, [\![C_n]\!](u)\big)$. Therefore, it suffices to show that $R(C) = \{[\![C]\!](u) : u \in \mathtt{dom}(C)\}$ is semi-linear to prove the lemma.

Suppose that $C = (A, \lambda)$ where $A = (Q, I, F, \Delta)$ and $\lambda$ maps any pair $p, q \in Q$ to some $n$-ary tuple of expressions $\big(\varphi_1(C_1^1, \dots, C_{k_1}^1), \dots, \varphi_n(C_1^n, \dots, C_{k_n}^n)\big)$. We can assume that $\Delta(p, q) = \mathtt{dom}(C_j^i)$ for all $1 \leq i \leq n$ and all $1 \leq j \leq k_i$. This is w.l.o.g. because, if $L = \bigcap_{1 \leq i \leq n} \bigcap_{1 \leq j \leq k_i} \mathtt{dom}(C_j^i)$ (which is regular by Proposition 3.3.3), we restrict the domain of any $C_j^i$ to $L$ and replace $\Delta(p, q)$ by $\Delta(p, q) \cap L$, this does not change the semantics of $C$. Closure under regular domain restriction was shown for (non-generalized) WCA in Proposition 3.3.5, but the same proof works for generalized WCA that are synchronized. With this assumption, we get $\mathtt{dom}(C) = L(A)$. We also define the range of $\lambda(p, q)$ as follow:

$$S_{p,q} = \Big\{ \Big( [\![\varphi_1 \, (C_1^1, \dots, C_{k_1}^1)]\!](u), \dots, [\![\varphi_n \, (C_1^n, \dots, C_{k_n}^n)]\!](u) \Big) : u \in \Delta(p, q) \Big\}$$

By synchronization of $C$, all the $C_i^j$ are all $m'$-WCA for some $m' < m$ and $\mathrm{Sync}\big(C_1^1, \dots, C_{k_1}^1, \dots, C_1^n, \dots, C_{k_n}^n\big)$ holds. Then the induction hypothesis on the tuple $\big(C_1^1, \dots, C_{k_1}^1, \dots, C_1^n, \dots, C_{k_n}^n\big)$ gives the semi-linearity of $S_{p,q}$ since $\varphi_1, \dots, \varphi_n$ are semi-linear maps.

Now, sets of $n$-tuple of integers have the structure of a monoid $(2^{\mathbb{Z}^n}, +, \mathbb{0}_n)$ where $+$ is defined by $S + S' = \{s + s' : s \in S, s' \in S'\}$ and $\mathbb{0}_n = \{(0, \dots, 0)\}$ (tuple of arity $n$). Consider the free monoid over $Q \times Q$ and the morphism $f$ from this monoid to $(2^{\mathbb{Z}^n}, +, \mathbb{0}_n)$, defined by $f((p, q)) = S_{p,q}$ for all $(p, q) \in Q \times Q$. It is easily shown that for any regular language $N \subseteq (Q \times Q)^*$, $f(N)$ is semi-linear. It is because the $S_{p,q}$ are semi-linear, and semi-linear sets are (effectively) closed under sum, union, and starring [ES69]. Thus, by taking $N \subseteq (Q \times Q)^*$ as the set of words for the form $(p_1, p_2)(p_2, p_3) \dots (p_{k-1}, p_k)$ such that $p_1 \in I$, $p_k \in F$, and for all $i \in \{1, \dots, k-1\}$, $\Delta(p_i, p_{i+1}) \neq \varnothing$ then the semantics of WCA which sum values along runs gives the desired result.

Details of the regularity of $N$ and the equivalence between $R(C), f(N)$ are given below:

- We prove that $N$ is regular. Let $A_{p,q}$ be the NFA recognizing $\Delta(p, q)$. It is simple to combine the automata $A_{p,q}$ such that $\Delta(p, q) \neq \varnothing$ into a single automaton recognizing $N$. For instance, one can take the disjoint union of all $A_{p,q}$ such that $\Delta(p, q) \neq \varnothing$, add the states of $A$, with $I$ the set of initial states and $F$ the set of final states, and add the following $\varepsilon$-transitions: from any state $p \in Q$, add $\varepsilon$-transitions to the initial states of any NFA $A_{p,q}$, and from any final state of any automaton $A_{p,q}$, add some $\varepsilon$-transition to $q$.
- We prove that $R(C) = f(N)$.
  - $\subseteq$ Let $x \in \mathrm{Range}(C)$. Hence, there exists $u \in \mathtt{dom}(C)$ such that $[\![C]\!](u) = x$. Let $p_1 \xrightarrow{u_1} p_2 \xrightarrow{u_2} p_3 \dots p_k \xrightarrow{u_k} p_{k+1}$ be the accepting run of $A$ and $u$, i.e. $u_i \in \Delta(p_i, p_{i+1})$ for all $i = 1, \dots, k$. We have $\beta = (p_1, p_2)(p_2, p_3) \dots (p_k, p_{k+1}) \in N$. By the semantics of WCA, $x = x_1 + \dots + x_k$ where $x_i = [\![\lambda(p_i, p_{i+1})]\!](u_i)$ for all $i = 1, \dots, k$. Hence $x_i \in S_{p_i, p_{i+1}}$ and $x \in f(\beta) \subseteq f(N)$.
  - $\supseteq$ Let $x \in f(N)$. There exists $(p_1, p_2) \dots (p_k, p_{k+1}) \in N$ such that $x \in S_{p_1, p_2} + \dots + S_{p_k, p_{k+1}}$, by definition of $f$ and $N$. Hence $x = \sum_{i=1}^{k} x_i$ for some $x_i \in S_{p_i, p_{i+1}}$. By definition of the $S_{p_i, p_{i+1}}$, there exists $u_1, \dots, u_k \in \Sigma^*$ such that $u_i \in \Delta(p_i, p_{i+1})$ and $x_i = [\![\lambda(p_i, p_{i+1})]\!](u_i)$. Moreover, by definition of $N$, $p_1$ is initial and $p_{k+1}$ is final, hence $u_1 \dots u_k \in \mathtt{dom}(C)$, and by the semantics of $C$, $[\![C]\!](u) = \sum_i x_i$, i.e. $x \in R(C)$. ♦

A direct consequence of the latter result is the semi-linearity of the range of synchronized WCA:

**Corollary 3.4.3**

Given $C$ a synchronized WCA, $\{[\![C]\!](u) : u \in \mathtt{dom}(C)\}$ is semi-linear and constructible.

The following theorem is a direct consequence of Corollary 3.4.3, Proposition 3.1.5 and Proposition 3.3.5.

**Theorem 3.4.4**

The emptiness and universality problems are decidable for synchronized WCA. The inclusion and equivalence problems for WCA $C_1, C_2$ such that $\mathrm{Sync}\,(C_1, C_2)$ holds are decidable.

## 3.4.2 Decidability of synchronized expressions

We conclude this section by showing that any synchronized expression can be converted into a synchronized WCA. This conversion is constructible and thus the decidability of synchronized expressions (Theorem 3.2.8) becomes a consequence of Theorem 3.4.4.

### Main ideas of the proof

Let us illustrate the intuition of this proof on an example. Take some synchronized expression $E = \varphi(A, B^{\circledast})$ for some unambiguous sum-automata $A, B$, and some Presburger combinator $\varphi$. The difficulty for converting this kind of expression into WCA, comes from the fact that $A$ is applied on the whole input word, while $B$ is applied iteratively on factors of it. The unambiguous sum-automata $A, B$ are both 0-WCA. Then, according to the construction of Proposition 3.3.5, the expression $B^{\circledast}$ can be converted into a 1-WCA which we also denote by $B^{\circledast}$. Therefore, $A$ and $B^{\circledast}$ are not synchronized, since $n$-WCA are synchronized with $n$-WCA only by Proposition 3.3.8[XV]. So, if we let $C_{\varphi(A,B^{\circledast})}$ be the WCA obtained by applying the Presburger combinator $\varphi$ to the 0-WCA $A$ and the 1-WCA $B^{\circledast}$ (as defined in Proposition 3.3.5), we have that $C_{\varphi(A,B^{\circledast})}$ is equivalent to $E$, but it is not a synchronized WCA in general.

To synchronize $A$ and $B^{\circledast}$, the idea is to artificially chop runs of $A$ into smaller pieces, synchronized with $B$. To do that, we construct a 1-WCA consisting to a hierarchical NFA which calls $A$ for partial computations from any state $p$ to any state $q$. Due to synchronization, these partial computations are required to accept words which are in $\mathtt{dom}(B)$. More precisely, for all states $p, q$ of $A$ we define $A_{p,q}$ to be the sum-automaton $A$ with initial state $p$, final state $q$, and whose domain is restricted to $\mathtt{dom}(B)$. Then, all the smaller automata $A_{p,q}$ are combined into a single 1-WCA which simulates successive applications of the automata $A_{p,q}$, by taking care of the fact that the words it accepts must be uniquely decomposable into factors of $\mathtt{dom}(B)$. This resulting 1-WCA, say $C_A$, is necessarily synchronized with $B^{\circledast}$, and we can return the single synchronized 2-WCA $C_{\varphi(C_A,B^{\circledast})}$ obtained by applying $\varphi$ on $C_A$ and $B^{\circledast}$ (again Proposition 3.3.5), which is equivalent to expression $E$. In order to be able to convert any synchronized expression into a synchronized WCA by structural induction, one needs a stronger statement, namely on tuples of synchronized expressions:

**Theorem 3.4.5**

For all tuples of iterated-sum expressions $\mathbb{E} = (E_1, \ldots, E_n)$ such that $\mathrm{Sync}\,(\mathbb{E})$, one can construct a tuple of WCA $\mathbb{C} = (C_1, \ldots, C_n)$ such that the following condition hold:

1. for all $i \in \{1, \ldots, n\}$, $\mathtt{dom}(C_i) = \bigcap_{j=1}^{n} \mathtt{dom}(E_j)$
2. for all $i \in \{1, \ldots, n\}$, for all $u \in \bigcap_{j=1}^{n} \mathtt{dom}(E_j)$, $[\![E_i]\!](u) = [\![C_i]\!](u)$
3. $\mathrm{Sync}\,(\mathbb{C})$

[XV]In this sense, the definition of synchronization may seem a bit strong, unlike that of synchronization of iterated-sum expressions, but it makes the decidability (and in particular the semi-linearity property Lemma 3.4.2) way easier to prove, because it allows from a product construction, just as in the non-iterated case (monolithic expressions)

**Proof** The proof goes by induction on $\langle\!\langle \mathbb{E} \rangle\!\rangle = \sum_{i=1}^{n} \langle\!\langle E_i \rangle\!\rangle$ where for any iterated-sum expression $E$, we define $\langle\!\langle E \rangle\!\rangle \in \mathbb{N}$ inductively by $\langle\!\langle A \rangle\!\rangle = 0$, $\langle\!\langle E^{\circledast} \rangle\!\rangle = \langle\!\langle E \rangle\!\rangle + 1$, $\langle\!\langle \varphi(E_1, E_2) \rangle\!\rangle = 1 + \langle\!\langle E_1 \rangle\!\rangle + \langle\!\langle E_2 \rangle\!\rangle$.
So, if $\langle\!\langle \mathbb{E} \rangle\!\rangle = 0$, then all expressions in $\mathbb{E}$ are unambiguous sum-automata $A_1, \dots, A_n$, and hence are 0-WCA. However, the conditions in the lemma requires that the domains of all WCA are equal to $D = \bigcap_{i=1}^{n} \mathtt{dom}(A_i)$. It suffices to restrict $A_1, \dots, A_n$ to $D$, which is always possible since unambiguous sum-automata are closed under regular domain restriction, and $D$ is regular. One obtains unambiguous sum-automata $A'_1, \dots, A'_n$, i.e. 0-WCA, which satisfy the requirements since unambiguous sum-automata are always mutually synchronized.
The case $\langle\!\langle \mathbb{E} \rangle\!\rangle > 0$ is more involved and is a disjunction of the following cases.

*First case.* There exists $i$ such that $E_i = \varphi(F_1, F_2)$. Then we define the $(n+1)$-tuple[XVI] $\mathbb{E}' = (E_1, \dots, E_{i-1}, F_1, F_2, E_{i+1}, \dots, E_n)$ which satisfies $\mathrm{Sync}(\mathbb{E}')$ by definition of synchronization for iterated-sum expressions. We also have $\langle\!\langle \mathbb{E}' \rangle\!\rangle < \langle\!\langle \mathbb{E} \rangle\!\rangle$, hence we can apply the induction hypothesis on $\mathbb{E}'$, and obtain a tuple $\mathbb{C}' = (C_1, \dots, C_{i-1}, C_i^1, C_i^2, C_{i+1}, \dots, C_n)$ of WCA such that:

i for all $j \neq i$, $\mathtt{dom}(C_j) = \mathtt{dom}(C_i^1) = \mathtt{dom}(C_i^2) = D'$
ii for all $j \neq i$, $[\![E_j]\!]|_{D'} = [\![C_j]\!]$, $[\![F_1]\!]|_{D'} = [\![C_i^1]\!]$, $[\![F_2]\!]|_{D'} = [\![C_i^2]\!]$
iii $\mathrm{Sync}(\mathbb{C}')$ (in particular all WCA in $\mathbb{C}'$ have the same chop-level $m$)

where $D' = \bigcap_{\alpha \in \mathbb{E}'} \mathtt{dom}(\alpha)$. We return the tuple of WCA

$$\mathbb{C} = (\varphi_{\mathrm{id}}(C_1), \dots, \varphi_{\mathrm{id}}(C_{i-1}), \varphi(C_i^1, C_i^2), \varphi_{\mathrm{id}}(C_{i+1}), \dots, \varphi_{\mathrm{id}}(C_n))$$

where $\varphi_{\mathrm{id}}$ is a Presburger combinator which realizes the identity function, and the Presburger operations on WCA have been defined in Proposition 3.3.5. Note that, we return $\varphi_{\mathrm{id}}(C_j)$ instead of $C_j$ for all $j \neq i$, to force each WCA of $\mathbb{C}$ to have the same chop level $m+1$ and therefore get synchronization of $\mathbb{C}$.

*Second case.* There are only stars in the tuple i.e. $\mathbb{E}$ is of the form $(F_1^{\circledast}, \dots, F_n^{\circledast})$. In this case we apply our induction hypothesis on $(F_1, \dots, F_n)$, which is synchronized since $(F_1^{\circledast}, \dots, F_n^{\circledast})$ is synchronized. Then we obtain a tuple of WCA $(C_1, \dots, C_n)$, and return $(C_1^{\circledast}, \dots, C_n^{\circledast})$ where the operation $^{\circledast}$ on WCA has been defined in Proposition 3.3.5.

*Third case.* Only unambiguous sum-automata and iterated-sum expressions are mixed. Let us explain the construction with only a single automaton and a single star expression, i.e. $\mathbb{E} = (A, F^{\circledast})$. The case where there are arbitrarily many automata and star expressions is more technical but not more difficult, the main difficulties being found in this special case. The difficulty comes from the fact that $A$ and $F$ do not apply at the same level of decomposition, as explained in the overview of the proof: $A$ runs on the whole input word, while $F$ runs on factors of it.

Hence we artificially chop runs of $A$ into run factors on words of $\mathtt{dom}(F)$. We assume w.l.o.g. that $A$ is trim[XVII], and for all states $p_1, p_2$ of $A$, define $A_{p_1,p_2}$ to be $A$ with only initial state $p_1$ and only accepting state $p_2$. Since $A$ is unambiguous and trim, so are the sum-automaton $A_{p_1,p_2}$.

Let $B$ be a DFA accepting the set of words $u$ that are uniquely decomposed into factors in $\mathtt{dom}(F)$ (see the proof of Proposition 3.2.2 for a construction of $B$). For all states $q_1, q_2$ of $B$, let $L_{q_1,q_2}$ be the set of words in $\mathtt{dom}(F)$ such that there exists a run of $B$ from state $q_1$ to state $q_2$. Clearly:

$$\mathtt{dom}(F^{\circledast}) = \bigcup_{m \in \mathbb{N}} \left\{ L_{q_0,q_1} L_{q_1,q_2} \dots L_{q_{m-1},q_m} \;\middle|\; \begin{array}{l} q_0, q_1, \dots, q_m \text{ are states of } B, \\ q_0 \text{ is initial and } q_m \text{ final} \end{array} \right\}$$

Now, for all states $p_1, p_2$ of $A$ and all states $q_1, q_2$ of $B$, define $A_{p_1,p_2,q_1,q_2}$ as the sum-automaton $A_{p_1,p_2}$ restricted to the domain $L_{q_1,q_2}$. It can be assumed to be unambiguous as well, since $A_{p_1,p_2}$ is unambiguous and $B$ is deterministic. Then, apply the induction hypothesis on the pairs $(A_{p_1,p_2,q_1,q_2}, F)$ which is synchronized to get equivalent synchronized WCA $(C^1_{p_1,p_2,q_1,q_2}, C^2_{p_1,p_2,q_1,q_2})$.

We now explain how to combine these WCA into a pair of WCA $(C_1, C_2)$ equivalent to $(A, F^{\circledast})$ on $\mathtt{dom}(A) \cap \mathtt{dom}(F^{\circledast})$. Let $C_{p_1,p_2,q_1,q_2}$ be a generalized WCA defined as the product $C^1_{p_1,p_2,q_1,q_2} \otimes C^2_{p_1,p_2,q_1,q_2}$ (hence with values in $\mathbb{Z}^2$). We now have to combine all these WCA into a single one $C$ running on the whole input word. We construct $C$ by taking the union of all WCA $C_{p,p',q,q'}$, and by "merging", for all states $p_1, p_2, p_3$ of $A$ and all states $q_1, q_2, q_3$ of $B$, the accepting states of $C_{p_1,p_2,q_1,q_2}$ with the initial states of $C_{p_2,p_3,q_2,q_3}$. The merging operation is non-deterministic, because we may need to stay in the automaton $C_{p_1,p_2,q_1,q_2}$ even if we have already seen one of its accepting state: when $C_{p_1,p_2,q_1,q_2}$ triggers a transition to one of its accepting state, it either go to it, or to some initial state of $C_{p_2,p_3,q_2,q_3}$. The initial states of $C$ are the initial states of any $C_{p,p',q,q'}$ such that $p$ is initial in $A$, $q$ is initial in $B$. The accepting states of $C$ are the accepting states of any $C_{p,p',q,q'}$ such that $p'$ is accepting in $A$ and $q'$ is accepting in $B$. That way, we have $\mathtt{dom}(C) = \mathtt{dom}(A) \cap \mathtt{dom}(F^{\circledast})$. The WCA $C_1, C_2$ are obtained from $C$ by projecting the pairs of expressions occurring in $C$ to their first and second component respectively.

Finally, let us sketch how to proceed if there are more than one automaton $A_1, \dots, A_n$ in $\mathbb{E}$ and more than one star expression $F_1^{\circledast}, \dots, F_m^{\circledast}$ in $\mathbb{E}$. The idea is very similar, but we consider an automaton $B$ that accepts all words that are uniquely decomposed according to $\bigcap_j \mathtt{dom}(F_j)$. Since the star expressions are synchronized, they all decompose the input word the same way, making this construction sound. Then, in the sub-weighted automata $A_{p,p',q,q'}$, $p$ and $p'$ are instead tuples of states of each automata $A_j$. ♦

Finally, the announced direct corollary of the previous lemma is:

**Corollary 3.4.6**

Any synchronized expression $E$ is equivalent to some synchronized weighted chop automaton $C_E$, i.e. $[\![E]\!] = [\![C_E]\!]$.

We conjecture that synchronized WCA are strictly more expressive than synchronized expressions. In particular, we conjecture that synchronized expressions are not closed under split sum $\odot$, unlike synchronized WCA as proved in Proposition 3.3.5. The quantitative language of Example 3.3.2 does not seem to be definable by any synchronized expression.

## 3.5 Summary and related works

We investigated generalizations of regular automata as a quantitative formalism that uses the Presburger arithmetic to aggregate values unambiguously summed along runs.

### Related works

Chatterjee et al. have introduced a recursive model of $\mathsf{WA}^{\max}_{\text{sum}}$ [CHO15] in the context of infinite words as input, which is called nested weighted automata (NWA) and has a decidable emptiness and inclusion problems. Using the vocabulary of the authors, a master WCA decomposes the input word and calls a slave WCA on a fragment while, a master NWA calls a slave on an infinite remaining suffix of the input word and then the slave NWA will never halt. So, the two models NWA and WCA bear similarities but we conjecture that their expressiveness is incomparable. On one hand, the NWA formalism differs from WCA since they can define quantitative languages whose ranges are not semi-linear. On the other hand, the NWA are not closed under Presburger definable functions.

Alur et al. have recently introduced a general model of weighted automata for stream processing, which is hierarchical as chop automata and, is parametrized by a set of aggregators [AR13]. They goal was to provide efficient evaluation algorithms for their model. It could be interesting to see if the notion of synchronization could be defined on their model to obtain decidability results with respect to quantitative verification questions, which are left open in [AR13].

In the context of vector addition systems over integers (VASS), Bonnet has shown

[XVI]Note that this case exhibits already the need to consider tuples of expressions rather a single expression, to prove Theorem 3.4.5 inductively.

[XVII]Trim mean that, all its states are accessible from an initial state and co-accessible from a final state. It is well-known that any automaton can be trimmed in polynomial time.

decidability of nested zero-test extension [Bon11]. A VASS is a program that manipulates counters with increments, decrements, and conditional transitions that test the value of some counters and require them to be zero. Consider a total order on counters of VASS inducing a priority level between counters. The nested zero-test is a restriction of zero-test where a counter can be tested for zero on a transition if all the smaller counters are so. Hence, nested zero-tests are of the form $c_k = c_{k-1} \cdots = c_2 = c_1 = 0$ for some $k$. The main distinction with WCA is that VASS does not read input words. However, the nesting of zero-tests share similarities with the chop level, and it is well known that Presburger relations are exactly the class of relations expressible by flat[XVIII] counter automata with increments, resets, positive decrements, and zero-tests [Gur85]. In [HZ19], authors considered a generalization of Presburger formula by introducing the iterated sum operator and thus, provide tight complexity results for VASS with nested zero-tests. This approach seems to be compatible with the WCA formalism and then is a promising starting point to obtain tight complexity results for the emptiness of WCA.

**Future works**

An expression formalism with the unambiguous iterated sum, conditional choice and split sum, whose atoms are constant quantitative languages[XIX], was already introduced in [AFR14] which shown that this expression formalism is equivalent to unambiguous $\mathsf{WA}^{\max}_{\text{sum}}$. Our goal was then to go beyond this expressibility, by having a formalism closed under Presburger combinators. Adding such combinators to the expressions of [AFR14] would immediately yield an undecidable formalism, as a consequence of Theorem 3.2.5. It turns out that extending iterated-sum expressions with split sum $\odot$ and conditional choice $\triangleright$, with a suitable notion of synchronization, gives a formalism equivalent to synchronized WCA. An interesting further direction would be to define a simple notion of synchronization for the extension of [AFR14] with Presburger combinators. More generally, our notion of synchronization is semantical (but decidable). This raises the question of whether another weighted expression formalism with a purely syntactic notion of synchronization could be defined.

Also, Proposition 3.2.6 highlights the fact that WCA, which are unambiguous by definition, are strictly more expressive than finitely ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$. It turns out that WCA can also define quantitative languages which seem to require exponential ambiguity of $\mathsf{WA}^{\max}_{\text{sum}}$, for instance the function $f$ of Example 3.2.1, given by:

$$f(u_1 s_1 \dots u_n s_n) = \sum_{i=1}^{n} \max\{|u_i|_a, |u_i|_b\}$$

As a matter of fact, we conjecture that $\mathsf{WA}^{\max}_{\text{sum}}$ and WCA have incomparable expressive power. We have shown in Proposition 3.1.4 that simple expressions (which are strictly less expressive than WCA) can express quantitative languages which are not definable by $\mathsf{WA}^{\max}_{\text{sum}}$, for instance the function $u \mapsto \min\{|u|_a, |u|_b\}$. We conjecture that the following quantitative language "max prefix", which is trivially definable by some linearly ambiguous $\mathsf{WA}^{\max}_{\text{sum}}$, is not definable by any WCA.

$$u \in \{a,b\}^* \mapsto \max\{|u'|_a - |u'|_b : u' \sqsubseteq u\}$$

Indeed, and intuitively, chop automata decompose the input word unambiguously into independent, non-overlapping factors, which are themselves decomposed recursively. Implementing the function "max prefix" requires to consider linearly many overlapping factors (the prefixes). Of course, this function would be definable by an extension of WCA where the unambiguity condition is dropped, however, this would yield a model strictly more expressive than $\mathsf{WA}^{\max}_{\text{sum}}$ and as a consequence, undecidable for the inclusion problem for instance.

[XVIII]Flat automata cannot have cycles that contain a cycle, i.e. they can have simple cycles only.

[XIX]Any word from a regular language is mapped to a same constant value.

# II

# Robustness

# Chapter 4

# Error models

The model-checking has proved to be successful applications for program verification with respect to a given specification. However, this approach does not provide a measure of confidence about the answer and thus, raises an idealistic technique that is too rigid when the system parameters are prone to errors. For instance, when the specification comes from machine-learning as medical observations or when the input has been performed by a noisy cyber-physical system as signal sensors.

In this chapter, we present a quantitative approach that places a metric on words to provide a natural notion of distance between words. Such a metric leads to a topological neighborhood of words and languages that we use to model the cost of errors and which allows us to challenge the systems on its robustness. In particular, we assume that the cost of rewriting a word back to itself is 0. Hence, a language is said to be robust if the language membership cannot differ for two "close" words, and that leads to robust versions of all the classical decision problems as language inclusion for instance.

Transducers is a well know formalism that generalizes regular automata to define word-to-word relations [Ber79]. In the transducer formalism, we are able to model various error situations depending on the (input, output) label of a transition, a letter insertion $(\varepsilon, a)$, a deletion $(a, \varepsilon)$ and a mutation $(a, b)$. Our contribution is to study robustness verification problems in the context of weighted transducers which assign a non-negative rational cost to each pair of words $(u_1, u_2)$ to model the cost of rewriting $u_1$ into $u_2$. More precisely, given a distance $d\colon \Sigma^* \times \Sigma^* \to \mathbb{Q}$ defined by a weighted-transducer and a threshold $\nu \in \mathbb{Q}_{\geq 0}$, the $\nu$-neighborhood of a regular language $N$ is defined as $N_\nu = \{u' : \exists u \in N\ \mathsf{C}(u, u') \leq \nu\}$. Hence, the *robust inclusion* problem $N_\nu \subseteq L$ relax the classical inclusion $N \subseteq L$ up to an error $\nu$. We also investigate the threshold synthesis that consists of computing the largest $\nu \in \mathbb{Q}_{\geq 0}$ for which the robust inclusion holds.

To denote such distance functions, we define the weighted transducers together with an *aggregator* that combines the cost of each individual rewriting of the transducer into an overall cost between the input and output words. Taking the minimal number of letter transformations insertion, deletion and mutation, (i.e. the minimal sum with cost 1 for all of these transformations, defines the Levenshtein (a.k.a. edit) distance [Moh02]. Aggregating the costs with a discounted sum allows us to make the costs of rewrites dependent on the positions where the rewrites take place. And finally, the average is often used when the length of rewrites does not matter. We provide algorithms to decide the robust inclusion problem for these three measures (sum, mean, and discounted sum). Furthermore, we provide case studies including modeling human control failures and approximate recognition of type-1 diabetes using robust detection of blood sugar level swings.

## 4.1 Robust verification framework

Weighted transducers extend regular automata with both string outputs and weights on transitions [DKV09]. Automata with string outputs and automata with integer outputs have been defined in Section 1.4. For weighted transducers, any accepting run over some input word rewrites each input symbol into a (possibly empty) word, with some cost in $\mathbb{N}$. Transducers can also have $\varepsilon$-input transitions with non-empty outputs, such that output symbols can be produced even though nothing is read on the input, e.g. allowing for symbol insertions. The output of a run is the concatenation of all output words occurring on its transitions. Its cost is obtained by applying an *aggregator function* $\mathsf{C}\colon \mathbb{N}^* \to \mathbb{Q}_{\geq 0}$, which maps a sequence of naturals[I] to a rational number. We consider three different aggregator functions, given later. Since there are possibly several accept-

[I] Note carefully that $\mathbb{N}^*$ refer to the set of words from the infinite alphabet $\mathbb{N}$.

ing runs over the same input, and generating the same output, we take the minimal cost of them to compute the value of a pair of input and output words.

**Definition – $\mathsf{C}$-transducers**

> Let $\mathsf{C}\colon \mathbb{N}^* \to \mathbb{Q}_{\geq 0}$ be an aggregator. A $\mathsf{C}$-transducer $T$ is a tuple $(A, \mathtt{W})$ where $A = (Q, Q_I, Q_F, \Delta, \gamma)$ is a transducer for which $\varepsilon$-transition cannot output $\varepsilon$ and $\mathtt{W}\colon \Delta \to \mathbb{N}$ associates weights to each transition.

Consider a $\mathsf{C}$-transducer $T = (A, \mathtt{W})$. As defined in Section 1.4, $\mathtt{AccRun}_A(u_1, u_2)$ denotes the set of accepting runs of $A$ on the input $u_1$ and the output $u_2$. The transducer $T$ has the same set of accenting runs that we refer as $\mathtt{AccRun}_T(u_1, u_2)$. Given $\varrho \in \mathtt{AccRun}_T(u_1, u_2)$ such that $\varrho = t_1 \dots t_n \in \Delta^*$ of $A$ is associated to the aggregated cost $\mathsf{C}(\varrho)$ defined by $\mathsf{C}(\mathtt{W}(\varrho))$ with the weighted sequence $\mathtt{W}(\varrho) = \mathtt{W}(t_1) \dots \mathtt{W}(t_n)$. The input/output relation $[\![T]\!]$ is defined as $\{(u_1, u_2) : \mathtt{AccRun}_T(u_1, u_2) \neq \varnothing\}$. The domain and the range of $T$ are respectively $\mathtt{dom}(T) = \{u_1 : (u_1, u_2) \in [\![T]\!]\}$ and $R(T) = \{u_2 : (u_1, u_2) \in [\![T]\!]\}$. The cost associated by $T$ from a pair of words $(u_1, u_2)$ is given by:

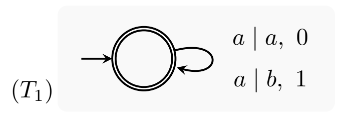
$$\mathsf{C}_T(u_1, u_2) = \begin{cases} +\infty & \text{if } \mathtt{AccRun}_T(u_1, u_2) = \varnothing \\ \min\{\mathsf{C}(\varrho) : \varrho \in \mathtt{AccRun}_T(u_1, u_2)\} & \text{otherwise} \end{cases}$$

Since runs of $T$ consume at least one symbol of the input or one of the output on every transition, there are finitely many runs on a pair $(u_1, u_2)$, hence the min is well-defined. We assume that our $\mathsf{C}$-transducers $T$ satisfy the condition that for all $u \in \mathtt{dom}(T)$, $\mathsf{C}_T(u, u) = 0$ and so, in particular $(u, u) \in [\![T]\!]$. In other words, it is always possible to rewrite $u$ into itself at zero cost. Finally, given $\nu \in \mathbb{Q}$ and an input word $u_1 \in \mathtt{dom}(T)$, we define the threshold output language of $u_1$ as $T_{\leq \nu}(u_1) = \{u_2 : \mathsf{C}_T(u_1, u_2) \leq \nu\}$. This notation extends naturally to languages $N \subseteq \Sigma^*$ by setting $T_{\leq \nu}(N) = \bigcup_{u_1 \in N \cap \mathtt{dom}(T)} T_{\leq \nu}(u_1)$.

We consider three aggregator functions, namely the sum, the mean and the discounted-sum. Let $\lambda \in \mathbb{Q} \cap (0, 1)$ be a discount factor. Given a sequence of weights $w = w_1 \dots w_n$, those three functions are defined by:

$$\mathtt{Sum}(w) = \sum_{i=1}^{n} w_i \qquad \mathtt{Mean}(w) = \begin{cases} 0 & \text{if } w = \varepsilon \\ \frac{\mathtt{Sum}(w)}{n} & \text{otherwise} \end{cases} \qquad \mathtt{DSum}(w) = \sum_{i=1}^{n} \lambda^{(i-1)} w_i$$

Note that, by ignoring the word outputs of a $\mathtt{Sum}$-transducer and keeping only the weight outputs raise a definition that coincide with $\mathsf{WA}^{\min}_{\text{sum}}$ of Section 1.4, i.e. weighted automata over the semi-ring $(\mathbb{N} \cup \{+\infty\}, \min, +)$.



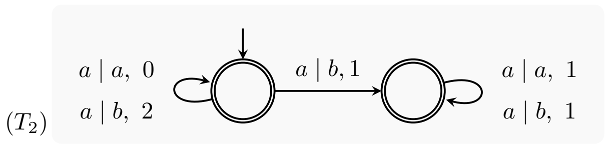


Figure 4.1: Examples of weighted transducers of domain $\{a\}^*$ over the alphabet $\{a, b\}$.

The weighted transducer $T_1$ on Figure 4.1 has exactly one accepting run on any input/output pair of words. As a straightforward application of the definitions, we have that $\mathtt{Sum}_T(aaaa, baab) = 2$, $\mathtt{Mean}_T(aaaa, baab) = \frac{1}{2}$ and $\mathtt{DSum}_T(aaaa, baab) = \lambda^0 + \lambda^3$ for any discount factor $\lambda \in \mathbb{Q} \cap (0, 1)$. In contrast, the weighted transducer $T_2$ on Figure 4.1 admits a polynomial number of accepting runs on the length of the input/output pair of words. Basically, achieving the minimal cost consists of determining the position in the input word when it becomes beneficial to always aggregate 1 instead of a non-equitable share depending on the output letter. For all $u \in \{a, b\}^*$, we have that $\mathtt{Sum}_T(a^{|u|}, u) = \min\{2|v_1|_b + |v_2| : u = v_1 v_2\}$ where $|v_1|_b$ denotes the number of occurrences of $b$ in $v_1$ and $\mathtt{Mean}_T(a^{|u|}, u) = \frac{\mathtt{Sum}_T(a^{|u|}, u)}{|u|}$. Hence, $\mathtt{Sum}_T(aaaa, baab) = 3$ and $\mathtt{Mean}_T(aaaa, baab) = \frac{3}{4}$. For the discounted sum aggregator, the minimal cost run also depends on the discount factor $\lambda \in \mathbb{Q} \cap (0, 1)$, we have that $\mathtt{DSum}_T(a^{|u|}, u) = \min\{\sum_{i=1}^{|v_1|} x_i \lambda^{i-1} + \frac{\lambda^{|v_1|} - \lambda^{|u|+1}}{1 - \lambda} : u = v_1 v_2 \wedge u[i] = b \Rightarrow x_i = 1 \wedge u[i] = a \Rightarrow x_i = 0\}$. In particular, $\mathtt{DSum}_T(aaaa, baab) = \min\{2 + \lambda^3, \frac{1 - \lambda^4}{1 - \lambda}\}$.

**Robustness problems**

We study the following three fundamental problems related to robustness for three different aggregator functions $\mathsf{C} \in \{\mathtt{Sum}, \mathtt{Mean}, \mathtt{DSum}\}$. Given a threshold $\nu \in \mathbb{Q}$, a $\mathsf{C}$-transducer $T$ and a regular language $L$, a word $u \in \mathtt{dom}(T)$ is said to be *$\nu$-robust*, or just robust if $\nu$ is clear from the context, if $T_{\leq\nu}(u) \subseteq L$. In other words, all its rewritings of cost $\nu$ at most are in $L$. A language $N \subseteq \Sigma^*$ is said to be $\nu$-robust if $N \cap \mathtt{dom}(T)$ contains only $\nu$-robust words. Finally, the *$\nu$-robust kernel* of $T$ is the set $\mathtt{Rob}_T(\nu, L)$ of $\nu$-robust words: $\mathtt{Rob}_T(\nu, L) = \{u \in \mathtt{dom}(T) : T_{\leq\nu}(u) \subseteq L\}$

The following Proposition states that bigger is the error threshold, coarser is the robust kernel.

**Proposition 4.1.1**

Given $\nu, \nu' \in \mathbb{Q}_{\geq 0}$ and a $\mathsf{C}$-transducer $T$ and a regular language $L$, we have that $\nu' \leq \nu \implies \mathtt{Rob}_T(\nu', L) \subseteq \mathtt{Rob}_T(\nu, L)$.

**Proof** By definition $T_{\leq\nu}(u_1) = \{u_2 : \mathsf{C}_T(u_1, u_2) \leq \nu\}$. For all $u_1 \in \mathtt{dom}(T)$ we have that $u_1 \in \mathtt{Rob}_T(\nu, L)$ iff for all $u_2$ both $u_2 \in L$ and $\mathsf{C}_T(u_1, u_2) \leq \nu$ hold. Clearly $u_1 \in \mathtt{Rob}_T(\nu, L)$ implies $u_1 \in \mathtt{Rob}_T(\nu', L)$ for any $\nu' \leq \nu$. ♦

We are in a position to formally define the three key problems studied in this chapter. For these definitions, we let $\mathsf{C} \in \{\mathtt{Sum}, \mathtt{Mean}, \mathtt{DSum}\}$.

**Problem 4.1.2 – Robust Inclusion**

Given a $\mathsf{C}$-transducer $T$, a regular language $N \subseteq \Sigma^*$ as an $\mathsf{NFA}$, a threshold $\nu \in \mathbb{Q}_{\geq 0}$ and a language $L \subseteq \Sigma^*$ as a $\mathsf{DFA}$, the robust inclusion problem is to decide whether $N \subseteq \mathtt{Rob}_T(\nu, L)$, i.e. whether $T_{\leq\nu}(N) \subseteq L$.

**Problem 4.1.3 – Threshold synthesis**

Given a $\mathsf{C}$-transducer $T$, a regular language $N \subseteq \Sigma^*$ as an $\mathsf{NFA}$, and a regular language $L \subseteq \Sigma^*$ as a $\mathsf{DFA}$, the threshold synthesis problem is to output a partition of the set of thresholds $\mathbb{Q}_{\geq 0} = G \uplus B$ into sets $G$ and $B$ of *good* and *bad* thresholds, i.e. $G = \{\nu \in \mathbb{Q}_{\geq 0} : N \subseteq \mathtt{Rob}_T(\nu, L)\}$ and $B = \{\nu \in \mathbb{Q}_{\geq 0} : N \not\subseteq \mathtt{Rob}_T(\nu, L)\}$.

As direct consequence of Proposition 4.1.1, the sets $G$ and $B$ are intervals of values, that is for all $\nu_1, \nu_2 \in \mathbb{Q}_{\geq 0}$, if $\nu_1 < \nu_2$ and $\nu_2 \in G$, then $\nu_1 \in G$, and if $\nu_1 \in B$ then $\nu_2 \in B$. Also note that one of the two sets may be empty, in which case the other is the entire set $\mathbb{Q}_{\geq 0}$.

## 4.2 Robustness toward shortest path problem

In this section, given an instance of the threshold synthesis problem, we show how to compute the interval of good thresholds $G$ and the interval of bad thresholds $B$ in PTime for $\mathtt{Sum}, \mathtt{Mean}, \mathtt{DSum}$ measures. As a corollary, we show that the robust inclusion problem is in PTime for all the three measures we consider.

In the following, we assume that $N = \mathtt{dom}(T)$. This is w.l.o.g. as transducers are closed under regular domain restriction thanks to a product construction of $T$ with the automaton for $N$ constructible in polynomial time. With this assumption, the set of good threshold $G$ becomes $G = \{\nu \in \mathbb{Q}_{\geq 0} : \mathtt{dom}(T) \subseteq \mathtt{Rob}_T(\nu, L)\}$ and dually for the set of bad thresholds $B$. We let $\nu_{T,L}$ be the infimum of the set of bad thresholds, i.e. $\nu_{T,L} = \inf B = \inf\{\nu \in \mathbb{Q}_{\geq 0} : \mathtt{dom}(T) \not\subseteq \mathtt{Rob}_T(\nu, L)\}$. As illustrated by the following example, computing $\nu_{T,L}$ allows us to compute $B$ and $G$.

**Example**

Let $\Sigma = \{a, b, c\}$ and $\mathsf{C} \in \{\mathtt{Mean}, \mathtt{DSum}\}$. Consider the best threshold problem for $T$ the $\mathsf{C}$-transducer of Figure 4.2, $N = \mathtt{dom}(T) = \{a\}^*$ and $L = \{a\}^* \cup \{b\}^*$. Note that the translations accepted by OK and ID belong to $L$. On the contrary, translations accepted by KO do not belong to $L$ and so they are not robust with respect to $L$ for any threshold. For $\mathtt{Mean}$ measure, the cost of a translation into

$\{c\}^*$ is exactly 1 while the one into $\{b\}^*$ range over $[0, 1)$. Hence $\nu_{T,L}^{\texttt{Mean}} = 1$ and the set partition of good and bad thresholds is $G^{\texttt{Mean}} = [0, 1)$ and $B^{\texttt{Mean}} = [1, +\infty)$. In the case of `DSum` with discount factor 0.5, the cost of a translation into $\{c\}^*$ range over $[2, 2.5)$ while the one into $\{b\}^*$ range over $[0, 2)$. So $\nu_{T,L}^{\texttt{DSum}} = 2$ and the thresholds are partitioned by $G^{\texttt{DSum}} = [0, 2)$ and $B^{\texttt{DSum}} = [2, +\infty)$.

$(T)$

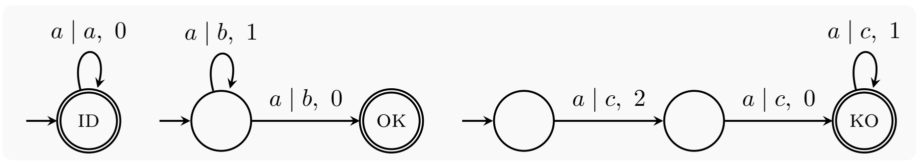


Figure 4.2: Transducer $T$ for which the infimums $\nu_{T,L}^{\texttt{Mean}} = 1$ and $\nu_{T,L}^{\texttt{DSum}} = 2$ are bad thresholds for $T$ interpreted as `Mean`- and `DSum`-transducer with discount factor $\frac{1}{2}$ respectively, and for $L = \{a\}^* \cup \{b\}^*$.

Then, we associate with every transducer $T$ and property $L$ given by some DFA $A$, a transition system with natural weights as defined in Section 1.4 and we denote it by $G_{T,A}$. Intuitively, $G_{T,A}$ is obtained by first taking the synchronized product of $T$ and $A$ (where $A$ is simulated on the outputs of $T$) and then by projecting this product on the inputs.

Formally, given $T = (Q, Q_I, Q_F, \Delta, \texttt{W})$ and $A = (P, p_I, P_F, \nabla)$, the associated transition system is the synchronized product $G_{T,A} = (V, E, V_I, V_F)$ weighted by $\texttt{W}' \colon E \to \mathbb{N}$ defined such that:

- $V = Q \times P$
- $E$ is the set of transitions $e = (q, p) \to (q', p')$ such that there exists $a \in \Sigma \cup \{\varepsilon\}$ and a transition $t = (q, a, u, q') \in \Delta$ such that $p' = \nabla(p, u)$ where $\nabla$ has been extended to words in the expected way. We say that $e$ is compatible with $t$.
- For all $e \in E$, $\texttt{W}'(e) = \min\{\texttt{W}(t) : e \text{ is compatible with } t\}$.
- $V_I = Q_I \times \{p_I\}$ is the set of initial states.
- $V_F = Q_F \times (P \setminus P_F)$ is the set of final states.

Additionally, given a run $\varrho$ in this transition system as a sequence of transitions $e_1 \dots e_n$, we let $\mathsf{C}(\varrho) = \mathsf{C}(\texttt{W}'(e_1) \dots \texttt{W}'(e_n))$ be its weight.

The following lemma establishes some connection between $\nu_{T,L}$ and the runs of $G_{T,A}$.

**Lemma 4.2.1**

Let $G_{T,A}$ be the weighted transition system associated to the given $\mathsf{C}$-transducer $T$ and the given regular automaton $A$. The infimum cost of accepting runs is equal to $\nu_{T,L}$, i.e. $\nu_{T,L} = \inf\{\mathsf{C}(\varrho) : \varrho \in \texttt{AccRun}_{G_{T,A}}\}$.

**Proof** We first show that any accepting run $\varrho$ satisfies $\mathsf{C}(\varrho) \geq \nu_{T,L}$. By construction of $G_{T,A}$, there exists an input word $u_1 \in \texttt{dom}(T)$, some output word $u_2 \notin L$ and an accepting run $r$ of $T$ on $(u_1, u_2)$ of value $\mathsf{C}(r) = \mathsf{C}(\varrho)$. Since the value $\mathsf{C}_T(u_1, u_2)$ is the minimal value of all accepting runs of $T$ overs $(u_1, u_2)$, we have $\mathsf{C}(r) \geq \mathsf{C}_T(u_1, u_2)$ and $u_1$ is not robust for threshold $\mathsf{C}_T(u_1, u_2)$, *a fortiori* for threshold $\mathsf{C}(r)$, from which we get $\mathsf{C}(r) = \mathsf{C}(\varrho) \geq \nu_{T,L}$. This shows that $\nu_{T,L} \leq \inf\{\mathsf{C}(\varrho) : \varrho \in \texttt{AccRun}_{G_{T,A}}\}$.
Suppose that $\nu_{T,L}$ is strictly smaller than this infinimum (that we denote $m$) and take some rational number $\nu$ such that $\nu_{T,L} < \nu < m$. Since $\nu_{T,L} < \nu$, it is a bad threshold which means that there exists $u_1 \in \texttt{dom}(T)$ such that $u_1 \notin \texttt{Rob}_T(\nu, L)$. Hence there exists $u_2 \notin L$ such that $\mathsf{C}_T(u_1, u_2) \leq \nu$, and by definition of $G_{T,A}$, there exists an accepting run $\varrho$ of value $\mathsf{C}(\varrho) \leq \nu$. This contradicts the fact that $\nu < m$ by definition of $m$. Hence, $\nu_{T,L} = m$, concluding the proof. ⧫

The next lemma establishes that the infimum of values of accepting runs in a weighted transition system can be computed in PTIME and it is also decidable in PTIME if the infimum is realized, for all the three measures considered in this chapter. As a direct corollary of this lemma we obtain the main theorem of the section.

**Lemma 4.2.2**

For a transition system $G = (V, E, V_I, V_F)$ weighted by $\mathtt{W}\colon E \to \mathbb{Q}_{\geq 0}$, the infimum amongst the value of accepting runs can be computed in PTime for all $\mathsf{C} \in \{\mathtt{Sum}, \mathtt{DSum}, \mathtt{Mean}\}$. Moreover, we can decide in PTime if this infimum is realizable.

**Proof** We first trim the transition system $G$ by removing all the states that cannot be reached from an initial state or that cannot reach a final state as those states cannot participate to run acceptance. The set of accepting runs is empty iff the trimmed transition system is empty and then the infimum is equal to $+\infty$. Now, we assume the trimmed transition system to be non-empty, i.e. there is at least one accepting run.
We now consider the three measures in turn. For `Sum`, the infimum amongst the value of accepting runs in the transition system $G$ with non-negative weights can be computed in PTime using Dijkstra shortest-path algorithm and it is always realizable by a cycle-free accepting run.
For `Mean`, we first note that the infimum is either realized by a cycle-free accepting run or comes from the value of a reachable cycle from an initial state that can be iterated arbitrarily many times before moving to some accepting state. In the latter case, if $c$ is a cycle of `Mean` $m$ which is smaller than the `Mean` value of any accepting run then the family of runs $\varrho_k = \varrho \cdot c^k \cdot \varrho'$, where $\varrho$ is a cycle-free run from some initial state to $c$ and $\varrho'$ is a cycle-free run from $c$ to some final state[II]and such that $\lim_{k \mapsto +\infty} \mathtt{Mean}(\varrho_k) = \mathtt{Mean}(c)$ and $\mathtt{Mean}(c)$ is the infimum. Now if all the cycles have a value larger than the infimum, they are not beneficial as those cycles can be systematically removed and give runs with smaller values. Note that the minimum `Mean` values amongst cycle-free accepting runs can be computed in PTime by a simple dynamic program that considers the minimal values of runs of lengths at most equal to the number of states. Moreover, the minimum mean value of cycles in the trimmed transition system can be computed in PTime using the algorithm of [Kar78]. It is easy to see that if the infimum is realizable iff it equals the minimum `Mean` value of cycle-free runs.
For `DSum`, Theorem 1 of [AKTY13] tells us that for all $v \in V$ we can compute the infimum value $x_v$ of runs from $v$ to some final state in PTime. According to Lemma 1 of [AKTY13], and similarly to the case of `Mean`, the infimum `DSum` value $x_v$ is either realized by a cycle-free run or by a family of runs of the form $\varrho \cdot c^k \cdot \varrho'$. This is because if it is beneficial to include a cycle $c$ to reduce the cost of a run from $v$ to some final state then it is beneficial to iterate this cycle arbitrarily many times. In particular, the infimum is realizable only when there exists not such beneficial cycle. In order to decide the realizability of the values $x_v$ for all $v \in V$, we consider $G'$ as the transition system $G$ where we keep only those transitions $e = (v, v')$ such that the optimal value $x_v$ to reach some final state from $v$ can be realized through the state $v'$. Let $\lambda$ be the discount factor. Formally, we construct the transition system $G' = (V, E', V_I, V_F)$ with $E' \subseteq E$ and such that $(v, v') \in E'$ if $x_v = \lambda x_{v'} + \mathtt{W}(v, v')$. We claim that, $V_F$ is reachable from $v$ in $G'$ iff $x_v$ is realizable in $G$ from $v$, hence testing realizability boils down to checking the existence of a run in $G'$.
The left-to-right implication comes by induction on the length of the run $\varrho$ from $v$ to some final state. If $v_I \in V_F$ then $|\varrho| = 0$ and $x_{v_I, v_F} = 0$ which is realizable. Assume $v_I \notin V_F$ and $\varrho = (v, v')\varrho'$. By induction hypothesis, $x_{v'}$ is realizable by some run $\varrho''$ from $v'$ to $V_F$. By construction of $G'$ we have $x_v = \lambda x_{v'} + \mathtt{W}(v, v')$. Hence $x_v$ is realizable by $(v, v')\varrho''$.
For the right-to-left implication, if $v \in V_F$ it is trivial, so assume that $v \notin V_F$ and let $\varrho = (v, v')\varrho'$ a run that realizes $x_v$. Assume $x_v > \lambda x_{v'} + \mathtt{W}(v, v')$. This contradicts the optimality of $x_v$, as $\varrho$ witnesses a better discounted value from $v$ to $V_F$. Assume $x_v < \lambda x_{v'} + \mathtt{W}(v, v')$, then since $\varrho$ realizes $x_v$, we have $x_v = \mathtt{W}(v, v') + \lambda\mathtt{DSum}(\varrho')$. It implies $\mathtt{DSum}(\varrho') < x_{v'}$. This contradicts the minimality of $x_{v'}$, as then $\varrho'$ witnesses a better value for runs from $v'$ to $V_F$. Hence $x_v = \lambda x_{v'} + \mathtt{W}(v, v')$ and $(v, v')$ is an transition of $G'$. By induction on the length of $\varrho$, we can also conclude that $\varrho'$ is a run of $G'$ and then $\varrho$ is a run of $G'$ from $v$ to $V_F$. ♦

[II]Such cycle-free run exist as the transition system is trimmed.

**Theorem 4.2.3**

For a given C-transducer $T$, a language $N \subseteq \Sigma^*$ given as an NFA and $L \subseteq \Sigma^*$ given as a DFA, the set partition of good and bad thresholds $(G, B)$ for $\mathsf{C} \in \{\mathtt{Sum}, \mathtt{DSum}, \mathtt{Mean}\}$ can be computed in PTime.

**Proof** First, we restrict the domain of $T$ to $N$ by taking the product of $T$ and the automaton for $N$ (simulated over the input of $T$). Then we can compute in PTime the value $\nu_{T,L}$ thanks to Lemma 4.2.2. This value is the infimum of $B$ according to Lemma 4.2.1. If this infimum is realizable then the interval $B$ is left closed and equal to $[\nu_{T,L}, +\infty)$ while $G = [0, \nu_{T,L})$, and on the contrary, if this infimum is not realizable, then $B$ is left open and equal to $(\nu_{T,L}, +\infty)$, while $G = [0, \nu_{T,L}]$. Note that when $\nu_{T,L} = 0$ and is realizable, then $G = [0, 0) = \varnothing$. ♦

As a direct consequence, the robust inclusion problem for a threshold $\nu$ can be solved by checking if $\nu \in G$, and so we have the following corollary.

**Corollary 4.2.4**

Let $\mathsf{C} \in \{\mathtt{Sum}, \mathtt{DSum}, \mathtt{Mean}\}$. Given $T$ a C-transducer, $N \subseteq \Sigma^*$ given as an NFA, $L \subseteq \Sigma^*$ given as a DFA and $\nu \in \mathbb{Q}$. The language inclusion $N \subseteq \mathtt{Rob}_T(\nu, L)$ can be decided in PTime.

## 4.3 Case Studies

Some experiment part on robustness have been investigate by peoples from University of Colorado Boulder, thanks to a collaboration. This section describes the implementation of the ideas shown thus far and their application to a detailed case studies. Sankaranarayanan and Trivedi have implemented in Python the threshold synthesis problem 4.1.3 for the discounted and average costs. This implementation supports the specification of a language $L$ specified as a NFA, a weighted transducer $T$ and a property $P$ specified as some DFA.

### 4.3.1 Robustness of Human Control Strategies

An industrial motor operates under many gears $g_1, \ldots, g_5$. Under fault, the human operator must take control of the machine and achieve the following: If the system goes into a fault the operator must ensure that (i) the system is immediately set in gears $3-5$. Subsequently, for the next 5 cycles (ii) it must never go to gear $g_1$ or $g_2$. And (iii) must shift and stay at a higher gear $g_4$ or $g_5$ after the $5^{th}$ cycle until the fault is resolved.

(P)

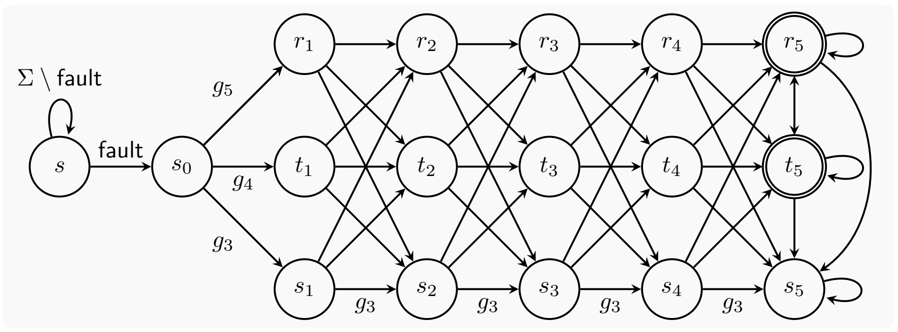


Figure 4.3: Finite state automaton $P$ showing a desired property for the automatic transmission system. All incoming edges to $s_1, \ldots, s_5$ have label $g_3$, incoming edges to $t_1, \ldots, t_5$ have label $g_4$ and $r_1, \ldots, r_5$ have incoming edges labeled $g_5$. All edges not shown lead to a rejecting sink state.

Figure 4.3 shows a finite state machine $P$ that accepts all words satisfying this property: fault is not in the operator's control but $g_1, \ldots, g_5$ are operator actions. Consider that the operator can perform this task in two different ways: $\sigma_1$: fault $g_4\ g_4\ g_4\ g_4\ g_5\ g_5$ versus $\sigma_2$: fault $g_3\ g_3\ g_3\ g_3\ g_3\ g_4$. The input $\sigma_1$ induces the run $s, s_0, t_1, t_2, t_3, r_4, r_5$ whereas the input $\sigma_2$ induces the run $s, s_0, s_1, s_2, s_3, s_4, t_5$. Both $\sigma_1, \sigma_2$ satisfy the property of interest and as such there is nothing to choose one over the other. Suppose the human operator can make mistakes, especially since they are under stress. We will consider that the operator can substitute a command for gear $g_i$ with $g_{i-1}$ (for $i > 1$) or $g_{i+1}$ (for $i < 5$). We use a weighted transducer $T_0$ shown in Figure 4.4 to model these substitutions. The transducer defines possible ways in which a string $\sigma$ can be converted to $\sigma'$ with a notion of cost for the conversion. This example consider two notions of cost, the `DSum`-cost and the `Mean`-cost. These costs now allow us to compare $\sigma_1$ versus $\sigma_2$. For instance, under both notions, it turn out that $\sigma_1$ is much more robust than $\sigma_2$. The robustness of $\sigma_1$ under both cost models is $\infty$ since any change to $\sigma_1$ under the transducer continues to satisfy the desired property. On the other hand $\sigma_2$ has a finite robustness, since operator mistakes can cause violations.

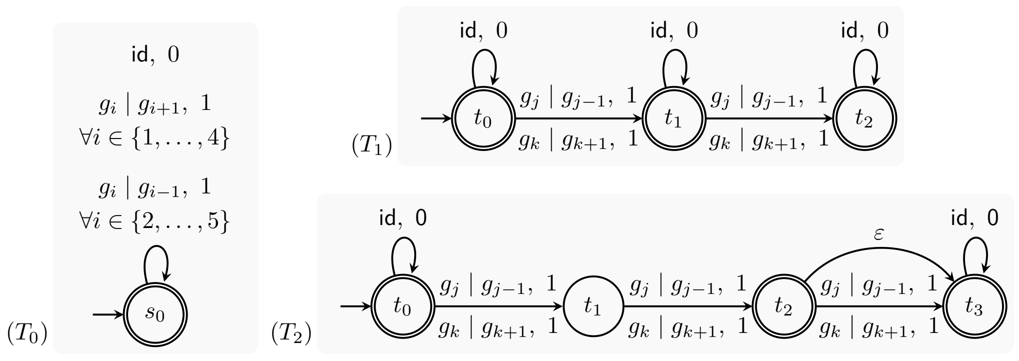


Figure 4.4: Transducers modeling potential human operator mistakes along with their costs: $T_0$ allows arbitrarily many mistakes whereas $T_1$ restricts the number of mistakes to at most 2, whereas $T_2$ models a "bursty" set of mistakes. The edge $a \mid b, w$ denotes a replacement of the letter $a$ by $b$ with a cost $w$. For convenience $T_2$ uses an $\varepsilon$ transition that can be removed.

| String | $T_0$ | | | $T_1$ | | | $T_2$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | Disc. | Avg. | Time | Disc. | Avg. | Time | Disc. | Avg. | Time |
| $g_4^4 g_5^2$ | $\infty$ | $\infty$ | $\varepsilon$ | $\infty$ | $\infty$ | $\varepsilon$ | $\infty$ | $\infty$ | $\varepsilon$ |
| $g_3^3 g_4^3$ | $2^{-5}$ | $\frac{1}{6}$ | 0.03 | $2^{-5}$ | $\frac{1}{6}$ | 0.03 | $\frac{7}{32}$ | $\frac{1}{2}$ | 0.03 |
| $g_3 g_4^3 g_5 g_4^3 g_3 g_4$ | 0 | 0 | 0.04 | 0 | 0 | 0.06 | 0 | 0 | 0.06 |
| $g_3^{10} g_4^{10}$ | 0 | 0 | 0.07 | 0 | 0 | 0.09 | 0 | 0 | 0.1 |
| $g_3^5 g_4^{15} g_5^5 g_4^3 g_5$ | 7.45e-9 | 0.035 | 0.12 | 7.45e-9 | 0.035 | 0.2 | 2.6e-8 | 0.103 | 0.2 |
| $g_3^4 g_4^{25} g_5^{25}$ | 3.7e-9 | 0.019 | 0.15 | 3.73e-9 | 0.019 | 0.4 | 6.52e-9 | 0.056 | 0.3 |

Table 4.1: Running times and robustness values computed by Sankaranarayanan and Trivedi for various input strings (the first letter fault is common to all the strings and is omitted). All timings are measured in seconds, $\varepsilon$ denotes time $< 0.01$ seconds.

The use of a transducer allows for a richer specification of errors. For instance, transducer $T_2$ in Figure 4.4 shows a model of bounded number of mistakes that assume that the operator makes at most 2 mistakes whereas $T_3$ in Figure 4.4 shows a model with "bursty" mistakes that assume that mistakes occur in bursts of at least 2 but at most 3 mistakes at a time. These models are useful in capturing fine grained assumptions about errors that are often the case in the study of human error or errors in physical systems.

Using the prototype implementation, Sankaranarayanan and Trivedi report on the robustness of various inputs for this motivating example under the three transducer error models. The property $P$ is as shown in Figure 4.3 and the transducers $T_0 - T_2$ are as shown in Figure 4.4. Table 4.1 reports the robustness values for various input strings and the running time. Note that while our approach takes about 0.3 seconds for a string of length 50, the prototype can be made much more efficient to reduce the time to compute robustness. Furthermore, discounted sum becomes smaller as the strings grow larger while the average robustness value does not. Hence, average robustness is a more useful measure due to this property in this particular example.

### 4.3.2 Monitoring properties for patients with type-1 diabetes

We will now apply our ideas to the approximate pattern recognition problem for analyzing clinical data for patients with type-1 diabetes. People with type-1 diabetes are required to monitor their blood glucose levels periodically using devices such as continuous glucose monitors (CGMs) that are calibrated multiple times each day using a finger stick blood glucose meter. Data from CGMs is uploaded on-line and available for review by clinicians during periodic doctor visits. Many applications such as Medtronic Carelink(tm) support the automatic upload and visualization of this data by clinicians.

Physicians are commonly interested in knowing about hypoglycemic episodes (defined as blood glucose $\leq 70$ mg/dl) suffered by the patient especially during night times, about prolonged extreme hyperglycemic events that can lead to diabetic ketacidosis (defined as blood glucose $\geq 300$ mg/dl) and about instances of "rebound" hyperglycemia wherein the blood glucose levels swing from hypoglycemia to extreme hyperglycemia within a relatively short time window ($\leq 2$ hrs). We can write queries to retrieve data corresponding the following patterns:

- *Prolonged Hypoglycemia (P1):* Do the blood glucose levels stay below 70 mg/dl (hypoglycemia) for more than 3 hours continuously?[III]
- *Prolonged Hyperglycemia (P2):* Do the blood glucose levels remain above 300 mg/dl (hyperglycemia) for more than 3 hours continuously?[IV]
- *Rebound Hyperglycemia (P3):* Do the blood glucose levels go below 70 mg/dl and then rise rapidly up to 300 mg/dl or higher within 2 hours?[V]

Note that these patterns specify bad events that should not happen. A straightforward and strict pattern recognition approach based on specifying the properties above will hide potentially bad scenarios that nearly match the desired pattern for two main reasons:

- The CGM can be noisy and inaccurate. Its accuracy depends on many factors. First, CGMs are designed to be more accurate for glucose levels near the hypoglycemia limit, since hypoglycemia is much more dangerous than hyperglycemia. Also accuracy improves when the CGM is calibrated using a finger stick measurement but then degrades over time. Moreover the CGMs can "dropout" periodically giving out a very small value and resume working normally.
- The properties above are specified using cutoffs such as 70 mg/dl and 3 hours that are somewhat arbitrary with different opinions among physicians. For instance, a clinician will consider a scenario wherein the patient's blood glucose levels stays at 75 mg/dl for 2.75 hours as a serious case of prolonged hypoglycemia even though such a scenario would not satisfy the property P1.

We propose to solve the approximate pattern recognition problem using the techniques developed in this chapter. Here given a string $w$, a transducer $T$ and a language $L$, we are looking for a word $w'$ such that $w' \in L$ and $C_T(w', w)$ is *as small as possible.* In other words, we wish to solve the threshold synthesis problem 4.1.3 for a language $L$ that is the complement of $\mathsf{P1}$ ($\mathsf{P2}$ or $\mathsf{P3}$).

We partition the range of CGM outputs $[40, 400]$ mg/dl into intervals of size 10 mg/dl over the range $[40, 80]$ mg/dl and 20 mg/dl intervals over the remaining range $[80, 400]$ mg/dl. This yields a finite alphabet $\Sigma$ where $|\Sigma| = 20$. For instance $a_{60,70} \in \Sigma$ represents a range $[60, 70] mg/dl$ whereas $a_{220,240}$ represents the $[220, 240]$ mg/dl range. CGMs provide a reading periodically at 5 minute intervals. This yields a string where each letter describes the interval that contains the glucose value.

#### Transducer

The CGM error model is given by a transducer that considers possible errors that a CGM can make (see Figure 4.5). The transducer has four states: Not Calib denoting that no calibration has happened, Calib denoting a calibration event in the past, DropoutNC a sensor drops out under the non calibrated mode and DropoutC a calibration event has happened and sensor drops out. As mentioned earlier, we have variable costs. It also associates a cost with these variable translation costs defined by a function $\mathsf{cost}(lb, ub, lb', ub')$. These costs are set to be higher for ranges $[lb, ub]$ that are close to

[III]Such an event can lead to dangerous (and silent) night time seizures.

[IV]Such an event can lead to a potentially dangerous condition called diabetic ketacidosis.

[V]Rebound hyperglycemia can lead to large future swings in the blood glucose level, raising the burden on the patient for managing their blood glucose levels.

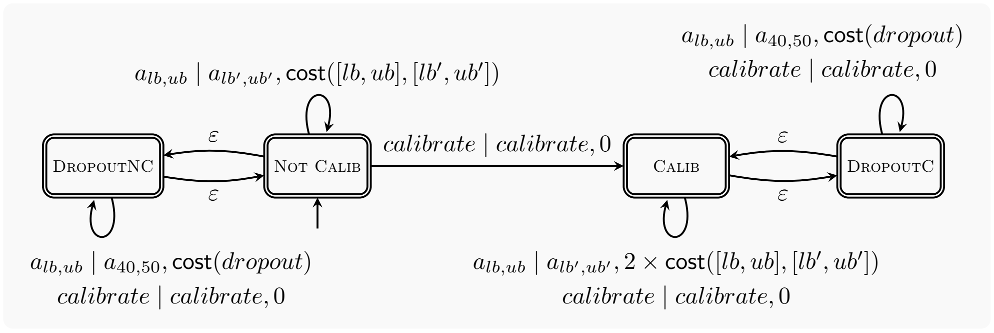

$a_{lb,ub} \mid a_{lb',ub'}, \mathsf{cost}([lb,ub],[lb',ub'])$
$a_{lb,ub} \mid a_{40,50}, \mathsf{cost}(dropout)$
$calibrate \mid calibrate, 0$
DropoutNC
Not Calib
Calib
DropoutC
$\varepsilon$
$\varepsilon$
$calibrate \mid calibrate, 0$
$\varepsilon$
$\varepsilon$
$a_{lb,ub} \mid a_{40,50}, \mathsf{cost}(dropout)$
$calibrate \mid calibrate, 0$
$a_{lb,ub} \mid a_{lb',ub'}, 2 \times \mathsf{cost}([lb,ub],[lb',ub'])$
$calibrate \mid calibrate, 0$

Figure 4.5: Transducer model for capturing the errors made by continuous glucose monitors.

hypoglycemia. Also note that we can model calibration events and the doubling of costs if the sensor is in the calibrated mode. In practice, the sensor drifts over time after a calibration event and this is not modeled in our study. Finally, there are states modeling CGM dropouts that results in a constant signal in the range $[40, 50]$ mg/dl. These events are not true hypoglycemia but a temporary sensor malfunction.

| Prop. | Total Time | Threshold Values synthesized | | | | |
|---|---|---|---|---|---|---|
| | | 0 | $(0, 0.1]$ | $(0.1, 1.0]$ | $> 1.0$ | $\infty$ |
| $P1$ | $4hr10m31s$ | 0 | 8 | 2 | 95 | 1927 |
| $P2$ | $2hr10m30s$ | 0 | 28 | 13 | 0 | 1991 |
| $P3$ | $2h0m9s$ | 0 | 11 | 10 | 0 | 2011 |

Table 4.2: Table filled by Sankaranarayanan and Trivedi, showing total time taken per property and number of matches for various ranges of robustness threshold.

## Property Specifications

We specify the three different properties described above formally using finite state machines over the alphabet $\Sigma$ as defined above. The prolonged hypoglycemia property can be written as a regular expression: $\Sigma^*(a_{40,50} + a_{50,60} + a_{60,70})^{36}\Sigma^*$ which can be easily translated into a NFA with roughly 38 states. The number 36 represents a period of 180 minutes since CGM values are sampled at 5 minute intervals. Similarly, the other two properties are also easily expressed as NFA.

Finally, we compose the transducer model with the properties P1, P2, P3 individually and calculate the mean robustness. More precisely, for each sequence of measures $w$, we compute the minimal threshold $\nu$ such that $w$ can be rewritten by $T$ at mean cost $\nu$ into some $w'$ satisfying P1 (and P2, P3 respectively). The discounted sum robustness is not useful in this situation since the patterns can match approximately anywhere in the middle of a trace. Also, in most cases the discounted sum robustness value was very close to zero for any discount factor $< 1$ or became forbiddingly large for discount factors slightly larger than 1, due to the large size of the traces.

## Patient Data

We used actual patient data involving nearly 50 patients with type-1 diabetes undergoing a clinical trial of an artificial pancreas device, and nearly 40 nights of data per patient, leading to an overall 2032 nights. Each night roughly corresponds to a 12 hour period when CGM data was recorded [MCB+14]. This is converted to a string of size 140 (or slightly larger, depending on how many calibration events occurred). The threshold synthesis problem 4.1.3 was solved for each of the input strings, and the results were sorted by the threshold robustness value for properties P1, P2, P3.

Table 4.2 shows for each property, the total time taken to complete the analysis of the full patient data, and the number of matches obtained corresponding to various threshold values. The implementation provided by Sankaranarayanan and Trivedi is currently a prototype that constructs a full product graph of the property automaton, transducer and the input string converted into a “straight line” automaton. This results in roughly $4 \times 140 \times 38 * 4 \simeq 80,000$ node graph for each individual trace, and is not quite efficient in

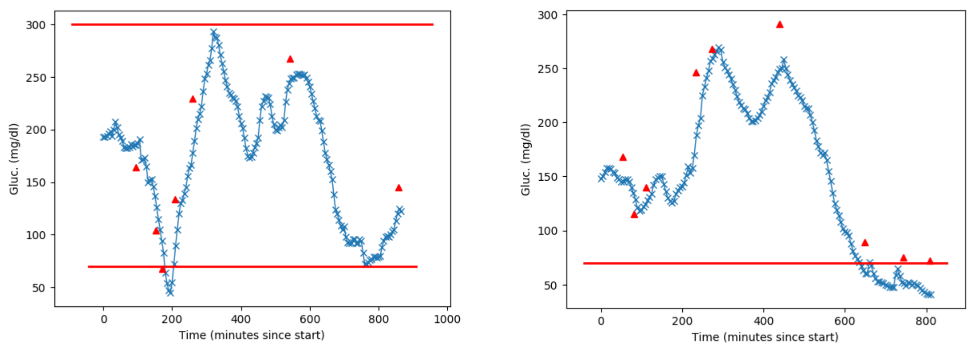


Figure 4.6: Examples of patterns with small robustness thresholds for properties P1 (left) with robustness threshold value of 0.7 and P3 (right) with robustness value of 0.02. The red triangles show calibration events.

terms of time to construct the product graph, prune it and calculate the mean robustness value. Nevertheless, each trace takes less than 1 minute for property P1 and roughly half that time for properties P2 and P3. As the table reveals, *no single trace matches any of the properties perfectly.* However, our approach is more nuanced, and thus, allows us to find numerous approximate matches that can be sorted by their robustness threshold values. Note that many of the input traces yield a threshold value of $\infty$: this signifies that no possible translation as specified by the transducer can cause the property to hold. Figure 4.6 shows two of the approximate pattern matches obtained with a small robustness value. Notice that the CGM values on the left do not satisfy the criterion for a prolonged hypoglycemia for 3 hours (P1) in a strict sense due to a single point at the end of the trace that is slightly above the 70 mg/dl threshold. Nevertheless, our approach assigns this trace a very low robustness. Likewise, the plot on the right shows a rapid rise from a hypoglycemia to a hyperglycemia within 120 minutes (P3) towards the beginning, except that the peak value just falls short of the threshold of 300 mg/dl.

Note that related work in the area of monitoring cyber-physical [FP09, DM10, AH15, DMP17] can be used to perform approximate pattern recognition using robustness of temporal properties over hybrid traces. However, we note important differences that are achieved due to the theory developed in this chapter. For one, the use of a transducer can provide a nuanced model of how errors transform a trace, wherein the transformation itself changes based on the transducer state. This is used in many ways in our application:

- It models the fact that the sensor is less error prone at smaller glucose values.
- It models events such as calibration that make the sensor more accurate for a period of time after calibration.
- The use of transducers can also (roughly) model important CGM errors such as dropouts and pressure-induced sensor attenuation [BCB+14]. A detailed transducer model of CGM errors remains beyond the scope of this study but will likely be desirable for applications to the analysis of patterns in type-1 diabetes data.

# Chapter 5

# Robust kernel synthesis

In this chapter, we carry the study of robustness a little further and consider properties on the robust kernel of a regular language with respect to an error model. Given two regular languages $N, L$ and a weighted transducer $T$, the robust kernel is the set of words from $N$ for which any perturbation performed by $T$ that is in the $\nu$-neighborhood, belongs to $L$. In other words, the robust kernel is the largest subset of $N$ for which any rewriting of $T$ with a cost bounded by $\nu$, belongs to $L$. In particular, we investigate the regularity and the emptiness of the robust kernel, denoted $\mathtt{Rob}_T(\nu, L)$ in Chapter 4, for a given $\mathsf{C}$-transducer $T$, a non-negative rational $\nu$ and a regular language $L$ represented by a $\mathsf{DFA}$. We show that the robust kernel is regular for the $\mathtt{Sum}$ measure, and checking its emptiness is PSPACE-C. For $\mathtt{Mean}$, we show that it is not necessarily regular, and checking its emptiness is undecidable. For $\mathtt{DSum}$, we conjecture that the robust kernel is non-regular and leave the emptiness problem open. Moreover, we provide sufficient conditions for $\mathtt{Mean}$ and for $\mathtt{DSum}$ measures under which the robust kernel is regular and computable, implying decidability of its emptiness.

## 5.1 Sum measure

To show robust kernel regularity, we rely on the construction of Theorem 2 of [ABK11] in the context of weighted automata over the semi-ring $(\mathbb{N} \cup \{+\infty\}, \min, +)$, denoted $\mathsf{WA}^{\min}_{\text{sum}}$ in Section 1.4. The following lemma, use the same automata construction and provides an upper bound on the number of states required to denote a threshold language with a $\mathsf{DFA}$.

To show robust kernel regularity, we rely on the following lemma:

**Lemma 5.1.1**

Let $U$ be an $n$ states $\mathsf{WA}^{\min}_{\text{sum}}$ and $\nu \in \mathbb{N}$. The threshold language $L_\nu(U) = \{u : [\![U]\!](u) \geq \nu\}$, where $[\![U]\!](u)$ is defined as $+\infty$ if there is no accepting run on $u$, otherwise as the minimal sum of the weights along accepting runs on $u$, is regular. Moreover $L_\nu(U)$ is recognized by a $\mathsf{DFA}$ with $O\big((\nu+2)^n\big)$ states.

**Proof** First, let assume that $U$ has universal domain (i.e. any word has some accepting run), otherwise we complete it by assigning value $\nu$ to each word of its complement.
Then, $[\![U]\!](u) \geq \nu$ iff all the accepting runs on $u$ have value at least $\nu$. We design a $\mathsf{DFA}$ $D$ that accepts exactly those words. Since the weights of $U$ are non-negative, $D$ just has to monitor the sum of all runs up to $\nu$, by counting in its states. If $Q$ is the set of states of $U$, the set of states of $D$ is $2^{Q \times \{0,\dots,\nu-1,\nu^+\}}$, where $\nu^+$ intuitively means any value $\geq \nu$. We extend natural addition to $X = \{0, \dots, \nu-1, \nu^+\}$ by letting $i + j = \nu^+$ iff $i = \nu^+$, or $j = \nu^+$, or $i, j \geq \nu$. Then, $D$ is obtained by subset construction, there is a transition $P \xrightarrow{a} P'$ in $D$ iff $P' = \{(q', i+j) : (q, i) \in P \wedge q \xrightarrow{a \mid j} q'\}$. A state $P$ is accepting if $P \cap \big((Q \setminus F) \times \{0, \dots, \nu-1\}\big) = \varnothing$, where $F$ are the accepting states of $U$.
Though simple, the latter construction does not give the claimed complexity, as the number of states of $D$ is $2^{n\nu}$. But the following simple observation allows us to get a better state complexity. Consider an input word of the form $uv$. If after reading $u$, $D$ reaches some state $P$ such that for some state $q$, there exists $(q,i),(q,j) \in P$ such that $i < j$, then if there is an accepting run of $U$ from $q$ on $v$, with sum $s$, there is an accepting run on $uv$ with sum $i + s$ and one with sum $j + s$. Therefore if $i + s \geq \nu$, then $j + s \geq \nu$ and the pair $(q, j)$ is useless in $P$. So, we can keep only the minimal elements in the states of $D$, where minimality is defined with respect to the partial order $(q,i) \preceq (p,j)$ if $q = p$ and $i \leq j$. Let us call $D_{opt}$ the resulting "optimized" $\mathsf{DFA}$. It states can be therefore seen

as functions from $Q$ to $\{0,\dots,\nu,\nu^+\}$, so that we get the claimed state-complexity. ♦

**Lemma 5.1.2 – Robust language regularity**

Let $T$ be a $\texttt{Sum}$-transducer, $\nu \in \mathbb{N}$ and $L$ a regular language. The language of *robust words* $\texttt{Rob}_T(\nu, L)$ is regular. Moreover, if $L$ is given by a DFA with $n_L$ states and $T$ have $n_T$ states, then $\texttt{Rob}_T(\nu, L)$ is recognizable by a DFA with $O\big((\nu+2)^{n_T \times n_L}\big)$ states.

**Proof** First, we show the regularity of the complement of $\texttt{Rob}_T(\nu, L)$ defined as

$$\overline{\texttt{Rob}_T(\nu, L)} = \{u_1 : \exists u_2,\ \texttt{Sum}_T(u_1,u_2) < \nu \wedge u_2 \notin L\}$$

First, let us assume that $L$ is given by some NFA $A$, let $\overline{A}$ be a DFA recognizing the complement of $L$. We first transform $T$ into $T \otimes \overline{A}$, which simulates $T$ and controls that the output words belong to $\overline{L}$. In particular, it rejects whenever the rewriting by $T$ is in $L$. It is obtained as a product of $T$ with $\overline{A}$ run on the output, with set of states $Q_T \times Q_{\overline{A}}$. It accepts whenever the final pair of states $(p,q)$ is a pair of accepting states both for $T$ and $\overline{A}$. Then, we have the following:

$$\overline{\texttt{Rob}_T(\nu, L)} = \{u_1 : \exists u_2,\ \texttt{Sum}_{T\otimes\overline{A}}(u_1,u_2) < \nu\}$$

Now, by definition of $\texttt{Sum}_{T\otimes\overline{A}}(u_1,u_2)$ we have $u_1 \in \overline{\texttt{Rob}_T(\nu, L)}$ iff there exists a word $u_2$ and an accepting run $r$ over $(u_1,u_2)$ such that $\texttt{Sum}(r) < \nu$. Therefore, we can project $T \otimes \overline{A}$ on its input dimension (thus, we just ignore the outputs) and obtain a $\mathsf{WA}^{\min}_{\text{sum}}$ that we call $U$ such that

$$\overline{\texttt{Rob}_T(\nu, L)} = \{u_1 : [\![U]\!](u_1) < \nu\}$$

where $[\![U]\!](u_1)$ is defined as $+\infty$ if there is no accepting run of $U$ on $u_1$, and as the minimal sum of the accepting runs on $u_1$ otherwise. Complementing again, we get:

$$\texttt{Rob}_T(\nu, L) = \{u_1 : [\![U]\!](u_1) \geq \nu\}$$

Now, we apply directly Lemma 5.1.1 on $U$ to conclude for regularity. The state-complexity is again given by Lemma 5.1.1 and the fact that $U$ has $n_T \times n_L$ states. ♦

**Theorem 5.1.3**

Let $T$ be a $\texttt{Sum}$-transducer, $\nu \in \mathbb{N}$ given in binary and $L$ a regular language given as a DFA. Then, it is PSPACE-C to decide whether there exists a robust word $u \in \texttt{Rob}_T(\nu, L)$. The hardness holds even if $\nu$ is a fixed constant, $T$ is letter-to-letter[VI] and io-unambiguous[VII], and its weights are fixed constants in $\{0,1\}$.

**Proof** From Lemma 5.1.2, $\texttt{Rob}_T(\nu, L)$ is recognizable by a DFA with $O\big((\nu+2)^{n_T\times n_L}\big)$ states, where $n_T$ is the number of states of $T$ and $n_A$ the number of states of the DFA defining $L$. Checking emptiness of this automaton can be done in PSPACE (apply the standard NLOGSPACE emptiness checking algorithm on an exponential automaton that needs not be constructed explicitly, but whose transitions can be computed on-demand). To show PSPACE-hardness, we reduce the problem from [Koz77] of checking the non-emptiness of the intersection of $n$ regular languages given by $n$ DFA $A_1,\dots,A_n$, over some alphabet $\Gamma$. In particular, we construct $T$, $\nu$ and a DFA $A$ such that $\bigcap_i L(A_i) \neq \varnothing$ iff there exists a robust word with respect to $T$,$\nu$ and $L$.
We define the alphabet as $\Sigma = \Gamma \cup \{\#_1,\dots,\#_n,\diamond\}$ where we assume that $\#_1,\dots,\#_n,\diamond \notin \Gamma$, and construct a transducer $T$ which reads a word $u\diamond$ of length $k = |u|+1$ with $u \in \Gamma^*$, and rewrites it into either itself, or $(\#_i)^k$ for all $i \in \{1,\dots,n\}$. The identity rewriting has total weight 0 while the rewriting into $\#_i^k$ has total weight 1 if $u \in L(A_i)$, and 0 otherwise. The transducer $T$ is constructed as the disjoint union of $n+1$ transducers $T_1,\dots,T_n,T_\diamond$.

[VI] A transducer is letter-to-letter if $\Delta \subseteq Q \times \Sigma \times \Sigma \times Q$.
[VII] For all word pairs $(u_1,u_2)$, there exists at most one run of $T$ on $u_1$ outputting $u_2$.

For all $i \in \{1,\dots,n\}$, $T_i$ simulates $A_i$ on the input and outputs $\#_i$ whenever it reads an input letter different from $\diamond$, with weight 0. When reading $\diamond$ from an accepting state of $A_i$, it outputs $\diamond$ with weight 1, and if it reads $\diamond$ from a non-accepting state, it outputs $\diamond$ with weight 0. Finally, $T_\diamond$ just realizes the identify function with weight 0. Note that $T$ has polynomial size in $A_1,\dots,A_n$ and it is letter-to-letter and (input,output)-deterministic. Now we prove that a word $u\diamond$ is robust iff $u \in \bigcap_i L(A_i)$. Assume that there exists a robust word $u\diamond$ for the property $L = (\Gamma \cup \{\diamond\})^*$ and threshold $\nu = 0$. Equivalently, it means that for all rewritings $\alpha \in \Sigma^*$, if $\texttt{Sum}_T(u\diamond,\alpha) \leq 0$ then $\alpha \in L$. It is equivalent to say that all its rewritings $\alpha$ satisfies either $\texttt{Sum}_T(u\diamond,\alpha) \geq 1$ or $\alpha \in L$. By definition of $T$, it is equivalent to say that all rewritings $\alpha$ are such that either $\alpha \in (\#_i)^*\diamond$ for some $i$ and $u \in L(A_i)$, or $\alpha = u\diamond$. Since $T$ necessarily rewrites $u\diamond$ into $u\diamond$, as well as into $(\#_1)^k,\dots,(\#_n)^k$, where $k = |u|+1$, the latter assumption is equivalent to saying that $u \in L(A_i)$ for all $i \in \{1,\dots,n\}$, concluding the proof. ♦

## 5.2 Mean measure

Let us first establish non-regularity of the robust kernel.

**Lemma 5.2.1**

Given a regular language $L$, a $\texttt{Mean}$-transducer $T$ and $\nu \in \mathbb{Q}_{\geq 0}$, the language $\texttt{Rob}_T(\nu, L)$ is not necessarily regular, but recursive.

**Proof** Consider the language $L = \{u : \exists i \in \mathbb{N}\ u[i] = a\}$ on the alphabet $\Sigma = \{a,b\}$, i.e. the set of words on $\Sigma$ that contain at least one $a$. Now, consider a (one state) transducer $T$ that can non-deterministically copy letters or change the current letter from $a$ to $b$ with weight one. Now, if we fix $\nu$ to be equal to $\frac{1}{2}$, then all the translations of $u$ by $T$ of cost less than $\frac{1}{2}$ are included in $L$, i.e. each translation of $u$ will contain at least one letter $a$ iff the number of $a$'s in $u$ is larger than the number of $b$'s in $u$, i.e. $\texttt{Rob}_T(\frac{1}{2}, L) = \{u : |u|_a > |u|_b\}$, which is not regular. Note that in general $\texttt{Rob}_T(\nu, L)$ is recursive because the membership problem to it, is decidable by Corollary 4.2.4 (applied on a singleton language). ♦

For $\texttt{Sum}$-transducers, the regularity of the robust kernel rely on the non-negativity of its weights. However, in the case of $\texttt{Mean}$-transducers, being weighted over non-negative integers or over $\mathbb{Z}$ does not change the problem. We use this fact to show that testing the non-emptiness of the robust kernel is undecidable.

**Theorem 5.2.2**

Let $L$ be a regular language, $T$ be a $\texttt{Mean}$-transducer and $\nu \in \mathbb{Q}_{\geq 0}$. Determine whether $\texttt{Rob}_T(\nu, L) \neq \varnothing$ is undecidable. It holds even if $T$ is io-unambiguous.

**Proof** The proof goes by reduction from the universality problem for a weighted automata $A$ over $(\mathbb{Z}, \min, +)$, known to be undecidable [CDH10b, ABK11]. We recall that this problem asks whether all words have a finite value smaller than 0.
Given $A = (Q, Q_I, Q_F, \Delta, \lambda)$, we construct $L$ as the set of *non* accepting runs of $A$ union $\Sigma^*$, the threshold $\nu$ as the maximal absolute weight of $A$ and $T$ such that:

$$\texttt{Mean}_T = \bigcup \begin{array}{l} \{(u,u) \mapsto 0 : u \in \Sigma^*\} \\ \{(u,\varrho) \mapsto x_\varrho + \nu|u| : \varrho \in \texttt{AccRun}_A(u) \wedge \lambda(\varrho) = x_\varrho\} \end{array}$$

We can construct $T$ as the disjoint union between a single-state transducer with weights zero realizing the identity, and a transducer that outputs all the possible runs of $A$ on its input, such that each $T$-transition simulating an $A$-transition $t$ of value $x$ (in $A$) has value $\nu + x$, which is positive by definition of $\nu$. Hence $T$ is indeed weighted over non-negative numbers. Note that $T$ is io-unambiguous: if the input and output are fixed, there is at most one run of $T$. Now, we show that $\texttt{Rob}_T(\nu, L) = \varnothing$ iff $\forall u\ [\![A]\!](u) \leq 0$, i.e.

$$\forall u_1 \exists u_2 \in \overline{L}\ \texttt{Mean}_T(u_1, u_2) \leq \nu \iff \forall u\ [\![A]\!](u) \leq 0$$

We have the following equivalences: $\forall u_1 \exists u_2 \in \overline{L}\ \texttt{Mean}_T(u_1,u_2) \leq \nu$ iff for all $u_1$, there exists an accepting run $\varrho$ of $A$ on $u_1$ such that $\texttt{Mean}_T(u_1,\varrho) \leq \nu$, i.e. $\texttt{Sum}_T(u_1,\varrho) \leq \nu|u_1|$

and by definition of $T$, it is equivalent to asking that $\lambda(\varrho) + \nu|u_1| \leq \nu|u_1|$, i.e. $\lambda(\varrho) \leq 0$. Hence, the latter statement is equivalent to the fact that for all words $u_1$, there exists an accepting run of $A$ of value $\leq 0$. Since $A$ takes the minimal value of all accepting runs on $u_1$, it is equivalent to saying that for all $u_1$, $[\![A]\!](u_1) \leq 0$, i.e. $A$ is universal, concluding the proof. ♦

## 5.3 Discounted sum measure

For DSum-transducer, we conjecture that $\mathtt{Rob}_T(\nu, L)$ is in general non-regular. This claim is substantiated by the fact that DSum-automata over $\mathbb{Q}$ and $\omega$-words have in general non-regular cut-point languages, i.e. the set of words of DSum value below a given threshold is in general non-regular [CDH10a]. With a proof similar to that of Theorem 5.2.2 for Mean-transducers, it is possible to show that the universality problem for DSum-automata, which is open to the best of our knowledge, reduces to checking the emptiness of the robust language of a DSum-transducer.

Following an approach that originates from the theory of probabilistic automata, it is has been shown that cut-point languages are regular when the threshold is $\delta$-isolated [CDH10a]. Formally, a threshold $\nu \in \mathbb{Q}$ is $\delta$-isolated, for $\delta > 0$ and for some DSum-transducer $T$ if, for all accepting runs $r$ of $T$, $\mathtt{DSum}_T(r) \in [0, \nu - \delta] \cup [\nu + \delta, +\infty)$. It is *isolated* if it is $\delta$-isolated for some $\delta$. Our objective now is to show that when $\nu$ is isolated, then $\mathtt{Rob}_T(\nu, L)$ is regular and one can effectively construct an automaton recognizing it. We will also give a (possibly non-terminating) algorithm which, when it terminates, returns an automaton recognizing $\mathtt{Rob}_T(\nu, L)$, and which is guaranteed to terminate whenever $\nu$ is $\delta$-isolated for some $\delta$. Towards these results, we first give intermediate useful results. For a state $q$ of $T$, we call *continuation* of $q$ any run from $q$ leading to some accepting state of $T$. By extension, we also call continuation of a run $r$ any continuation of the last state of $r$. A transducer $T$ is said to be *trim* if all its states admits some continuation. Note that any transducer can be transformed into an equivalent trim one in PTIME, just by removing states that do not admit any continuation (this can be tested in PTIME).

**Lemma 5.3.1**

Let $T$ be a trim DSum-transducer and $\nu \in \mathbb{Q}$. If $\nu$ is $\delta$-isolated for some $\delta$, then there exists $\hat{n} \in \mathbb{N}$ such that any run $r$ of length at least $\hat{n}$ satisfies one of the following properties:

1. $\mathtt{DSum}(r) \leq \nu - \delta$ and any continuation $r'$ of $r$ satisfies $\mathtt{DSum}(rr') \leq \nu - \delta$
2. $\mathtt{DSum}(r) \geq \nu + \delta/2$ and any continuation $r'$ of $r$ satisfies $\mathtt{DSum}(rr') \geq \nu + \delta$.

**Proof** Let $r$ be a run of length $n$ of $T$. Since $T$ is trim, there exists a continuation $r'$ of $r$, and moreover we have $\mathtt{DSum}(rr') = \mathtt{DSum}(r) + \lambda^n \mathtt{DSum}(r')$. We have $\mathtt{DSum}(r') \leq \sum_{i=0}^{+\infty} \lambda^i \mu = \mu(1-\lambda)^{-1}$ where $\mu$ is the largest absolute weight of $T$. We let $B_n = \lambda^n \mu (1-\lambda)^{-1}$. Let $\hat{n}$ be the smallest non-negative integer such that $B_{\hat{n}} \leq \delta/2$ (it exists since $B_n$ is strictly decreasing of limit 0). Assume that the length of $r$ is greater than $\hat{n}$ i.e. $n \geq \hat{n}$. As a consequence $B_n \leq B_{\hat{n}}$. Since $\nu$ is $\delta$-isolated, we have two cases:

i. If $\mathtt{DSum}(rr') \leq \nu - \delta$ then $\mathtt{DSum}(r) \leq \nu - \delta$ since $\mathtt{DSum}(r) \leq \mathtt{DSum}(rr')$ by non-negativity of the weights of $T$
ii. If $\mathtt{DSum}(rr') \geq \nu + \delta$ then $\mathtt{DSum}(r) \geq \nu + \delta - \lambda^n \mathtt{DSum}(r')$. Moreover $\lambda^n \mathtt{DSum}(r') \leq B_n \leq B_{\hat{n}} \leq \delta/2$ by construction. So $-\lambda^n \mathtt{DSum}(r') \geq -\delta/2$ which implies $\mathtt{DSum}(r) \geq \nu + \delta/2$.

We have just shown that either $\mathtt{DSum}(r) \leq \nu - \delta$ by (i) or $\mathtt{DSum}(r) \geq \nu + \delta/2$ by (ii). We prove now that, for all continuation $r'$ of $r$ we have (i) implies $\mathtt{DSum}(rr') \leq \nu - \delta$ and (ii) implies $\mathtt{DSum}(rr') \geq \nu + \delta$. In the first case, assume by contradiction that (i) holds and some continuation $r'$ of $r$ satisfies $\mathtt{DSum}(rr') \geq \nu + \delta$. As a consequence $\lambda^n \mathtt{DSum}(r') \geq 2\delta$, which is impossible since $\lambda^n \mathtt{DSum}(r') \leq B_n \leq B_{\hat{n}} \leq \delta/2$. In the second case, if $\mathtt{DSum}(r) \geq \nu + \delta/2$ then any continuation $r'$ of $r$ satisfies $\mathtt{DSum}(rr') \geq \mathtt{DSum}(r) > \nu + \delta/2$. Since $\nu$ is $\delta$-isolated, we get $\mathtt{DSum}(rr') \geq \nu + \delta$. ♦

We now show how to construct better and better regular under-approximations of the set of *non*-robust words, show that they "finitely" converge to the set of non-robust words when $\nu$ is isolated.

**Lemma 5.3.2**

Let $T$ be a $\texttt{DSum}$-transducer, $\nu \in \mathbb{Q}$ and $L$ a regular language given as a $\mathsf{DFA}$. For all $n$, we can construct an $\mathsf{NFA}$ $A_n$ such that:

1. $L(A_n) \subseteq L(A_{n+1})$
2. $L(A_n) \subseteq \overline{\texttt{Rob}_T(\nu, L)} \cap \texttt{dom}(T)$

Moreover, if $\nu$ is isolated, there exists $\hat{n}$ such that $L(A_{\hat{n}}) = \overline{\texttt{Rob}_T(\nu, L)} \cap \texttt{dom}(T)$.

**Proof** For all $n$, we let $B_n = \lambda^n W(1-\lambda)^{-1}$, as in the proof of Lemma 5.3.1. A run $r$ on a pair $(w_1, w_2)$ is called *bad* if $\texttt{DSum}(r) \leq \nu$, $w_2 \notin L$ and $r$ is accepting. Not that necessarily, $w_1 \notin \texttt{Rob}_T(\nu, L)$. The run $r$ is called *dangerous* if $|r| \geq n$ and $\texttt{DSum}(r) \leq \nu - B_n$. A dangerous run $r$ can possibly be extended to a bad run $rr'$. It is possible iff there exists a continuation $r'$ of $r$ such that the output of $rr'$ is not in $L$. Note that the cost of $rr'$ does not matter because the largest value $r'$ can achieve is $B_n$, keeping $\texttt{DSum}(rr')$ smaller than $\nu$. Hence, when a dangerous run is met, only a regular property has to be tested to extend it to a bad run. We exploit this idea in the automata construction. Namely, $A_n$ will accept words for which there exists a bad run of length $n$ at most, or a dangerous run of length $n$ which can be extended to a bad run.

*Automata construction.* Let $\texttt{AccRun}_T^{\leq n}$ be the runs of $T$ of length at most $n$, and $Q$ its set of states. We assume that for all $(w_1, w_2) \in [\![T]\!]$, $w_2 \notin L$ holds. This can be ensured by taking the synchronized product of $T$ (on its outputs) with an automaton recognizing the complement of $L$. Let us now build the $\mathsf{NFA}$ $A_n$. Its set of states is $\texttt{AccRun}_T^{\leq n} \cup Q$. Its transitions are defined as follows: for all $T$-runs $r$ of length $n-1$ at most ending in some state $q$, for all $\sigma \in \Sigma_\varepsilon$, if there exists a transition $t$ of $T$ from state $q$ on reading $\sigma$, then we create the transition $r \xrightarrow{\sigma} rt$ in $A_n$. From any run $r$ of length $n$, we consider two cases: if $r$ is not dangerous, then $r$ has no outgoing transitions in $A_n$. If $r$ is a dangerous run, then we add some $\varepsilon$-transition to its last state: $r \xrightarrow{\varepsilon} p$ where $p$ is the last state of $r$. Finally, we add a transition from any state $q$ to any state $q'$ on $\sigma$ in $A_n$ whenever there is a transition from $q$ to $q'$ on input $\sigma$ in $T$. Accepting states are bad runs of $\texttt{AccRun}_T^{\leq n}$ and accepting states of $T$.

*Correctness.* Let us show that the family $A_n$ satisfies the requirements of the lemma. First, we show that $L(A_n) \subseteq L(A_{n+1})$. Let $w \in L(A_n)$ and $\varrho$ some accepting run of $A_n$ on $w$. To simplify the notations, we assume here in this proof that runs of $A_n$, $A_{n+1}$ and $T$ are just sequences of states rather than sequences of transitions. By definition of $A_n$, $\varrho$ can be decomposed into two parts $\varrho_1\varrho_2$ such that $\varrho_1 \in (\texttt{AccRun}_T^{\leq n})^*$ and $\varrho_2 \in Q^*$ with an $\varepsilon$-transition from the last state of $\varrho_1$ to the first of $\varrho_2$. We consider two cases. If $|\varrho_2| = 0$, then $\varrho = \varrho_1$ and by definition of $A_{n+1}$, $\varrho$ is still an accepting run of $A_{n+1}$. In the other case, there is a dangerous run $r$ of $T$ such that $\varrho_1$ can be written $\varrho_1 = r[..1]r[..2]\dots r[..n]$ where $r[..i]$ is the prefix of $r$ of length $i$, and $\varrho_2 = q_1q_2\dots, q_k$ is a proper run of $T$. Note that $q_1$ is the last state of $r$ by construction of $A_n$. Moreover, $r\varrho_2$ is bad. Since $r$ was dangerous at step $n$, we also get that $rq_2$ is dangerous at step $n+1$, in the sense that $|rq_2| = n+1$ and $\texttt{DSum}(rq_2) \leq \nu - B_{n+1}$, by definition of $B_{n+1}$ and the fact that $\texttt{DSum}(r) \leq \nu - B_n$. So, we get that the sequence of states $\varrho_1 \cdot (rq_2) \cdot q_2 \dots q_k$ is a run of $A_{n+1}$ on $w$ is accepting in $A_{n+1}$ (note that $rq_2$ here is a state of $A_{n+1}$ and there is an $\varepsilon$-transition from $(rq_2)$ to $q_2$), concluding the first part of the proof.

Now, suppose that $\nu$ is $\delta$-isolated for some $\delta$. Then, take $\hat{n}$ as given by Lemma 5.3.1 and let us show that $\overline{\texttt{Rob}_T(\nu, L)} \cap \texttt{dom}(T) \subseteq L(A_{\hat{n}})$ (the other inclusion has just been proved for all $n$). Let $w \in \texttt{dom}(T)$ such that $w \notin \texttt{Rob}_T(\nu, L)$. There exists $(w_1, w_2) \in R_T$ and an accepting run $r$ of $T$ on it such that $\texttt{DSum}(r) \leq \nu$ and $w_2 \notin L$. In other words, $r$ is bad. If $|r| \leq \hat{n}$, then $r[..1]r[..2]\dots r[..|r|]$ is an accepting run of $A_{\hat{n}}$ on $w$, and we are done. Now suppose that $|r| > \hat{n}$. Since $\nu$ is $\delta$-isolated, we have $\texttt{DSum}(r) \leq \nu - \delta$. By Lemma 5.3.1, we also get that $\texttt{DSum}(r[..\hat{n}]) \leq \nu - \delta$. By definition of $\hat{n}$ being the smallest integer such that $B_{\hat{n}} < \delta/2$, we get $\texttt{DSum}(r[..\hat{n}]) \leq \nu - B_{\hat{n}}$, hence $r[..\hat{n}]$ is dangerous. We can conclude since then $r[..1]r[..2]\dots r[..\hat{n}]r[\hat{n}]r[\hat{n}+1]\dots r[|r|]$ is an accepting run of $A_{\hat{n}}$ on $w$.⧫

We also show that one can test whether given $n$, we have $\overline{\texttt{Rob}_T(\nu, L)} \cap \texttt{dom}(T) \subseteq L(A_n)$, as stated by the following lemma:

**Lemma 5.3.3**

Given a regular language $N$ (given as some NFA), it is decidable to check whether $\overline{\texttt{Rob}_T(\nu, L)} \cap \texttt{dom}(T) \subseteq N$ holds.

**Proof** It suffices to take the synchronized product of $T$, $\overline{L}$ (on the output) and $\overline{N}$ (on the input), project the output, and test for the existence of path from an initial to a final vertex of discounted sum $\leq \nu$. ♦

Those results allow us to define the semi-algorithm of Figure 5.1.

```
ComputeRob($T, \nu, L$)
For $n$ from 1 to $+\infty$
    compute $A_n$                                                              // as in Lemma 5.3.2
    if $\overline{\texttt{Rob}_T(\nu, L)} \cap \texttt{dom}(T) \subseteq L(A_n)$ return $A_n$        // using Lemma 5.3.3
```

Figure 5.1: Semi-algorithm that computes the robust kernel for a given DSum-transducer $T$, a non-negative rational $\nu$ and a regular language $L$ represented by a DFA.

**Lemma 5.3.4**

The algorithm ComputeRob$(T, \nu, L)$ satisfies the following properties:
1. if it terminates, then it returns an automaton recognizing $\overline{\texttt{Rob}_T(\nu, L)} \cap \texttt{dom}(T)$,
2. if $\nu$ is isolated, it terminates.

**Proof** If it terminates at steps $n$, then by Lemma 5.3.2 and the test at line 4 we know that $L(A_n) = \overline{\texttt{Rob}_T(\nu, L)} \cap \texttt{dom}(T)$, and if $\nu$ is isolated, the test will eventually succeed. ♦

Note that the algorithm may terminate even if $\nu$ is not isolated. It is the case for instance when the threshold is $\delta$-isolated for "long" runs only, but not necessarily for small runs, in the sense that it is only required that for some $n$, any accepting runs of length at least $n$ satisfies either $\texttt{DSum}(r) \leq \nu - \delta$ or $\texttt{DSum}(r) \geq \nu + \delta$.

As a corollary of Lemma 5.3.4, $\texttt{Rob}_T(\nu, L)$ is regular when $\nu$ is isolated: it suffices to run Algorithm ComputeRob, complement the obtained automaton and restrict it its language to $\texttt{dom}(T)$.

**Theorem 5.3.5**

Let $T$ be a DSum-transducer and $\nu \in \mathbb{Q}$ and $L$ a regular language. If $\nu$ is isolated, then $\texttt{Rob}_T(\nu, L)$ is regular.

## 5.4 Related works

The framework of model measuring or robust model-checking has been studied with various approaches in [MRT11, HO13, CHJS15] for diverse formalisms and measures.

The DSum measure is one the most popular in the literature to model dynamic systems. In our case, to ensure the regularity of the robust kernel, we considered DSum weighted automata that are isolated. This technique have been used similarly in [CDH10a] in the context of infinite words. Another way to recover decidability results would be to consider structural properties. In fact, the functional fragment of DSum weighted automata, i.e. the subclass that defined automata which associate at most one value for all input, also have decidable quantitative language inclusion [FGR15, BHO15] as well as the fragment of finite valued when the discount factor is of the form $\frac{1}{n}$ for some $n \in \mathbb{N}_{\neq 0}$ [FGR14]. However, the general statement is related to open problems in theoretical computer science that are known to be challenging as noticed in [BHO15].

The `Mean` is also a classical measure of the literature, more particularly in the context of infinite words. The window semantics similar to the one defined in Section 5.2 can be found in [HPR18] in the context of mean pay-off games. Note that, `Mean` weighted automata over infinite words, which is the case for mean pay-off games, are known to be prefix-independent and that is not the case with finite words. Again, bounding the number of values associated to any input is also a promising restriction to get decidability results. Since `Mean` weighted automata belongs to the class of group automata, they admit a decomposition into functional `Mean` weighed automata (as proved in [FGR14]), for which the language inclusion is decidable [FGR15].

# III

# Structural properties

# Chapter 6

# Specification languages for structural properties

Traditionally, the model-checking problem in computed-aided verification rely on regular automata [VW86, CHVB18]. Extensions have been developed for input/output systems, the most popular formalism are transducers and weighted automata. However, classical decision problems such as inclusion and equivalence, become quickly undecidable for those formalisms. It is also well known that automata-based techniques suffer from state explosions, even for the regular models. To retain feasibility and tractability of the model-checking, the verification community often fall back on automata subclasses. Note that, the relevance of a subclass may be conditioned by the existence of an efficient decision procedure for the subclass-membership. Since structural pattern characterizations are practical to implement, the literature counts many patterns and ad-hoc methods for determining whether an automaton fulfills them or not.

Our goal in this chapter is to introduce a specification framework which allows us to express the structural patterns of the literature as well as new ones. Hence, we provide a generic logic to define properties of automata with outputs in some monoid, in particular the set of predicates talking about the output values is parametric. Then, we consider three particular automata models (regular automata, transducers and automata weighted by integers) and instantiate the generic logic for each of them. We give tight complexity results for the model-checking problem of three logics and we study their expressiveness by expressing classical structural patterns characterizing for instance unambiguity in the case of finite automata, determinizability and finite-valuedness in the case of transducers and automata weighted by integers.

## 6.1 A generic specifications framework

In this section, we introduce a generic pattern logic. It is generic in the sense that the predicates talking about output monoid values are parameters. It is built over four kind of variables, namely path, state, input and output variables. More precisely, we let $X_P = \{\pi, \pi_1, \dots\}$, $X_Q = \{q, q_1, p \dots, \}$, $X_I = \{u, u_1, \dots\}$ and $X_O = \{v, v_1, \dots\}$ be disjoint and countable sets of resp. path, state, input and output variables. We define $\mathrm{Terms}(X_O, \oplus, \mathbb{0})$ as the set of terms built over variables of $X_O$, a binary function symbol $\oplus$ (representing the monoid operation) and constant symbol $\mathbb{0}$ (representing the neutral element of the monoid).

### Syntax of PL

The logic syntax is parametrized by a set of predicates $\mathcal{O}$, called output predicates. For all $p \in \mathcal{O}$, we denote by $\alpha(p)$ its arity. Output predicates of arity 0 are called constant symbols. Predicates talking about states, paths and input words are however fixed in the logic.

**Definition 6.1.1**

A *pattern formula* $\Phi$ over a set of output predicates $\mathcal{O}$ is of the form[I]:

$$\Phi \ ::= \ \exists \pi_1 \colon p_1 \xrightarrow{u_1|v_1} q_1, \dots, \exists \pi_n \colon p_n \xrightarrow{u_n|v_n} q_n \ \mathcal{C}$$

where for all $1 \leq i \leq n$, $\pi_i \in X_P$ and they are all pairwise different, $p_i, q_i \in X_Q$, $u_i \in X_I$, $v_i \in X_O$, and $\mathcal{C}$ is a Boolean combination of atoms amongst

| | | | |
|---|---|---|---|
| Input predicates | : | $u \sqsubseteq u' \mid u \in L \mid \lvert u\rvert \leq \lvert u'\rvert$ | $u, u' \in X_I$ |
| Output predicates | : | $p(t_1, \dots, t_{\alpha(p)})$ | $p \in \mathcal{O}, t_i \in \text{Terms}(X_O, \oplus, \mathbb{0})$ |
| State predicates | : | $\mathsf{init}(s) \mid \mathsf{final}(s) \mid s = s'$ | $q, q' \in X_Q$ |
| Path predicates | : | $\pi = \pi'$ | $\pi, \pi' \in X_P$ |

where $u, u' \in \{u_1, \dots, u_n\}$, $s, s' \in \{p_1, \dots, p_n, q_1, \dots, q_n\}$, $\pi, \pi' \in \{\pi_1, \dots, \pi_n\}$ and $L$ is a regular language of words over $\Sigma$ (assumed to be represented as a DFA). The sequence of existential quantifiers before $\mathcal{C}$ in $\Phi$ is called the *prefix* of $\Phi$. We denote by $\mathsf{PL}[\mathcal{O}]$ the set of pattern formulas over $\mathcal{O}$, and by $\mathsf{PL}^+[\mathcal{O}]$ the fragment where output predicates does not occur under an odd number of negations.

The size of a formula $\Phi$, denoted $|\Phi|$, is the number of its symbols plus the number of states of all DFA representing the membership predicates. We denote by $\text{Var}(\Phi)$ the variables occurring in any pattern formula $\Phi$, and by $\text{Var}_P(\Phi)$ (resp. $\text{Var}_Q(\Phi)$, $\text{Var}_I(\Phi)$, $\text{Var}_O(\Phi)$) its restriction to path (resp. state, input, output) variables. Finally, we define some macros for convenience as the state distinctness $q \neq q' \overset{\text{def}}{=} \neg(q = q')$, the input equality $u = u' \overset{\text{def}}{=} u \sqsubseteq u' \wedge u' \sqsubseteq u$, the input size comparisons $|u| = |u'| \overset{\text{def}}{=} |u| \leq |u'| \wedge |u'| \leq |u|$ and $|u| < |u'| \overset{\text{def}}{=} \neg(|u'| \leq |u|)$.

## Semantics of PL

To define the semantics of a pattern formula $\Phi$, we first fix some monoid $\mathcal{M} = (D_\mathcal{M}, \oplus_\mathcal{M}, \mathbb{0}_\mathcal{M})$ together with an interpretation $p^\mathcal{M}$ of each output predicates $p \in \mathcal{O}$ of arity $\alpha(p)$, such that $p^\mathcal{M} \in D_\mathcal{M}$ if $\alpha(p) = 0$ and $p^\mathcal{M} \subseteq D_\mathcal{M}^{\alpha(p)}$ otherwise. Given a valuation $\nu\colon X_O \to D_\mathcal{M}$, the interpretation over $\mathcal{M}$ can be inductively extended to terms $t \in \text{Terms}(X_O, \oplus, \mathbb{0})$ by letting $\mathbb{0}^{\nu,\mathcal{M}} = \mathbb{0}_\mathcal{M}$, $(t_1 \oplus t_2)^{\nu,\mathcal{M}} = t_1^{\nu,\mathcal{M}} \oplus_\mathcal{M} t_2^{\nu,\mathcal{M}}$ and $x^{\nu,\mathcal{M}} = \nu(x)$.

Then, a formula $\Phi \in \mathsf{PL}[\mathcal{O}]$ is interpreted in an automaton with outputs $A$ over $\mathcal{M}$ as a set of valuations $[\![\Phi]\!]_A$ of $\text{Var}(\Phi)$ which we now define. Each valuation $\nu \in [\![\Phi]\!]_A$ maps state variables to states of $A$, path variables to paths of $A$, etc. Such a valuation $\nu$ satisfies an atom $u \sqsubseteq u'$ if $\nu(u)$ is a prefix of $\nu(u')$, $u \in L$ if $\nu(u) \in L$, $|u| \leq |u'|$ if $|\nu(u)| \leq |\nu(u')|$. Similarly for state predicates, $\nu$ satisfies an atom $\mathsf{init}(q)$ if $\nu(q)$ is initial, $\mathsf{final}(q)$ if $\nu(q)$ is final and $q = q'$ of $\nu(q) = \nu(q')$. Given a predicate $p \in \mathcal{O}$, an atom $p(t_1, \dots, t_{\alpha(p)})$ is satisfied by $\nu$ if $(t_1^{\nu,\mathcal{M}}, \dots, t_{\alpha(p)}^{\nu,\mathcal{M}}) \in p^\mathcal{M}$. Finally, $\nu$ satisfies $\pi = \pi'$ of $\nu(\pi) = \nu(\pi')$. The satisfiability relation is naturally extended to Boolean combinations of atoms. Finally, assume that $\Phi$ is of the form $\exists \pi_1 \colon p_1 \xrightarrow{u_1 \mid v_1} q_1, \dots, \exists \pi_n \colon p_n \xrightarrow{u_n \mid v_n} q_n\ \mathcal{C}$, we say that $A$ satisfies $\Phi$, denoted by $A \models \Phi$, if there exists a valuation $\nu$ of $\text{Var}(\Phi)$ such that for all $i \in \{1, \dots, n\}$, $\nu(\pi_i)\colon \nu(p_i) \xrightarrow{\nu(u_i) \mid \nu(v_i)} \nu(q_i)$ and $\nu$ satisfies $\mathcal{C}$ (i.e. $\nu \models \mathcal{C}$). Given a pattern formula $\Phi$ and an automaton with outputs $A$, the *model-checking problem* consists in deciding whether $A$ satisfies $\Phi$, i.e. $A \models \Phi$.

**Example 6.1.2**

An automaton is *not* deterministic if there exists an $\varepsilon$ transition or if there exists two distinct outgoing transitions labeled with the same input letter of $\Sigma$. This can be expressed in $\mathsf{PL}[\varnothing]$ by the following formula.

$$\exists \pi \colon p \xrightarrow{e} p',\ \exists \pi_1 \colon q \xrightarrow{\sigma} q_1,\ \exists \pi_2 \colon q \xrightarrow{\sigma} q_2 \quad \big(e \in \{\varepsilon\} \wedge p \neq p'\big) \vee \big(\sigma \in \Sigma \wedge q_1 \neq q_2\big)$$

**Example 6.1.3**

An automata is *ambiguous* iff if there exists two distinct accepting runs labeled over the same input word. This can be expressed in $\mathsf{PL}[\varnothing]$ by the following formula.

$$\exists \pi_1 \colon p_1 \xrightarrow{u} q_1,\ \exists \pi_2 \colon p_2 \xrightarrow{u} q_2 \quad \pi_1 \neq \pi_2 \wedge \mathsf{init}(p_1) \wedge \mathsf{final}(q_1) \wedge \mathsf{init}(p_2) \wedge \mathsf{final}(q_2)$$

[I]To lighten the notations of quantifications of the form $\exists \pi \colon p \xrightarrow{u \mid v} q$, we may not write the input variable $u$ nor the output variable $v$ nor both when it is not used.

## 6.2 Pattern logic for finite automata

Finite automata can be seen as automata with outputs in a trivial monoid (with a single element). As the monoid is trivial, there is no need for predicates over it and so we specialize our pattern logic into $\mathsf{PL}_{\text{nfa}} \overset{\text{def}}{=} \mathsf{PL}[\varnothing]$.

### 6.2.1 Syntax, semantics and model-checking problem of $\mathsf{PL}_{\text{nfa}}$

As the generic pattern logic, formulas of the following logic for $\mathsf{NFA}$ are build $\mathrm{Var}_P$ of path variables, $\mathrm{Var}_Q$ of state variables, etc.

**Definition – Pattern logic for NFA**

The logic $\mathsf{PL}_{\text{nfa}} = \mathsf{PL}[\varnothing]$ is the set of formulas of the form

$$\begin{aligned} \Phi &::= \quad \exists \pi_1 : p_1 \xrightarrow{u_1} q_1, \dots, \exists \pi_n : p_n \xrightarrow{u_n} q_n \ \mathcal{C} \\ \mathcal{C} &::= \quad \neg\mathcal{C} \mid \mathcal{C} \vee \mathcal{C} \mid u \sqsubseteq u' \mid u \in L \mid |u| \leq |u'| \mid \mathsf{init}(s) \mid \mathsf{final}(s) \mid s = s' \mid \pi = \pi' \end{aligned}$$

where for all $i \neq j$, $\pi_i \neq \pi_j$, $L$ denote regular languages over $\Sigma$ (assumed to be represented as $\mathsf{DFA}$), $u, u' \in \{u_1, \dots, u_n\}$, $s, s' \in \{p_1, \dots, p_n, q_1, \dots, q_n\}$ and $\pi, \pi' \in \{\pi_1, \dots, \pi_n\}$.

The model-checking problem asks if a given $\mathsf{NFA}$ $A$ satisfies a given $\mathsf{PL}_{\text{nfa}}$-formula $\Phi$. Theorem 6.2.1 is proved in Section 7.2 since it involves new notions, in particular an intermediate logic equivalent to the pattern logic.

**Theorem 6.2.1**

The model-checking problem of $\mathsf{NFA}$ against formulas in $\mathsf{PL}_{\text{nfa}}$ is PSPACE-C. It is in NLOGSPACE-C when the formula is fixed.

### 6.2.2 Examples of NFA subclasses

As a yardstick to measure the expressiveness of $\mathsf{PL}_{\text{nfa}}$, we have considered the structural properties of $\mathsf{NFA}$ studied in two classical papers [WS91, AMR11]. The authors of these two papers provide a PTIME membership algorithms for $k$-ambiguity, finite ambiguity, polynomial ambiguity and exponential ambiguity (with as applications the approximation of the entropy of probabilistic automata for instance). The solutions to these membership problems follow a recurrent schema: one defines (1) a pattern that identifies the members of the class and (2) an algorithm to decide if an automaton satisfies the pattern. In fact, all these membership problems can be reduced to the model-checking problem of $\mathsf{PL}_{\text{nfa}}$ using a constant space reduction. The proof of this theorem is obtained by showing how the patterns identified in [WS91], can be succinctly and naturally encoded into (constant) $\mathsf{PL}_{\text{nfa}}$ formulas.

**Class of $k$-ambiguous automata**

An automaton is said to be *$k$-ambiguous* if it admits at most $k$ distinct accepting runs for all input word. Hence, it is *not* $k$-ambiguous iff it satisfies the following $\mathsf{PL}_{\text{nfa}}$ formula:

$$\begin{array}{c} \exists \pi_1 : p_1 \xrightarrow{u} q_1 \\ \vdots \\ \exists \pi_{k+1} : p_{k+1} \xrightarrow{u} q_{k+1} \end{array} \quad \left( \bigwedge_{1 \leq i \leq k+1} \mathsf{init}(p_i) \wedge \mathsf{final}(q_i) \right) \wedge \left( \bigwedge_{1 \leq i < j \leq k+1} \pi_i \neq \pi_j \right)$$

We have only depicted the input word variables as the output variables are useless.

**Class of finitely ambiguous automata**

An automaton is said to be *finitely ambiguous* if there exists $k \in \mathbb{N}$ such that it is $k$-ambiguous. As shown in [WS91], an automaton has a finite ambiguity iff it does not satisfy the pattern of Figure 6.1, expressible by the following $\mathsf{PL}_{\text{nfa}}$ formula:

$$\begin{array}{c} \exists \pi_I : q_I \rightarrow q_1,\ \exists \pi : q_1 \xrightarrow{u} q_2,\ \exists \pi_F : q_2 \rightarrow q_F \\ \exists \pi_1 : q_1 \xrightarrow{u} q_1,\ \exists \pi_2 : q_2 \xrightarrow{u} q_2 \end{array} \quad \mathsf{init}(q_I) \wedge q_1 \neq q_2 \wedge \mathsf{final}(q_F)$$

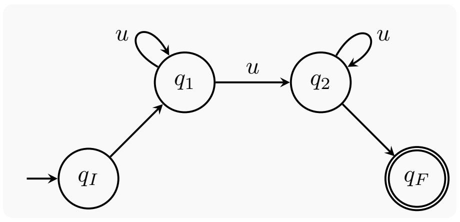


Figure 6.1: Pattern of non-finite ambiguity

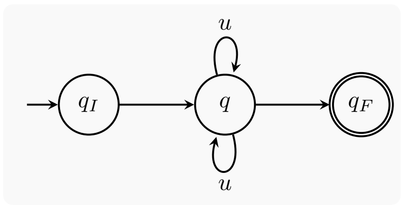


Figure 6.2: Pattern of non-polynomial ambiguity

**Class of polynomially ambiguous automata**

An automaton is said to be *polynomially ambiguous* if the number of accepting runs is at most polynomial in the length of the input word. Formally, there exists $c \in \mathbb{N}$ such that for all $u \in \Sigma^*$, the number of accepting runs is upper bounded by $|u|^c$. As shown in [WS91], an automaton with output has a polynomial ambiguity iff it does not satisfy the pattern[II] of Figure 6.2 expressible by the $\mathsf{PL}_{\text{nfa}}$ formula:

$$\exists \pi_I \colon q_I \to q, \quad \begin{matrix} \exists \pi_1 \colon q \xrightarrow{u} q, \\ \exists \pi_2 \colon q \xrightarrow{u} q, \end{matrix} \quad \exists \pi_F \colon q \to q_F \quad \mathsf{init}(q_I) \wedge \pi_1 \neq \pi_2 \wedge \mathsf{final}(q_F)$$

**Class of exponentially ambiguous automata**

An automaton is said to be *exponentially ambiguous* if the number of accepting runs is at most exponential in the length of the input word.

We claim that an automaton is exponentially ambiguous iff it does not contain an accessible and co-accessible cycle on the empty word. The left-to-right direction is proved by contrapositive. Suppose that there exists such cycle denoted $\pi_c$ on a state $q$. Since $\pi_c$ is accessible, there exists $\pi_I$ a run from some initial state to $q$ over $u_1$ and since $\pi_c$ is co-accessible, there exists also $\pi_F$ a run from $q$ to some final state over $u_2$. For all $n \in \mathbb{N}$ we can define $\pi_n \overset{\text{def}}{=} \pi_I \pi_c^n \pi_F$ such that $\pi_n \neq \pi_m$ for all $n \neq m$. For the right-to-left direction, assume that the automaton does not have any $\varepsilon$-cycle. Each run cannot have more than $\mathtt{card}(Q)$ consecutive $\varepsilon$-transitions otherwise that would contradict the non-existence of an $\varepsilon$-cycle. So, for all word $u \in \Sigma^*$, each run $u$ have a length bounded by $|u|\mathtt{card}(Q)$ which implies that there are at most $\mathtt{card}(Q)^{|u|\mathtt{card}(Q)}$ runs over $u$.

Hence, an automaton is exponentially ambiguous iff it does not satisfy the following $\mathsf{PL}_{\text{nfa}}$ formula:

$$\exists \pi_I \colon q_I \to q, \ \exists \pi_c \colon q \xrightarrow{e} q, \ \exists \pi_F \colon q \to q_F \quad \mathsf{init}(q_I) \wedge \mathsf{final}(q_F) \wedge e \in \{\varepsilon\}$$

**Corollary**

Let $k \in \mathbb{N}$ be a constant. The membership problem to the classes of $k$-ambiguous, finitely ambiguous, polynomially ambiguous and exponentially ambiguous $\mathsf{NFA}$ is in NLogSpace.

**Proof** The proof goes by a constant space reduction to the model-checking problem of $\mathsf{PL}_{\text{nfa}}$. For each membership problem, our reduction copies (in constant space) the $\mathsf{NFA}$ and considers the model-checking for this $\mathsf{NFA}$ against a constant $\mathsf{PL}_{\text{nfa}}$ formula (one for each class). So, NLogSpace membership comes as a corollary of Theorem 6.2.1. ♦

## 6.3 Pattern logic for transducers

Transducers are automata with outputs in a free monoid $(\Gamma^*, \cdot, \varepsilon)$ and therefore define subsets of $\Sigma^* \times \Gamma^*$. Since our general pattern logic can test for output equalities (by repeating twice an output variable in the quantification part), the model-checking is easily shown to be undecidable by encoding Post Correspondence Problem.

[II] As in the formulas, we have only depicted input variables that matter.

**Theorem**

The model-checking problem of transducers against formulas in $\mathsf{PL}[\varnothing]$ is undecidable.

**Proof** We encode the Post Correspondence problem (PCP here after). Given a set $S$ of dominoes over some (non-unary) alphabet $\Sigma$, defined as $S = \{(u_1, v_1), \dots, (u_n, v_n)\}$, we construct the transducer $T_{\text{pcp}}$ of Figure 6.3, which defines the following subset of $\{1, \dots, n\}^* \times \Sigma^*$:

$$\forall k \geq 0 \qquad \bigcup_{i_1 \dots i_k \in \{1,\dots,n\}^k} \{(i_1 \dots i_k, u_{i_1} \dots u_{i_k})\} \cup \{(i_1 \dots i_k, v_{i_1} \dots v_{i_k})\}$$

The solution of PCP is a finite sequence $x_1 \dots x_N$ where each $x_i$ are in $\{1, \dots, n\}$ such that $N > 0$ and $u_{x_1} \dots u_{x_N} = v_{x_1} \dots v_{x_N}$. The set $S$ admits $x_1 \dots x_N$ has solution iff there exists two distinct accepting runs over the non-empty input word $x_1 \dots x_N$ which have the same output. Hence, $S$ has a solution iff the $\mathsf{PL}[\varnothing]$ formula $\exists \pi \colon q \xrightarrow{u \mid v} q, \exists \pi' \colon q' \xrightarrow{u \mid v} q', q \neq q' \wedge u \notin \{\varepsilon\}$ is satisfied by $T_{\text{pcp}}$. ♦

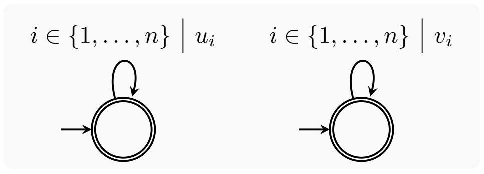
$(T_{\text{pcp}})$ 
$i \in \{1, \dots, n\} \mid u_i$
$i \in \{1, \dots, n\} \mid v_i$

Figure 6.3: The transducer $T_{\text{pcp}}$ which is non-deterministic

To obtain a decidable logic for transducers, we exclude equality tests on the output words in the logic. However, as we will see, we can still have inequality tests $\neq$ as long as they do not occur under an odd number of negations in the formula. We also allow to test (non) membership of output word concatenations to a regular language, as well as comparison of output word concatenations w.r.t. their length.

## 6.3.1 Syntax, semantics and model-checking problem of $\mathsf{PL}_{\text{trans}}$

We have shown that having the equality (and more generally $\sqsubseteq$) predicate for outputs for transducer yields an undecidable model-checking. Here, we define $\mathsf{PL}_{\text{trans}}$ as the instance of the positive fragment of the pattern logic $\mathsf{PL}_{\text{trans}} \stackrel{\text{def}}{=} \mathsf{PL}^+[\{\not\sqsubseteq, \in N, \notin N, \leq, <\}]$ for all regular language $N$ over $\Gamma$ (assumed to be represented by some $\mathsf{DFA}$).

**Definition – Pattern logic for transducers**

The logic $\mathsf{PL}_{\text{trans}}$ is the set of formulas of the form

$$\begin{array}{lll} \Phi & ::= & \exists \pi_1 \colon p_1 \xrightarrow{u_1 \mid v_1} q_1, \dots, \exists \pi_n \colon p_n \xrightarrow{u_n \mid v_n} q_n \ \mathcal{C} \\ \mathcal{C} & ::= & \neg \mathcal{C} \mid \mathcal{C} \vee \mathcal{C} \mid u \sqsubseteq u' \mid u \in L \mid |u| \leq |u'| \mid \mathsf{init}(s) \mid \mathsf{final}(s) \mid s = s' \mid \pi = \pi' \mid \\ & & t \not\sqsubseteq t' \mid t \in N \mid |t| \leq |t'| \end{array}$$

where for all $1 \leq i < j \leq n$, $\pi_i \neq \pi_j$ and $v_i \neq v_j$ (no implicit output equality tests), $L$ (resp. $N$) denote regular languages over $\Sigma$ (resp. $\Gamma$) (assumed to be represented as $\mathsf{DFA}$), $u, u' \in \{u_1, \dots, u_n\}$, $s, s' \in \{p_1, \dots, p_n, q_1, \dots, q_n\}$, $t, t' \in \text{Terms}(\{v_1, \dots, v_n\}, \cdot, \varepsilon)$, $\pi, \pi' \in \{\pi_1, \dots, \pi_n\}$, and $t \not\sqsubseteq t'$ does not occur under an odd number of negations.

For convenience, we define the syntactic sugar for non-equality of outputs $t \neq t' \stackrel{\text{def}}{=} t \not\sqsubseteq t' \vee t' \not\sqsubseteq t$. Also, to express that words $t, t'$ mismatch together, that is when there exists a position $i$ such that $t[i] \neq t'[i]$ we define $t \nsim t' \stackrel{\text{def}}{=} t \not\sqsubseteq t' \wedge t' \not\sqsubseteq t$.

**Theorem 6.3.1**

The model checking of transducers against formulas in $\mathsf{PL}_{\text{trans}}$ is PSPACE-C. It is in NLOGSPACE-C when the formula is fixed.

Theorem 6.3.1 is proved in Section 7.3.

### 6.3.2 Using an alternative delay in $\mathsf{PL}_{\text{trans}}$

Many properties of automata with outputs over a group are based on a notion of delays between output values. In the case of transducers, this is formalized for any words $v_1, v_2$ by $\text{delay}(v_1, v_2) = (w_1, w_2)$ such that $v_1 = vw_1$ and $v_2 = vw_2$ where $v$ is the longest common prefix of $v_1$ and $v_2$. It is for instance used to characterize the sequential functions, i.e. functions definable by input deterministic transducer.

An interesting property of the delay is that, if by adding a pair of suffixes to the pair of output words the delay change, then we can highlight two witness which differ arbitrarily by iterating the same pair of suffixes. Let $\mathsf{del}_{\neq}(v_1, v_1', v_2, v_2') \stackrel{\text{def}}{=} \text{delay}(v_1, v_2) \neq \text{delay}(v_1v_1', v_2v_2')$. However, this notion of the delay is not directly expressible in our logic. We show here that an equivalent notion can be used, that we call Sdelay (for "simpler" delay). We define $\mathsf{Sdel}_{\neq}(v_1, v_1', v_2, v_2') \stackrel{\text{def}}{=} |v_1'| \neq |v_2'| \vee \big(v_1'v_2' \neq \varepsilon \wedge v_1 \not\sim v_2\big)$. Lemma 6.3.2 shows that for any word $v_1, v_1', v_2, v_2'$, if $\mathsf{Sdel}_{\neq}(v_1, v_1', v_2, v_2')$ holds then $\mathsf{del}_{\neq}(v_1, v_1', v_2, v_2')$ holds. The converse is not true in general but, by Lemma 6.3.3 if $\mathsf{del}_{\neq}(v_1, v_1', v_2, v_2')$ holds then $\mathsf{Sdel}_{\neq}(v_1(v_1')^i, v_1', v_2(v_2')^i, v_2')$ holds for any $i \in \mathbb{N}$ sufficiently big. Note that, the predicate $\mathsf{Sdel}_{\neq}$ is definable in $\mathsf{PL}_{\text{trans}}$.

The structural property characterizing the class of multi-sequential functions from [CS86, JF18], uses the predicate $\mathsf{Sdel}_{\neq}$ and then can be directly encoded in $\mathsf{PL}_{\text{trans}}$ as presented in the next subsection. It is not the case of the twinning property from [BCPS03, Cho77b, WK95] (characterizing the class of sequential functions). However, using predicate $\mathsf{Sdel}_{\neq}$ instead of $\mathsf{del}_{\neq}$ in the definitions of the twinning property give an equivalent characterizations.

**Lemma 6.3.2**

For all $v, x, w, y \in \Sigma^*$ we have the following property:

$$\mathsf{Sdel}_{\neq}(v, x, w, y) \implies \mathsf{del}_{\neq}(v, x, w, y)$$

**Proof** If $xy \neq \varepsilon \wedge v \not\sim w$ from the $\mathsf{Sdel}_{\neq}$ hypothesis, then at least one of $x, y$ is non-empty. Thus by iterating the loop once, the delay will accumulate after the mismatch between $v$ and $w$. In other words, we have that $\text{delay}(v, w) \neq \text{delay}(vx, wy)$.
Else if $|x| \neq |y| \wedge v \sqsubseteq w$ the proof goes by the way of absurd. Note that the case where $w \sqsubseteq v$ can be proved similarly. Suppose that $\text{delay}(v, w) = (\varepsilon, z) = \text{delay}(vx, wy)$ for some $z$. Then, we have that $x \sqsubseteq zy$. Consider the following two cases. If $|x| \leq |z|$, we fix $z = xz'$. Then $(\varepsilon, z) = \text{delay}(x, zy) = (\varepsilon, z'y)$. Hence, $z = z'y$, which is a contradiction since $|xz'| \neq |z'y|$. Otherwise if $|x| > |z|$, we fix $y = y_1y_2$ and $x = zy_1$. Then $(\varepsilon, z) = \text{delay}(x, zy) = (\varepsilon, y_2)$. Hence, $z = y_2$, which is a contradiction since $|zy_1| \neq |y_1y_2|$. ♦

**Lemma 6.3.3**

For all $v, x, w, y \in \Sigma^*$, there exists $N \in \mathbb{N}$ such that:

$$\forall i \geq N \qquad \mathsf{del}_{\neq}(v, x, w, y) \implies \mathsf{Sdel}_{\neq}(v(x)^i, x, w(y)^i, y)$$

**Proof** If $|x| \neq |y|$ then the statement trivially holds. Otherwise, we have that $xy \neq \varepsilon$ from the $\mathsf{del}_{\neq}$ hypothesis, which implies that both $x, y$ are non-empty assuming $|x| = |y|$. Suppose by the way of absurd that $\mathsf{del}_{\neq}(v, x, w, y)$ holds and there is no mismatch between $vx^i$ and $wy^i$ for all $i$. This implies that by iterating loops an infinite number of times we obtain the equality $vx^\omega = wy^\omega$. W.l.o.g. assume that $v$ is a prefix of $w$, i.e. $w = vz$ for some $z$ (the other case is symmetric). Let $k \in \mathbb{N}$ taken such that $|x^k| > |z|$ and $|x^{k-1}| \leq |z|$. Note that $k$ exists since $x \neq \varepsilon$. Then obtain the decomposition $x = x_1x_2$ from $x^{k-1}x_1 = z$ and the decomposition $y = y_1y_2$ where $|y_1| = |x_2|$ from $x^k = zy_1$.

Using $x^\omega = zy^\omega$ and $|y| = |x|$, we get $x^{k+1} = zy_1y_2y_1$. In particular $x = x_1x_2 = y_2y_1$ and $y = y_1y_2 = x_2x_1$. Therefore, $\text{delay}(v,w) = \text{delay}(v,vz) = (\varepsilon, z) = (\varepsilon, x^{k-1}x_1)$. On the other hand, we have that $\text{delay}(vx,wy) = \text{delay}(x,zy) = \text{delay}(x, x^{k-1}x_1y) = \text{delay}(x, x^{k-1}x_1x_2x_1) = \text{delay}(\varepsilon, x^{k-1}x_1) = \text{delay}(v,w)$ contradicting $\mathsf{del}_{\neq}$ hypothesis. Hence, there exists $i$ such that there is a mismatch between $vx^i$ and $wy^i$. To conclude we remark that if $vx^i$ and $wy^i$ mismatch at some position, then $vx^{i+1}$ and $wy^{i+1}$ mismatch as well (at the same position). ♦

In [DJRV17], the authors introduce the *branching twinning property* (characterizing the class of $k$-sequential functions) which the twinning property is a special case. The branching twinning property is defined with the use of the $\mathsf{del}_{\neq}$ predicate and we prove here that $\mathsf{Sdel}_{\neq}$ allows us to define an equivalent characterization. Note that, while the twinning property is defined with a predicate $\mathsf{del}_{\neq}$ over words, its generalization is defined with such predicate over terms of words. Lemma 6.3.3 cannot handle terms and thus is not sufficient to show that the branching twinning property is expressible in $\mathsf{PL}_{\text{trans}}$. We now generalize Lemma 6.3.3.

**Lemma 6.3.4**

For all $v_1,\dots,v_m,w_1,\dots,w_m,x_1,\dots,x_m,y_1,\dots,y_m \in \Sigma^*$, there exists $N \in \mathbb{N}$ such that for all $i \geq N$ we have that:

$$\mathsf{del}_{\neq}(v_1\dots v_m, x_m, w_1\dots w_m, y_m) \qquad \text{Hypothesis : } H_1$$

$$\wedge$$

$$\forall 1 \leq \ell < m,\ \mathsf{del}_{=}(v_1\dots v_\ell, x_\ell, w_1\dots w_\ell, y_\ell) \qquad \text{Hypothesis : } H_2$$

$$\Downarrow$$

$$\mathsf{Sdel}_{\neq}(v_1(x_1)^i\dots v_m(x_m)^i, x_m, w_1(y_1)^i\dots w_m(y_m)^i, y_m)$$

**Proof** We fix $V_j = v_1x_1^j\dots v_{m-1}x_{m-1}^jv_m$, $X = x_m$ and $W_j = w_1y_1^j\dots w_{k-1}y_{k-1}^jw_m$, $Y = y_m$ for all $j \in \mathbb{N}$. If $|X| \neq |Y|$ then the statement trivially holds. Otherwise, we can show the following properties:

1. For all $1 \leq j < m$, we that that $|x_j| = |y_j|$ by the contrapositive of Lemma 6.3.2 applied on $H_2$.
2. We have that $XY \neq \varepsilon$ due to $H_1$, which implies that both $x_m$ and $y_m$ are non-empty since $|X| = |Y|$.
3. Item (1) implies that $|V_jX^i| - |W_jY^i| = |V_0| - |W_0|$ for all $i,j \in \mathbb{N}$. We define $S = |V_0| - |W_0|$.
4. Lemma 6.3.3 applied on $\text{delay}(V_0,W_0) \neq \text{delay}(V_0X, W_0Y)$ ensures that, for some $M$ there is a mismatch between $V_0X^i$ and $W_0Y^i$ for all $i \geq M$ since $|X| = |Y|$ and $XY \neq \varepsilon$.

In the rest of the proof, we suppose that $S \geq 0$ (the case $|W_0| \geq |V_0|$ can be treated similarly). By (3) we have that $|V_jX^i| = |W_jY^i| + S$ for all $i,j \in N$. In order to deal with words of same length we define $Z$ as the suffix of $X^i$ such that $|Z| = S$. Note that $Z$ is well defined if $i \geq S$ since $|X| > 0$ by (2). Thus, we obtain that $V_0X^i \neq W_0Y^iZ$ for all $i \geq \max\{M,S\}$ by (3). Applying the contrapositive of Theorem 4.3. of [Saa15], if $V_jX^i \neq W_jY^iZ$ for some $j \in \mathbb{N}$ then there exists $M'$ such that $V_jX^i \neq W_jY^iZ$ for all $j \geq M'$. Hence, $V_iX^i \neq W_iY^iZ$ for each $i \geq \max\{M,M',S\}$. Due to $|V_iX^i| = |W_iY^iZ|$, the inequality holds by mismatching. Finally, since $Z$ is suffix of $X^i$, there is also a mismatch between $V_iX^i$ and $W_iY^i$ for all $i \geq \max\{M,M',S\}$ which concludes the proof. ♦

### 6.3.3 Examples of transducer subclasses

We review some of the main transducer subclasses studied in the literature. We refer the reader to the mentioned references for the formal definitions. As for the $\mathsf{NFA}$ subclasses of the previous section, deciding them usually goes in two steps: (1) identify a structural pattern characterizing the property, (2) decide whether such a pattern is satisfied by a given transducer.

**Class of sequential transducers**

A transducer is said to be *sequential* if its function can be defined by an input deterministic transducer. As shown in [BCPS03, Cho77b, WK95], a transducer is sequential iff it satisfies the twinning property iff it does not satisfy the pattern of Figure 6.4) which is equivalent to the negation of the following $\mathsf{PL}_{\text{trans}}$ formula, thanks to Lemmas 6.3.2 and 6.3.3:

$$\begin{array}{l}\exists\pi_1\colon q_1 \xrightarrow{u|v_1} p_1,\ \exists\pi_1'\colon p_1 \xrightarrow{u'|v_1'} p_1,\ \exists\pi_1''\colon p_1 \xrightarrow{u''|v_1''} r_1\\ \exists\pi_2\colon q_2 \xrightarrow{u|v_2} p_2,\ \exists\pi_2'\colon p_2 \xrightarrow{u'|v_2'} p_2,\ \exists\pi_2''\colon p_2 \xrightarrow{u''|v_2''} r_2\end{array} \bigwedge \begin{cases}\mathsf{init}(q_1)\wedge\mathsf{final}(r_1)\\ \mathsf{init}(q_2)\wedge\mathsf{final}(r_2)\\ \mathsf{Sdel}_{\neq}(v_1,v_1',v_2,v_2')\end{cases}$$

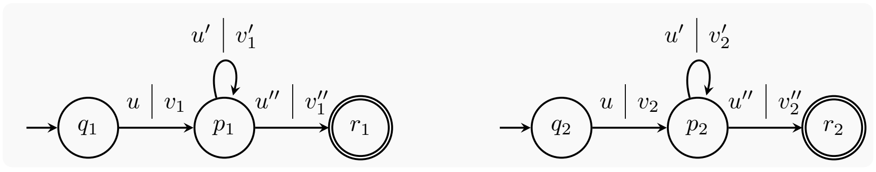


Figure 6.4: Pattern of the twinning property, where $\text{delay}(v_1, v_2) \neq \text{delay}(v_1v_1', v_2v_2')$

**Class of $k$-sequential transducers**

A transducer is said to be *$k$-sequential* if its function can be defined by a disjoint union of $k$ sequential transducers [DJRV17]. As shown in [DJRV17], a transducer is $k$-sequential iff it satisfies the branching twinning property iff it does not satisfy the pattern of Figure 6.5 which is equivalent to the negation of the following $\mathsf{PL}_{\text{trans}}$ formula.

$$\begin{array}{l}\exists_{i=1}^{k}\pi_{i,0}\colon q_{i-1,0} \xrightarrow{u_{i,0}|v_{i,0}} q_{i,0}\\ \qquad\vdots\\ \exists_{i=1}^{k}\pi_{i,k}\colon q_{i-1,k} \xrightarrow{u_{i,k}|v_{i,k}} q_{i,k}\\ \exists_{i=1}^{k}\pi_{i,0}'\colon q_{i,0} \xrightarrow{u_{i,0}'|v_{i,0}'} q_{i,0}\\ \qquad\vdots\\ \exists_{i=1}^{k}\pi_{i,k}'\colon q_{i,k} \xrightarrow{u_{i,k}'|v_{i,k}'} q_{i,k}\end{array} \quad \bigwedge_{j\neq j'}\bigvee_{1\leq m\leq k}\bigwedge\begin{cases}\bigwedge_{1\leq\ell\leq m} u_{\ell,j}=u_{\ell,j'}\wedge u_{\ell,j}'=u_{\ell,j'}'\\ \mathsf{Sdel}_{\neq}(v_{1,j}...v_{m,j},v_{m,j}',v_{1,j'}...v_{m,j'},v_{m,j'}')\end{cases}$$

Lemma 6.3.2 ensures that this $\mathsf{PL}_{\text{trans}}$ formula captures the complement of the class of $k$-sequential transducers. We show now that any transducer which satisfies this $\mathsf{PL}_{\text{trans}}$ formula cannot be $k$-sequential. Consider a transducer $T$ which does not satisfy the branching twinning property of order $k$, i.e. it satisfies the pattern of Figure 6.5. Then we have that:

$$\bigwedge_{j\neq j'}\bigvee_{1\leq m\leq k}\bigwedge\begin{cases}\bigwedge_{1\leq\ell\leq m} u_{\ell,j}=u_{\ell,j'}\wedge u_{\ell,j}'=u_{\ell,j'}'\\ \mathsf{del}_{\neq}(v_{1,j}\dots v_{m,j},v_{m,j}',v_{1,j'}\dots v_{m,j'},v_{m,j'}')\end{cases}$$

For all $0\leq j$ and $j'\leq k$, we choose $m$ minimal, i.e. $m$ which verifies that:

$$\bigwedge_{1\leq\ell<m}\mathsf{del}_{=}(v_{1,j}\dots v_{\ell,j},v_{\ell,j}',v_{1,j'}\dots v_{\ell,j'},v_{\ell,j'}')$$

By Lemma 6.3.4 there exists $i\in\mathbb{N}$ sufficiently big to ensure:

$$\bigwedge_{j\neq j'}\bigvee_{1\leq m\leq k}\bigwedge\begin{cases}\bigwedge_{1\leq\ell\leq m} u_{\ell,j}(u_{\ell,j}')^i=u_{\ell,j'}(u_{\ell,j'}')^i\wedge u_{\ell,j}'=u_{\ell,j'}'\\ \mathsf{Sdel}_{\neq}(v_1(x_1)^i\dots v_m(x_m)^i,x_m,w_1(y_1)^i\dots w_m(y_m)^i,y_m)\end{cases}$$

Since, our $\mathsf{PL}_{\text{trans}}$ formula quantify the words existentially, $T$ would be able to satisfies it.

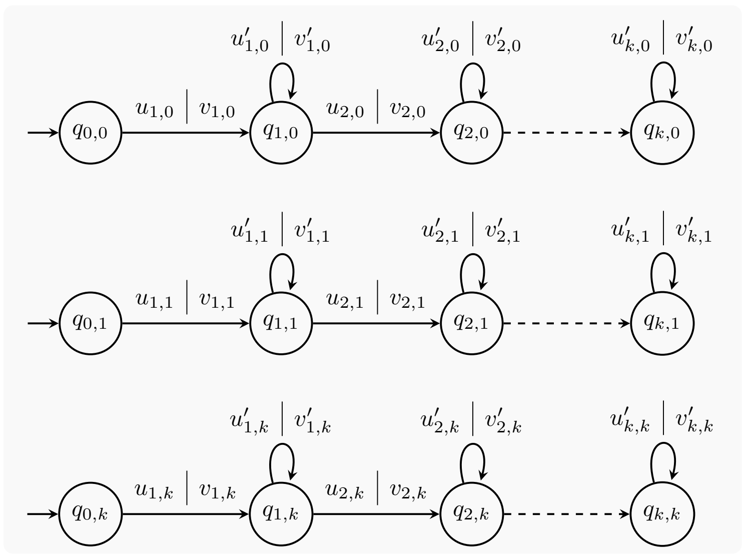


Figure 6.5: Pattern of the branching twinning property property where there are $j \neq j'$ such that for all $m \in \{1, \ldots, k\}$, if for every $1 \leq \ell \leq m$, we have $u_{\ell,j} = u_{\ell,j'}$ and $u'_{\ell,j} = u'_{\ell,j'}$, then we have $\text{delay}(v_{1,j} \ldots v_{m,j}, v_{1,j'} \ldots v_{m,j'}) \neq \text{delay}(v_{1,j} \ldots v_{m,j} v'_{m,j}, v_{1,j'} \ldots v_{m,j'} v'_{m,j'})$

**Class of multi-sequential transducers**

A transducer is said to be *multi-sequential* if there exists $k$ such that its function can be defined by a disjoint union of $k$ sequential transducers. As shown in [CS86, JF18] a transducer is multi-sequential iff it does not satisfy the pattern of Figure 6.6, called the fork property. This pattern is expressed by the $\mathsf{PL}_{\text{trans}}$ formula:

$$\begin{array}{l} \exists \pi_{0,1} \colon q_0 \to q_1,\ \exists \pi_{1,1} \colon q_1 \xrightarrow{u_1 \mid v_1} q_1,\ \exists \pi'_{1,1} \colon q_1 \xrightarrow{u_2 \mid v_2} q_1 \\ \exists \pi_{2,3} \colon q_2 \to q_3,\ \exists \pi_{1,2} \colon q_1 \xrightarrow{u_1 \mid v'_1} q_2,\ \exists \pi_{2,2} \colon q_2 \xrightarrow{u_2 \mid v'_2} q_2 \end{array} \bigwedge \begin{cases} \mathsf{init}(q_0) \wedge \mathsf{final}(q_3) \\ \mathsf{Sdel}_{\neq}(v_1, v'_1, v_2, v'_2) \end{cases}$$

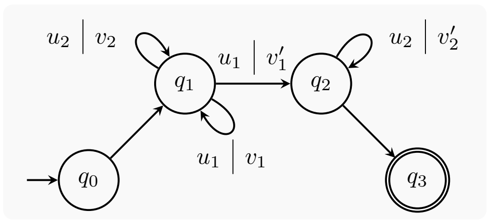


Figure 6.6: Pattern of the fork property where $\mathsf{Sdel}_{\neq}(v_1, v'_1, v_2, v'_2)$ holds

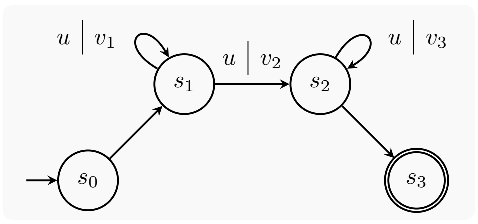


Figure 6.7: Pattern of the dumbbell computation where $s_1 \neq s_2$ and $v_1 v_2 \neq v_2 v_3$

**Class of $k$-valued transducers**

An automaton is said to be *$k$-valued* if it admits at most $k$ distinct output values for all input word, see [GI83, SdS10]. We have already shown in the Introduction that non-functionality is expressible in $\mathsf{PL}_{\text{trans}}$. This can be generalized to non-$k$-valuedness as follows:

$$\begin{array}{c} \exists \pi_1 \colon p_1 \xrightarrow{u \mid v_1} q_1 \\ \vdots \\ \exists \pi_{k+1} \colon p_{k+1} \xrightarrow{u \mid v_{k+1}} q_{k+1} \end{array} \quad \left( \bigwedge_{i=1}^{k+1} \mathsf{init}(p_i) \wedge \mathsf{final}(q_i) \right) \wedge \left( \bigwedge_{1 \leq i < j \leq k+1} v_i \neq v_j \right)$$

**Class of finite valued transducers**

A transducer is said to be *finite valued* if there exists $k$ such that any input word has at most $k$ distinct output words [Web90, Web93, SdS08]. As shown in [Web90, SdS08], a transducer is finite valued iff it does not satisfy any of the following three patterns.

- Figure 6.8 where $v_1 \neq v_2$, expressed in $\mathsf{PL}_{\text{trans}}$ by:

$$\begin{array}{l} \exists\pi_0^1 : q_0 \longrightarrow q_1,\ \exists\pi_1^1 : q_1 \xrightarrow{u|v_1} q_1,\ \exists\pi_2^2 : q_2 \xrightarrow{u|v_2} q_2 \\ \exists\pi_0^2 : q_0 \longrightarrow q_2,\ \exists\pi_1^3 : q_1 \longrightarrow q_3,\ \exists\pi_2^3 : q_2 \longrightarrow q_3 \end{array} \bigwedge \begin{cases} \mathsf{init}(q_0) \\ \mathsf{final}(q_3) \\ v_1 \neq v_2 \end{cases}$$

- Figure 6.7 where $s_1 \neq s_2$ and $v_1v_2 \neq v_2v_3$, expressed in $\mathsf{PL}_{\text{trans}}$ by:

$$\begin{array}{r} \exists\pi_0^1 : s_0 \longrightarrow s_1,\ \exists\pi_1^2 : s_1 \xrightarrow{u|v_2} s_2,\ \exists\pi_2^3 : s_2 \longrightarrow s_3 \\ \exists\pi_1^1 : s_1 \xrightarrow{u|v_1} s_1,\ \exists\pi_2^2 : s_2 \xrightarrow{u|v_3} s_2 \end{array} \bigwedge \begin{cases} \mathsf{init}(s_0) \\ \mathsf{final}(s_3) \\ s_1 \neq s_2 \\ v_1v_2 \neq v_2v_3 \end{cases}$$

- Figure 6.9 where $|v_1| \neq |v_2|$, expressed in $\mathsf{PL}_{\text{trans}}$ by:

$$\begin{array}{r} \exists\pi_0^1 : r_0 \to r_1,\ \exists\pi_1^1 : r_1 \xrightarrow{u_1|v_1} r_2,\ \exists\pi_2^2 : r_2 \xrightarrow{u_2} r_2,\ \exists\pi_2^2 : r_2 \xrightarrow{u_3} r_1 \\ \exists\pi_4^2 : r_4 \to r_6,\ \exists\pi_1^1 : r_1 \xrightarrow{u_1} r_3,\ \exists\pi_3^1 : r_3 \xrightarrow{u_2|v_2} r_3,\ \exists\pi_3^2 : r_3 \xrightarrow{u_3} r_4 \\ \exists\pi_4^2 : r_4 \xrightarrow{u_1} r_5,\ \exists\pi_5^1 : r_5 \xrightarrow{u_2} r_5,\ \exists\pi_5^1 : r_5 \xrightarrow{u_3} r_4 \end{array} \bigwedge \begin{cases} \mathsf{init}(r_0) \\ \mathsf{final}(r_6) \\ |v_1| \neq |v_2| \end{cases}$$

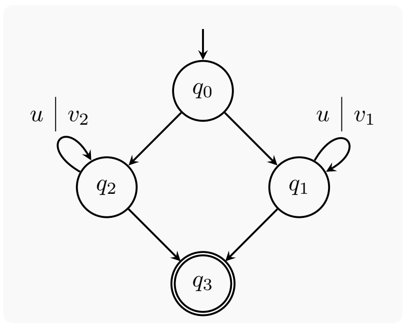


Figure 6.8: Pattern called co-terminal circuits where $v_1 \neq v_2$

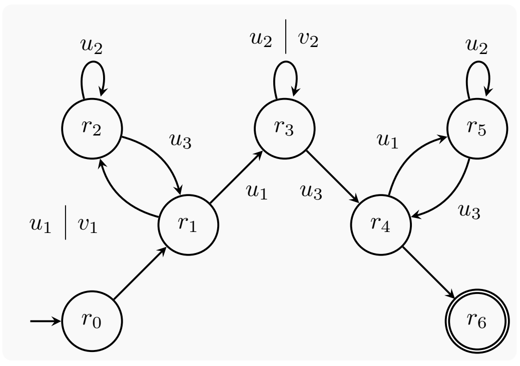


Figure 6.9: Pattern called the W computation where $|v_1| \neq |v_2|$

**Corollary 6.3.5**

Let $k \in \mathbb{N}$ be a constant. The membership problem of transducers to the classes of $k$-sequential, multi-sequential, $k$-valued and finite-valued transducers is decidable in NLogSpace.

**Proof** The proof goes by a constant space reduction to the model-checking problem of $\mathsf{PL}_{\text{trans}}$. Then, the obtained formulas are constant (as long as $k$ is fixed). So, NLogSpace membership comes as a corollary of Theorem 6.3.1. ♦

## 6.4 Pattern logic for sum-automata

We remind the reader that sum-automata are automata with outputs in the monoid $(\mathbb{Z}, +, 0)$ and therefore define subsets of $\Sigma^* \times \mathbb{Z}$. We consider in this section two logics for expressing structural properties of sum-automata: the logic $\mathsf{PL}_{\text{sum}}$ which is obtained as $\mathsf{PL}[\{\leq\}]$ where the output predicate $\leq$ is interpreted by the natural total order over integers, and a subset of this logic $\mathsf{PL}_{\text{sum}}^{\neq}$ obtained as $\mathsf{PL}^{+}[\{\neq\}]$ where the predicate $\neq$ never appears in the scope of an odd number of negations (to avoid the expressibility of the equality predicate). We show that the fragment $\mathsf{PL}_{\text{sum}}^{\neq}$ has better complexity results.

### 6.4.1 Syntax, semantics and model-checking problem of $\mathsf{PL}_{\text{sum}}$ and $\mathsf{PL}^{\neq}_{\text{sum}}$

**Definition – Pattern logic for sum-automata**

The logic $\mathsf{PL}_{\text{sum}}$ is the set of formulas of the form

$$\begin{array}{lll} \Phi & ::= & \exists \pi_1 : p_1 \xrightarrow{u_1|v_1} q_1, \dots, \exists \pi_n : p_n \xrightarrow{u_n|v_n} q_n \ \mathcal{C} \\ \mathcal{C} & ::= & \neg\mathcal{C} \mid \mathcal{C} \vee \mathcal{C} \mid u \sqsubseteq u' \mid u \in L \mid |u| \leq |u'| \mid \mathsf{init}(s) \mid \mathsf{final}(s) \mid s = s' \mid \pi = \pi' \mid \\ & & t \leq t' \end{array}$$

where for all $1 \leq i < j \leq n$, $\pi_i \neq \pi_j$, $L$ denote regular languages over $\Sigma$ (assumed to be represented as $\mathsf{DFA}$), $u, u' \in \{u_1, \dots, u_n\}$, $s, s' \in \{p_1, \dots, p_n, q_1, \dots, q_n\}$, $t, t' \in \text{Terms}(\{v_1, \dots, v_n\}, \cdot, \varepsilon)$ and $\pi, \pi' \in \{\pi_1, \dots, \pi_n\}$.

The logic $\mathsf{PL}^{\neq}_{\text{sum}}$ is defined as above but the constraint $t \leq t'$ is replaced by $t \neq t'$ and this constraint does not occur under an odd number of negations, and moreover $v_i \neq v_j$ for all $1 \leq i < j \leq n$ (no implicit output equality tests).

**Theorem 6.4.1**

The model checking of sum-automata against formulas in $\mathsf{PL}_{\text{sum}}$ is PSpace-C. It is NP-C when the formula is fixed, and NLogSpace-C if in addition weights of the automaton belong to $\{0, 1\}$.

**Theorem 6.4.2**

The model checking of sum-automata against formulas in $\mathsf{PL}^{\neq}_{\text{sum}}$ is PSpace-C. It is NLogSpace-C when the formula is fixed (even if the values of the automaton are encoded in binary).

Theorems 6.4.1 and 6.4.2 are proved in Section 7.4.

### 6.4.2 Example of sum-automata subclasses

We review here some of the main sum-automata subclasses of the literature that are decidable in PTime. The classes of $k$-valued, studied for instance in [FGR14, FGR15], can be expressed in $\mathsf{PL}^{\neq}_{\text{sum}}$ similarly as for transducers. For $k$-sequentiality we can observe that we have $\text{delay}(v_1, v_2) \neq \text{delay}(v_1 v_1', v_2 v_2')$ iff $v_1' \neq v_2'$, thanks to commutativity of sum.

**Corollary**

Let $k \in \mathbb{N}$ be a constant. The membership problem of sum-automata to the classes of $k$-sequential and $k$-valued sum-automata is decidable in NLogSpace.

**Proof** The proof goes by a constant space reduction to the model-checking problem of $\mathsf{PL}^{\neq}_{\text{sum}}$. Then, the obtained formulas are constant (as long as $k$ is fixed). So, NLogSpace membership comes as a corollary of Theorem 6.4.2. ⧫

## 6.5 Extensions

The logics we have presented can be extended in two ways by keeping the NLogSpace complexity results, for all the output monoids we have considered. The first extension allows to use arbitrary states predicate (where evaluation can be done in NLogSpace). This is useful for instance to express properties with equivalence relation on states. The second extension is adding universal state quantifiers before the formula. This does not change the complexity, and allows for instance to express properties such as whether an automaton is trim[III].

[III] An automaton is trim if all states are accessible from an initial state and can reach some finial state.

### 6.5.1 Arbitrary predicate on states

The PL logic have only three predicates for states (equality, initial and accepting). However, the model-checking algorithms can be modified to carry over to arbitrary predicates on states as follows. In Lemma 7.2.1, the case of the state predicate $P(\pi_{i_1}^{d_1}, \dots, \pi_{i_k}^{d_k})$ will be handled by the automaton of Figure 6.10. Note that its size is polynomial in $n$. Hence, as long as the evaluation of the predicates is computable, we obtain decidability. In order to keep the tight complexities, our non-deterministic algorithms start by guessing all starting and ending states which satisfy state predicates. So, the number of calls to the NLogSpace evaluation function is linear in the size of the formula.

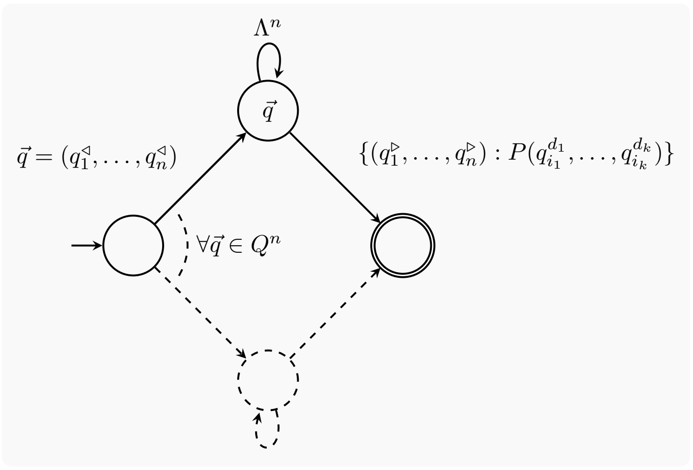

$\Lambda^n$
$\vec{q}$
$\vec{q} = (q_1^{\lhd}, \dots, q_n^{\lhd})$
$\{(q_1^{\rhd}, \dots, q_n^{\rhd}) : P(q_{i_1}^{d_1}, \dots, q_{i_k}^{d_k})\}$
$\forall \vec{q} \in Q^n$

Figure 6.10: Extension of Lemma 7.2.1 in the case of an arbitrary state predicate $P(\pi_{i_1}^{d_1}, \dots, \pi_{i_k}^{d_k})$

We denote by $\mathsf{PL}_{\text{nfa}}[\mathcal{Q}]$ (resp. $\mathsf{PL}_{\text{trans}}[\mathcal{Q}]$, $\mathsf{PL}_{\text{sum}}[\mathcal{Q}]$, $\mathsf{PL}_{\text{sum}}^{\neq}[\mathcal{Q}]$) the class of $\mathsf{PL}_{\text{nfa}}$ formulas (resp. $\mathsf{PL}_{\text{trans}}$, $\mathsf{PL}_{\text{sum}}$, $\mathsf{PL}_{\text{sum}}^{\neq}$) extended with predicate on states over $\mathcal{Q}$.

**Theorem 6.5.1**

Let $\mathcal{Q}$ be a set of state predicates whose evaluation can be done in NLogSpace. We have the following results:

- The model-checking problem of finite automata against formulas in $\mathsf{PL}_{\text{nfa}}[\mathcal{Q}]$ is PSpace-C and NLogSpace-C when the formula is fixed.
- The model-checking problem of transducers against formulas in $\mathsf{PL}_{\text{trans}}[\mathcal{Q}]$ is PSpace-C and NLogSpace-C when the formula is fixed.
- The model-checking problem of sum-automata against formulas in $\mathsf{PL}_{\text{sum}}[\mathcal{Q}]$ is PSpace-C, NP-C when the formula is fixed and NLogSpace-C if additionally the values of the automaton belongs to $\{0, 1\}$.
- The model-checking problem of sum-automata against formulas in $\mathsf{PL}_{\text{sum}}^{\neq}[\mathcal{Q}]$ is PSpace-C NLogSpace when the formula is fixed.

**Example**

We can use coloration of states in order to define a disjoint union of automata. Assume we want to check, given two sum-automata $S_1, S_2$, whether there exists an input word $u$ such that $S_1(u) \cap S_2(u) \neq \varnothing$. This property holds iff the disjoint union $S_1 \uplus S_2$ (where states of $S_i$ are colored by $c_i$) satisfy the formula:

$$\exists \pi_1 : p_1 \xrightarrow{u|v} q_1, \exists \pi_2 : p_2 \xrightarrow{u|v} q_2, \bigwedge_{i=1}^{2} \mathsf{init}(p_i) \wedge \mathsf{final}(q_i) \wedge c_i(p_i)$$

### 6.5.2 Universal quantification over states

The logic PL is purely existential. However simple properties such as whether all states of an automaton are reachable from an initial state are not directly expressible in PL. We define therefore the extension denoted $\forall^* Q\mathsf{PL}[\mathcal{O}]$ to be all formulas of the form

$\forall q_1 \dots \forall q_m\ \varphi$ where $\varphi \in \mathsf{PL}[\mathcal{O}]$. In $\forall^* Q\mathsf{PL}[\mathcal{O}]$, the previous property is expressible by

$$\forall q \exists \pi \colon q_0 \xrightarrow{u|v} q\ \mathsf{init}(q_0)$$

The complexity of model-checking does not change (for all the particular instances of the logic we have considered: finite automata, transducers and sum-automata) if we allow for universal state quantification. Indeed, if the formula is not fixed, it suffices to iterate over all $m$-tuples of states and call a PSpace algorithm over the automaton where these states have been marked (thanks to some coloring, as seen previously), this remains PSpace.

If the formula is fixed, then there is only a polynomial number of $m$-tuples of states and each of them are representable in logarithmic space. The procedure iterate over all tuples of states and run the corresponding model-checking presented in the previous section. This iteration can be done in logarithmic space, since each tuple representation and computing the successor of a tuple takes a logarithmic space. So, we easily get respectively $\text{LogSpace}^{\text{NLogSpace}}$ and $\text{LogSpace}^{\text{NP}}$ depending on the selected model-checking sub-procedure. However our algorithm uses the answer of the oracle only to choose between halting or continuing the iteration which yields respectively NLogSpace and NP memberships.

We get then the following Theorem where hardnesses directly come from the fragments without universal state quantifications.

**Theorem 6.5.2**

- The model-checking problem of finite automata against formulas in $\forall^* Q\mathsf{PL}_{\text{nfa}}$ is PSpace-C and NLogSpace-C when the formula is fixed.
- The model-checking problem of transducers against formulas in $\forall^* Q\mathsf{PL}_{\text{trans}}$ is PSpace-C and NLogSpace-C when the formula is fixed.
- The model-checking problem of sum-automata against formulas in $\forall^* Q\mathsf{PL}_{\text{sum}}$ is PSpace-C, NP-C when the formula is fixed and NLogSpace-C if additionally the weights of the automaton belongs to $\{0, 1\}$.
- The model-checking problem of sum-automata against formulas in $\forall^* Q\mathsf{PL}^{\neq}_{\text{sum}}$ is PSpace-C and NLogSpace-C when the formula is fixed.

# Chapter 7

# Model-checking of structural properties

In this chapter, we provide algorithms for answering the model-checking problem for automata with outputs against a pattern formula expressed in $\mathsf{PL}_{\text{nfa}}$, $\mathsf{PL}_{\text{trans}}$, $\mathsf{PL}_{\text{sum}}$ and $\mathsf{PL}^{\neq}_{\text{sum}}$ and, we investigate their complexity. Our approach relies on the existence of a model which takes as input a tuple of paths of the considered automaton with outputs. Loosely speaking, we show how the satisfiability of atomic predicates can be decided using decidability results of the model on tuples and then, we provide a decision procedure based on closure properties of this model. Our model-checking algorithms rely on regular automata for $\mathsf{PL}_{\text{nfa}}$ and rely on Parikh automata, defined in Section 2.2, for $\mathsf{PL}_{\text{trans}}$, $\mathsf{PL}_{\text{sum}}$ and $\mathsf{PL}^{\neq}_{\text{sum}}$.

## 7.1 An intermediate logic

In the pattern logic, we have four types of variables (paths, states, inputs and outputs). While this is appealing for succinctness and readability reasons, it is less easy to deal with for model-checking. Therefore, we introduce an intermediate logic, called *path pattern logic*, in which there is a single type of variables: the path variables. We then show its equivalence with the pattern logic, in the sense that satisfiability by any automaton with output is preserved.

**Definition**

Let $X$ a countable set of path variables and $\mathcal{O}$ a set of output predicates. A *path formula* $\Phi$ over $\mathcal{O}$ is a term generated by the following grammar:

| | | |
|---|---|---|
| Formula | $\Phi ::=$ | $\Phi \wedge \Phi \mid \Phi \vee \Phi \mid \neg\Phi \mid$ |
| Input predicates | | $\pi_1 \sqsubseteq_I \pi_2 \mid \pi \in_I L \mid \pi_1 \leq_I^{\text{len}} \pi_2 \mid$ |
| State predicates | | $\mathsf{init}(\pi^d) \mid \mathsf{final}(\pi^d) \mid \pi_1^{d_1} =_Q \pi_2^{d_2} \mid$ |
| Path predicates | | $\pi_1 =_P \pi_2$ |
| Output predicates | | $p(t_1, \dots, t_{\alpha(p)})$ |

where $\pi, \pi_1, \pi_2 \in X$, $d, d_1, d_2 \in \{\lhd, \rhd\}$ $t_1, \dots, t_n \in \text{Terms}(X, \oplus, \mathbb{0})$, $p \in \mathcal{O}$ of arity $\alpha(p)$ and $L$ ranges over regular languages over $\Sigma$. We denote by $\mathsf{PL}_{\text{paths}}[\mathcal{O}]$ the set of path formulas over $\mathcal{O}$, and by $\mathsf{PL}^{+}_{\text{paths}}[\mathcal{O}]$ its fragment where output predicates does not occur under an odd number of negations.

We introduce usual syntactic sugar, for the input equality $\pi =_I \pi' \overset{\text{def}}{=} \pi \sqsubseteq_I \pi' \wedge \pi' \sqsubseteq_I \pi$, for the input size comparisons $\pi =_I^{\text{len}} \pi' \overset{\text{def}}{=} \pi \leq_I^{\text{len}} \pi' \wedge \pi' \leq_I^{\text{len}} \pi$ and $\pi <_I^{\text{len}} \pi' \overset{\text{def}}{=} \neg(\pi' \leq_I^{\text{len}} \pi)$.

Informally, path formulas only speak about the paths of an automaton with output, the input constraints speak about properties of their inputs, the state constraints about properties of their starting and ending states, and $\mathcal{O}$ about properties of their outputs. For a given automaton with outputs $A = (Q, Q_I, Q_F, \Delta, \lambda)$ over the monoid $(D, \oplus, \mathbb{0})$, we refer by Paths$(A)$ the words belonging to $Q(\Sigma D Q)^*$ that correspond to a path of $A$. The semantics of such a formula is then, on an automaton with output $A$, valuations of path variables into Paths$(A)$.

Formally, we first fix some monoid $\mathcal{M} = (D_{\mathcal{M}}, \oplus_{\mathcal{M}}, \mathbb{0}_{\mathcal{M}})$ together with an interpretation $p^{\mathcal{M}}$ of each output predicate $p \in \mathcal{O}$ of arity $\alpha(p)$, such that $p^{\mathcal{M}} \in D_{\mathcal{M}}$ if $p$ is a constant i.e. $\alpha(p) = 0$ and $p^{\mathcal{M}} \subseteq D_{\mathcal{M}}^{\alpha(p)}$ otherwise. Let $A$ be an automaton with output over $\mathcal{M}$. Given a valuation $\nu\colon X \to \text{Paths}(A)$, the interpretation $t^{\nu,\mathcal{M}} \in D_{\mathcal{M}}$ of a term $t \in \text{Terms}(X, \oplus, \mathbb{0})$ is inductively defined by $(t_1 \oplus t_2)^{\nu,\mathcal{M}} = t_1^{\nu,\mathcal{M}} \oplus t_2^{\nu,\mathcal{M}}$, $\mathbb{0}^{\nu,\mathcal{M}} = \mathbb{0}_{\mathcal{M}}$ and $\pi^{\nu,\mathcal{M}} = \text{out}(\nu(\pi)) \in D_{\mathcal{M}}$. Then a formula $\Phi \in \mathsf{PL}_{\text{paths}}[\mathcal{O}]$ is interpreted in $A = (Q, Q_I, Q_F, \Delta, \lambda)$ as a set of valuations denoted $[\![\Phi]\!]_A \subseteq \text{Paths}(A)^X$ defined as follows.

$$
\begin{aligned}
&[\![\mathsf{init}(\pi^d)]\!]_A = \{\nu : \nu(\pi)^d \in Q_I\}\\
&[\![\pi_1^{d_1} =_Q \pi_2^{d_2}]\!]_A = \{\nu : \nu(\pi_1)^{d_1} = \nu(\pi_2)^{d_2}\}\\
&[\![\mathsf{final}(\pi^d)]\!]_A = \{\nu : \nu(\pi)^d \in Q_F\}\\
&[\![\neg\Phi]\!]_A = \text{Paths}(A)^X \setminus [\![\Phi]\!]_A\\
&[\![\Phi_1 \wedge \Phi_2]\!]_A = [\![\Phi_1]\!]_A \cap [\![\Phi_2]\!]_A\\
&[\![\Phi_1 \vee \Phi_2]\!]_A = [\![\Phi_1]\!]_A \cup [\![\Phi_2]\!]_A\\
&[\![\pi_1 \sqsubseteq_I \pi_2]\!]_A = \{\nu : \text{in}(\nu(\pi_1)) \sqsubseteq \text{in}(\nu(\pi_2))\}\\
&[\![\pi_1 =_P \pi_2]\!]_A = \{\nu : \nu(\pi_1) = \nu(\pi_2)\}\\
&[\![\pi_1 \leq_I^{len} \pi_2]\!]_A = \{\nu : |\text{in}(\nu(\pi_1))| \leq |\text{in}(\nu(\pi_2))|\}\\
&[\![\pi \in_I L]\!]_A = \{\nu : \text{in}(\nu(\pi)) \in L\}\\
&[\![p(t_1, \ldots, t_{\alpha(p)})]\!]_A = \{\nu : (t_1^{\nu,\mathcal{M}}, \ldots, t_{\alpha(p)}^{\nu,\mathcal{M}}) \in p^{\mathcal{M}}\}
\end{aligned}
$$

A path formula $\Phi$ is said to be satisfiable over $A$, denoted $A \models \Phi$ hereafter, if $[\![\Phi]\!]_A \neq \varnothing$. The path logic and the pattern logic are equivalent in the following precise sense.

**Lemma 7.1.1**

For all pattern formulas $\Phi \in \mathsf{PL}[\mathcal{O}]$, (resp. $\Phi \in \mathsf{PL}^+[\mathcal{O}]$), one can construct in linear time a path formula $\Psi \in \mathsf{PL}_{\text{paths}}[\mathcal{O} \cup \{=_O\}]$ (resp. $\Psi \in \mathsf{PL}^+_{\text{paths}}[\mathcal{O} \cup \{=_O\}]$) such that the following hold:
1. $\text{Var}_P(\Phi) = \text{Var}(\Psi)$
2. for all monoid $\mathcal{M} = (D, \oplus_{\mathcal{M}}, \mathbb{0}_{\mathcal{M}})$, all interpretation $p^{\mathcal{M}} \in D^{\alpha(p)}$ for each $p \in \mathcal{O}$, all automata with output $A$ over $\mathcal{M}$, we have $A \models \Phi$ iff $A \models \Psi$
3. there is a surjective mapping from the valuations (in $A$) satisfying $\Psi$ to the valuations satisfying $\Phi$.

In addition, if $\Phi$ does not contain twice the same output variable in its prefix, then $\Psi \in \mathsf{PL}_{\text{paths}}[\mathcal{O}]$.

**Proof** The proof is straightforward but technical. We include it here for the sake of completeness. Assume that $\Phi$ is of the form:

$$\exists \pi_1 \colon p_1 \xrightarrow{u_1|v_1} q_1,\ \ldots,\ \exists \pi_n \colon p_n \xrightarrow{u_n|v_n} q_n,\ \mathcal{C}$$

In a first step, we rename the $j$-th occurrences, $j > 1$, of any state, input and output variables so that eventually, all these variables are pairwise different in the prefix. This can be easily done in linear-time modulo adding some equality constraints to $\mathcal{C}$. For instance, if we have $p_i = p_j$, $i < j$, we use a fresh variable name $p$ and replace $p_j$ by $p$ in the prefix $\exists \pi_1 \ldots \exists \pi_n$, and add the constraint $p = p_i$ to $\mathcal{C}$. We do the same for the other types of variables. Note that it is not necessary to do it for the path variables $\pi_i$, since by definition of pattern formulas, they are assumed to occur only once in the prefix.
We end up with a formula of the form

$$\Phi' \equiv \exists \pi_1 \colon p_1' \xrightarrow{u_1'|v_1'} q_1',\ \ldots,\ \exists \pi_n \colon p_n' \xrightarrow{u_n'|v_n'} q_n',\ \mathcal{C}'$$

where all the variables occur once in the prefix, together with a mapping $f$ from the new variables to the old ones. We also assume that $f$ is the identify on the path variables. In our previous example, we would have $f(p) = p_j$ for instance.
Then, $\Phi'$ and $\Phi$ are equivalent in the following sense: over an automaton with output, for all valuations $\nu$ satisfying $\Phi$, $\nu \circ f$ satisfies $\Phi'$. Conversely, for all valuations $\nu'$ satisfying $\Phi'$, the valuation $\nu'$ restricted to the variables of $\Phi$ satisfies $\Phi$. Hence, it is possible to recover the models of $\Phi$ from the models of $\Phi'$, and conversely.
Now, note that in $\Phi'$, there is a functional dependency between the state, input and output variables, and the path variables. This dependency is formalized through three func-

tions $f_Q, f_I, f_O$ from respectively, $\{p'_1, \dots, p'_n, q'_1, \dots, q'_n\}$ to $\{\pi_1, \dots, \pi_n\}$, $\{u'_1, \dots, u'_n\}$ to $\{\pi_1, \dots, \pi_n\}$, and from $\{v'_1, \dots, v'_n\}$ to $\{\pi_1, \dots, \pi_n\}$.
Then, $\Psi$ is obtained by applying the following transformations on $\mathcal{C}$:

- Any atom $u \sqsubseteq u'$ is replaced by $f_I(u) \sqsubseteq_I f_I(u')$. Any atom $u \in L$ by $f_I(u) \in_I L$. Any atom $|u| \leq |u'|$ by $f_I(u) \leq_I^{\text{len}} f_I(u')$.
- Any atom $p(t_1, \dots, t_k)$ is replaced by $p(t'_1, \dots, t'_k)$ where $t'_i$ is obtained by substituting any variable $v$ in $t_i$ by $f_O(v)$.
- Any atom $\mathsf{init}(q)$ is replaced by $\mathsf{init}(f_Q(q)^{\lhd})$ if there exists a binding $\exists\pi\colon q \xrightarrow{u \mid v} q'$ in $\Phi'$ otherwise $\mathsf{init}(q)$ is replaced by $\mathsf{init}(f_Q(q)^{\rhd})$ and there exists a binding $\exists\pi\colon q' \xrightarrow{u \mid v} q$ in $\Phi'$. Any atom $\mathsf{final}(q)$ is replaced by $\mathsf{final}(f_Q(q)^{\lhd})$ if there exists a binding $\exists\pi\colon q \xrightarrow{u \mid v} q'$ in $\Phi'$ otherwise $\mathsf{final}(q)$ is replaced by $\mathsf{final}(f_Q(q)^{\rhd})$ and there exists a binding $\exists\pi\colon q' \xrightarrow{u \mid v} q$ in $\Phi'$.
- Finally, any atom $\pi = \pi'$ is replaced by $\pi =_P \pi'$.

Now, $\Phi$ and $\Psi$ are equivalent in the following sense. Any valuation $\nu$ satisfying $\Phi$ satisfies $\Psi$ as well (if restricted to its path variables). Conversely for all valuations $\nu'$ satisfying $\Psi$, we define the following valuation $\nu$:

$$\begin{array}{rcll} \nu(\pi) & = & \nu'(\pi) & \text{for all path variables } \pi \\ \nu(u) & = & \mathrm{in}(\nu'(f_I(u))) & \text{for all input variables } u \\ \nu(v) & = & \mathrm{out}(\nu'(f_O(v))) & \text{for all output variables } v \\ \nu(q) & = & (\nu'(f_Q(q)))^{\lhd} & \text{for all state variables } q \text{ such that there} \\ & & & \text{exists a binding } \exists\pi\colon q \xrightarrow{u \mid v} q' \text{ in } \Phi \\ \nu(q) & = & (\nu'(f_Q(q)))^{\rhd} & \text{otherwise} \end{array}$$

Then, $\nu$ satisfies $\Phi$. Hence, it is possible to recover all the models of $\Phi$ from the models of $\nu$. ♦

The converse of the previous lemma, although not needed for the purpose of this paper, holds as well. For all path formula one can build in linear time a pattern formula which is satisfied over some automaton $A$ iff the former is satisfied as well.

**Example**

Consider the following pattern formula:

$$\exists\pi_1 : p \xrightarrow{u|v} q,\ \exists\pi_2 : p \xrightarrow{u'|v} q',\ v \oplus v \neq v \wedge q \neq q'$$

It is first transformed into the pattern formula where implicit equalities are expressed by additional atoms:

$$\exists\pi_1 : p \xrightarrow{u|v} q,\ \exists\pi_2 : p' \xrightarrow{u'|v'} q',\ v \oplus v \neq v \wedge q \neq q' \wedge p = p' \wedge v = v'$$

Which in turn is transformed into the path formula:

$$\exists\pi_1,\ \exists\pi_2,\ v \oplus v \neq v \wedge \pi_1^{\rhd} \neq_Q \pi_2^{\rhd} \wedge \pi_1^{\lhd} =_Q \pi_2^{\lhd} \wedge \pi_1 =_O \pi_2 \wedge \pi_1 \neq_I \pi_2$$

## 7.2 Model-checking of $\mathsf{PL}_{\mathbf{nfa}}$: Proof of Theorem 6.2.1

Intuitively, we reduce the model-checking of an $\mathsf{NFA}$ $A$ to the emptiness problem of another $\mathsf{NFA}$ $M$, which takes as input a tuple of paths of $A$ corresponding to the existentially quantified paths in the considered pattern formula, modulo a suitable encoding of path tuples as words.

We present the following encoding of tuple of words that fit within to the definition of automata with outputs. Let $\Lambda$ be some alphabet and $\diamond$ a fresh symbol, $\pi \in \Lambda^*$ and $m \geq |\pi|$. The *padding* of $\pi$ with respect to $m$ is the word $\pi\diamond^{m-|\pi|}$. Let $\pi_1, \pi_2 \in \Lambda^*$ and $m = \max(|\pi_1|, |\pi_2|)$. For $j \in \{1, 2\}$, let $\pi'_j$ the padding of $\pi_j$ with respect to $m$. Note that $|\pi'_1| = |\pi'_2| = m$. The *convolution* $\pi_1 \otimes \pi_2$ is the word of length $m$ defined for all $1 \leq i \leq m$ by $(\pi_1 \otimes \pi_2)[i] = (\pi'_1[i], \pi'_2[i])$. E.g. $q_0a_1d_1q_1 \otimes p_0 = (q_0, p_0)(a_1, \diamond)(d_1, \diamond)(q_1, \diamond)$. The convolution can be naturally extended to $n$-tuple of words as follows: $\bigotimes_{i=1}^{n} \pi_i = \pi_1 \otimes (\pi_2 \otimes \ldots \otimes \pi_n)$.

We are now able to define, from an automaton with outputs $A = (Q, Q_I, Q_F, \Delta, \lambda)$, the NFA $\mathrm{Paths}^n(A)$ which accepts an $n$-tuple of paths of $A$ as follows. Let $D$ be the output values occurring on the transitions of $A$ and $\Lambda = \Sigma \cup Q \cup D \cup \{\diamond\}$. The NFA $\mathrm{Paths}^n(A)$ consists in an $n$-time product of $A$, augmented with intermediate states and transitions to read the states of $Q$ and the values in $D$ (seen as letters). Note that, this NFA has an alphabet exponential in $n$.

The following lemma states that all path, state and input predicates are regular sets of path tuples. It holds for any automata with outputs so we state it in this more general context, i.e. not just for finite automata.

**Lemma 7.2.1**

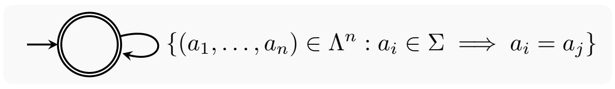
Consider $X = \{\pi_1, \dots, \pi_n\}$ be a finite set of path variables, arbitrarily ordered. Let $A$ be an automaton with outputs and $\Phi$ be an atomic path formula over $X$ built with a unique predicate of $\{\sqsubseteq_I, \in_I L, \leq_I^{len}, \mathsf{init}(.^d), \mathsf{final}(.^d), .^{d_1} =_Q .^{d_2}, =_P\}$ where $d, d_1, d_2 \in \{\lhd, \rhd\}$ and $L$ ranges over regular languages represented by DFA. One can construct an NFA $M$ with a number of states polynomial in $|M| + |\Phi|$ and such that:

$$\mathrm{Paths}^n(A) \cap L(M_\Phi) = \{(\nu(\pi_1), \dots, \nu(\pi_n)) : \nu \colon X \to \mathrm{Paths}(A) \wedge \nu \in [\![\Phi]\!]_A\}$$
$$\mathrm{Paths}^n(A) \cap L(M_{\neg\Phi}) = \{(\nu(\pi_1), \dots, \nu(\pi_n)) : \nu \colon X \to \mathrm{Paths}(A) \wedge \nu \notin [\![\Phi]\!]_A\}$$

**Proof** Let $A = (Q, Q_I, Q_F, \Delta, \lambda)$ be an automaton with output. We denote by $D$ the finite set of output values occurring on $A$. In this proof we provide the construction of an NFA $M_\Phi$ over the alphabet $\Lambda = \Sigma \cup Q \cup D \cup \{\diamond\}$ considering the different cases for $\Phi$:

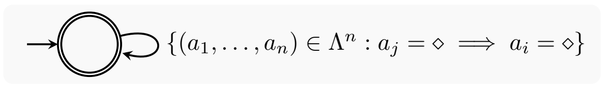
- $\pi_i \sqsubseteq_I \pi_j$. In order to check the equality of the input of the paths $\pi_i$ and $\pi_j$, we can ignore states and outputs, so we only consider input letter (belonging to $\Sigma$). Here $M_\Phi$ is defined as a single-state automaton (with $\mathtt{card}(\Lambda)^n$ transitions).

$(M_{\pi_i \sqsubseteq_I \pi_j})$


$\{(a_1, \dots, a_n) \in \Lambda^n : a_i \in \Sigma \implies a_i = a_j\}$

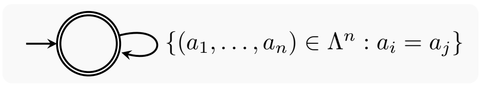
- $\pi_i \in_I L$. We construct an automaton accepting the set of words $u \in \Lambda^n$ such that $f(u) \in L$, where $f$ is the morphism defined by $f(a_1, \dots, a_n) = \varepsilon$ if $a_i \notin \Sigma$, and $f(a_1, \dots, a_n) = a_i$ otherwise, for all $(a_1, \dots, a_n) \in \Lambda^n$. In other words, $M_\Phi$ recognizes the language $f^{-1}(L)$. Since regular languages are closed under inverse morphism, we get the existence of $M_\Phi$. We can construct $M_\Phi$ from any DFA $B$ recognizing $L$ by replacing any transition $(q, \sigma, q')$ of $B$ by the transitions $(q, (a_1, \dots, a_n), q')$ for all $a_1, \dots, a_n \in \Lambda$ such that $a_i = \sigma$, and by adding intermediate states and transitions to $B$ reading letters whose $i$th component is not in $\Sigma$. This is doable by an automaton with a polynomial number of state in $|B|$.
- $\pi_i \leq_I^{\mathrm{len}} \pi_j$. By the definition of convolution, it suffices to check that whenever the $j$th component of the read letter is $\diamond$, then so is its $i$th component. This is done by the single-state automaton

$(M_{\pi_i \leq_I^{\mathrm{len}} \pi_j})$


$\{(a_1, \dots, a_n) \in \Lambda^n : a_j = \diamond \implies a_i = \diamond\}$

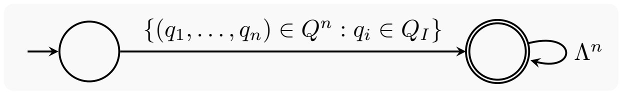
- Case $\pi_i =_P \pi_j$. It suffices to check that the $i$th and $j$th components are the same:

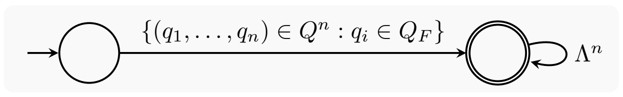
$(M_{\pi_i = \pi_j})$



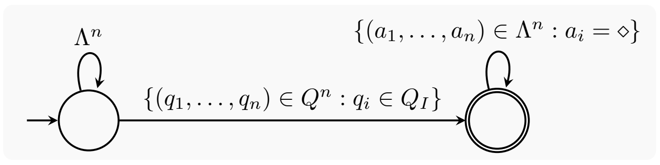
- $\mathsf{init}(\pi_i^\lhd)$ and $\mathsf{final}(\pi_i^\lhd)$. The NFA $M_\Phi$ are respectively:

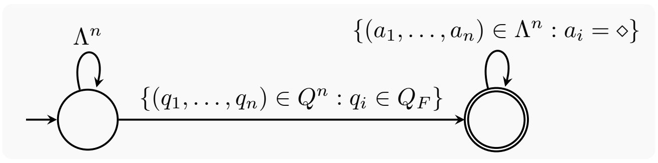
$(M_{\mathsf{init}(\pi_i^\lhd)})$



$(M_{\mathsf{final}(\pi_i^{\triangleleft})})$


- $\mathsf{init}(\pi_i^{\triangleright})$ and $\mathsf{final}(\pi_i^{\triangleright})$. The NFA $M_\Phi$ are respectively:

$(M_{\mathsf{init}(\pi_i^{\triangleright})})$

$\Lambda^n$
$\{(a_1,\ldots,a_n) \in \Lambda^n : a_i = \diamond\}$
$\{(q_1,\ldots,q_n) \in Q^n : q_i \in Q_I\}$

$(M_{\mathsf{final}(\pi_i^{\triangleright})})$

$\Lambda^n$
$\{(a_1,\ldots,a_n) \in \Lambda^n : a_i = \diamond\}$
$\{(q_1,\ldots,q_n) \in Q^n : q_i \in Q_F\}$

- $\pi_i^{d_1} =_Q \pi_j^{d_2}$ with $d_1, d_2 \in \{\triangleleft, \triangleright\}$. The constructions are similar as before. For instance, take the constraint $\pi_i^{\triangleleft} =_Q \pi_j^{\triangleright}$. Then, the automaton $M_\Phi$ needs to remember (in its control state) the first state of $A$ read on the $i$th component and checks later on that it is equal to the last state read on the $j$th component. This is doable by an automaton with a polynomial number of state in $|A|$.

For negations, we construct $M_{\neg\Phi}$ from the automata presented before as follows: Except for the predicate $\in_I L$, all cases have a constant number of states and therefore can be complemented in polynomial time. So, to treat the predicate $\neg(\pi_i \in_I L)$, we first rewrite it into $\pi_i \in_I \overline{L}$, which causes an exponential blow-up in $\Phi$ since $L$ is represented by a DFA. ♦

**Lemma 7.2.2 – easiness of Theorem 6.2.1**

The model-checking problem of NFA against $\mathsf{PL}_{\mathrm{nfa}}$ formulas is in PSPACE. It is in NLOGSPACE when the formula is fixed.

**Proof** Let $\Psi$ be a $\mathsf{PL}_{\mathrm{nfa}}$ formula and $A$ be an NFA. This proof presents an algorithm which decides $A \models \Psi$ in NPSPACE, becoming NLOGSPACE when the formula is fixed. To do so, we reduce the model-checking problem to the emptiness of a product of NFA. First, we consider some transformations of the pattern formula. Note that, those transformation can be done at constant cost in the case where the formula $\Psi$ is fixed. From $\Psi$, we can construct in polynomial time an equivalent path formula thanks to Lemma 7.1.1. The obtained path formula is then put in negation normal form[IV]. Finally all disjunction are non-deterministically resolved by picking which side the model will verify. Eventually, we end up with a path formula $\Phi$ of the form $\bigwedge_{i=1}^k \ell_i$ where each $\ell_i$ are literals[V]. Let $X = \{\pi_1, \ldots, \pi_n\}$ be the path variables of $\Phi$. For all literals $\ell_i$, one can construct in polynomial time an NFA from $A, X$ by Lemma 7.2.1 and then define $M$ an NFA such that:

$$\begin{aligned} L(M) &= \mathrm{Paths}^n(A) \cap \textstyle\bigcap_{i=1}^k L(M_{\ell_i}) \\ &= \textstyle\bigcap_{i=1}^k \{(\nu(\pi_1), \ldots, \nu(\pi_n)) : \nu \colon X \to \mathrm{Paths}(A) \land \nu \in [\![\ell_i]\!]_A\} \\ &= \{(\nu(\pi_1), \ldots, \nu(\pi_n)) : \nu \colon X \to \mathrm{Paths}(A) \land \textstyle\bigwedge_{i=1}^k \nu \in [\![\ell_i]\!]_A\} \\ &= \{(\nu(\pi_1), \ldots, \nu(\pi_n)) : \nu \colon X \to \mathrm{Paths}(A) \land \nu \in [\![\Phi]\!]_A\} \end{aligned}$$

Due to the equivalence between $\Phi$ and $\Psi$ we have $L(M) \neq \varnothing$ iff $A \models \Psi$. Remark that the number of states of $M$ is exponential in the size of the input (in particular in $|\Phi|$) and then cannot be explicitly computed in polynomial space. When the formula is fixed, $|M|$ becomes polynomial in the size of the input (in particular in $|A|$). The classical technique, consisting in searching a small non-emptiness witness non-deterministically on-the-fly and to perform transitions on-demand, allows us to decide the emptiness of $L(M)$ in NPSPACE and, NLOGSPACE when the formula is fixed. ♦

**Lemma 7.2.3 – hardness of Theorem 6.2.1**

The model-checking problem of NFA against formulas in $\mathsf{PL}_{\mathrm{nfa}}$ is hard for PSPACE. It is hard for NLOGSPACE when the formula is fixed.

**Proof** Let $A_1, \dots, A_k$ be $k$ DFA. The proof goes by reduction from the problem of deciding $\bigcap_{i=1}^{k} L(A_i) \neq \varnothing$, which is known to be PSPACE-C thanks to [Koz77] and NLOGSPACE-C if $k$ is fixed. So, we construct $A$ an NFA as the disjoint union of each $A_i$, i.e. $A = A_1 \uplus \cdots \uplus A_n$. Then we define $\Phi \in \mathsf{PL}_{\mathrm{nfa}}$ as follows.

$$\exists \pi_1 \colon q_1 \xrightarrow{u} q_1', \ \dots, \ \exists \pi_k \colon q_k \xrightarrow{u} q_k' \ \left( \bigwedge_{1 \le i < j \le k} q_i \neq q_j \right) \wedge \left( \bigwedge_{1 \le i \le k} \mathsf{init}(q_i) \wedge \mathsf{final}(q_i') \right)$$

The left-hand side part of $\Phi$ asks for $k$ accepting runs on the same input and the right-hand side part of $\Phi$ constraints these paths to start in distinct states. By the disjointness of the construction an accepting run in $A$ is an accepting run of some $A_i$. In addition, there are exactly $k$ initial states, one of each $A_i$ since the automata are deterministic. Thus $A \models \Phi$ if and only if $\bigcap_{i=1}^{n} L(A_i) \neq \varnothing$. ♦

## 7.3 Model-checking of $\mathsf{PL}_{\mathrm{trans}}$: Proof of Theorem 6.3.1

We have seen that input, state and path predicates can be encoded as NFA in Lemma 7.2.1 modulo encoding of path tuples as words, through convolutions. It is however not possible for output predicates of the logic $\mathsf{PL}_{\mathrm{trans}}$. For instance, the atom $|v| = |v'|$ for $v, v'$ two outputs word variables, has not necessarily a regular model. Indeed, consider a transducer $T$ with one state $q$ and two self loops $q \xrightarrow{a \mid \varepsilon} q$ and $q \xrightarrow{b \mid b} q$. The language $\{u \otimes u' : |[\![T]\!](u)| = |[\![T]\!](u')|\} = \{u \otimes u' : |u|_b = |u'|_b\}$ is not regular. In this section, we rely on Parikh automata defined in Section 2.2.

### 7.3.1 Proof of Theorem 6.3.1

The following lemma shows how to encode output predicates as a products of NPA.

**Lemma 7.3.1**

Consider $X = \{\pi_1, \dots, \pi_n\}$ be a finite set of path variables, arbitrarily ordered. Let $T$ be a transducer and $\Phi$ be an atomic path formula over $X$ of the form $t_1 \not\sqsubseteq t_2$ or $t \in N$ or $t \notin N$ or $|t_1| \le |t_2|$ or $|t_1| < |t_2|$ where $N$ ranges over regular languages represented by DFA and $t, t_1, t_2 \in \mathrm{Terms}(X, \cdot, \varepsilon)$. One can construct a NPA $M_\Phi$ with a number of states polynomial in $|T|$ and exponential in $|\Phi|$. Furthermore, its set of weight vectors and its acceptance constraint have a constant size. Finally, the NPA $M_\Phi$ satisfies

$$\mathrm{Paths}^n(T) \cap L(M_\Phi) = \{(\nu(\pi_1), \dots, \nu(\pi_n)) : \nu \colon X \to \mathrm{Paths}(T) \wedge \nu \in [\![\Phi]\!]_T\}$$

**Proof** Let $T = (Q, Q_I, Q_F, \Delta, \lambda)$ be an automaton with outputs in the free monoid. We denote its finite set of output values by $D \subset \Gamma^*$ and we denote by $\mu$ the length of the longest output word appearing on transitions of $T$ i.e. $\mu = \max\{|\lambda(\delta)| : \delta \in \Delta\}$. In this proof we provide the construction of $M_\Phi$ over the alphabet $\Lambda = \Sigma \cup Q \cup D \cup \{\perp\}$ considering the different predicates case by case:

- $|t_1| \preceq |t_2|$ where $\preceq \in \{\le, <\}$. We fix $t_i = \pi'_{i,1} \dots \pi'_{i,m_i}$ with $i \in \{1, 2\}$ and $\pi'_{i,j} \in X$. The predicate $|t_1| \preceq |t_2|$ is true if and only if $\sum_{j=1}^{m_1} |\mathrm{out}(\pi'_{1,j})| \preceq \sum_{j=1}^{m_2} |\mathrm{out}(\pi'_{2,j})|$. We describe the construction of the NPA $M_\Phi$ which takes $\pi_1 \otimes \ldots \otimes \pi_n$ as input word. Intuitively, $M_\Phi$ uses the two counters $c_1, c_2$ to compute for all $i \in \{1, 2\}$, the value $c_i = \sum_{j=1}^{n} |\mathrm{out}(\pi_j)| \times k_{i,j}$ where $k_{i,j} \le m_i$ is the number of occurrences of $\pi_j$ in $t_i$. For acceptance our NPA tests whether $c_1 \preceq c_2$ using an atomic Presburger formula. When $M_\Phi$ reads a symbol which is not an $n$-tuple of outputs, the counter values

[IV]The negation normal form of a formula is obtained by applying the two rewriting rules $\neg(\Phi_1 \wedge \Phi_2) \rightrightarrows (\neg\Phi_1) \vee (\neg\Phi_2)$ and $\neg(\Phi_1 \vee \Phi_2) \rightrightarrows (\neg\Phi_1) \wedge (\neg\Phi_2)$ until reaching a fixed point

[V]A literal is either an atomic formula or the negation of an atomic formula

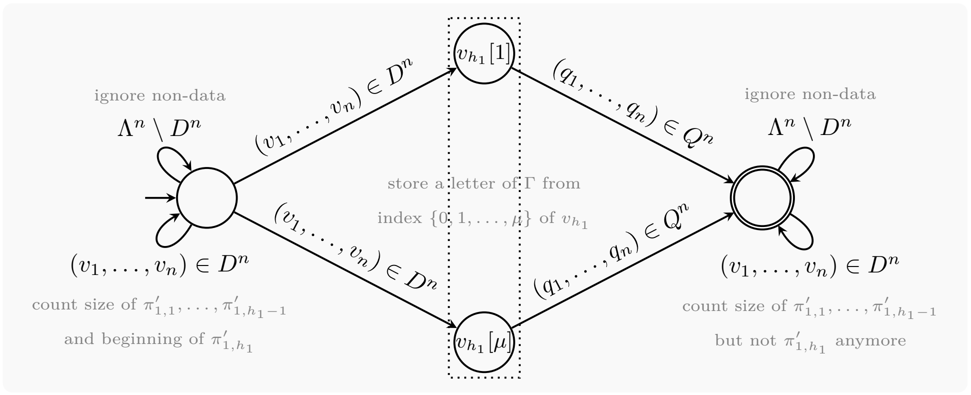

ignore non-data
$\Lambda^n \setminus D^n$
$(v_1,\ldots,v_n) \in D^n$
count size of $\pi'_{1,1},\ldots,\pi'_{1,h_1-1}$
and beginning of $\pi'_{1,h_1}$
$(v_1,\ldots,v_n) \in D^n$
$(v_1,\ldots,v_n) \in D^n$
$v_{h_1}[1]$
store a letter of $\Gamma$ from
index $\{0,1,\ldots,\mu\}$ of $v_{h_1}$
$v_{h_1}[\mu]$
$(q_1,\ldots,q_n) \in Q^n$
$(q_1,\ldots,q_n) \in Q^n$
ignore non-data
$\Lambda^n \setminus D^n$
$(v_1,\ldots,v_n) \in D^n$
count size of $\pi'_{1,1},\ldots,\pi'_{1,h_1-1}$
but not $\pi'_{1,h_1}$ anymore

Figure 7.1: Gadget handling on counter for the construction of the NPA $M_\Phi$

do not change. When it reads $(v_1,\ldots,v_n) \in D^n$, it uses exactly $n \times \max\{m_1,m_2\}$ transitions to update the two counters $c_1, c_2$ in order to keep $c_i = \sum_{j=1}^{n} |v_j| \times k_{i,j}$. Note that, such amount of transitions for updating counters permits to treat each path of the input tuple separately and thus can be realized by using weight vectors belonging to $\{-\mu,\ldots,-1,0,1,\ldots,\mu\}^2$. Finally, the NPA $M_\Phi$ can be constructed with a number of states and number of distinct weight vectors polynomial in both $|T|$ and $|\Phi|$.

- $t \Subset N$ where $\Subset \in \{\in,\notin\}$ and the language $N$ is denoted by the DFA $B = (Q_B, I_B, F_B, \Delta_B)$ over the alphabet $\Gamma$. We fix $t = \pi'_1 \ldots \pi'_m$ with $\pi'_j \in X$. The predicate $t \in N$ holds iff there exists an $(m+1)$-tuple of states $(q_1,\ldots,q_{m+1}) \in Q_B$ such that for all $1 \le j \le m$, there exists a path from $q_j$ to $q_{j+1}$ in $B$ over the word $\mathrm{out}(\pi'_j)$ and where $q_1$ is initial and $q_{m+1}$ is final. We describe the construction of the NPA $M_\Phi$ which takes $\pi_1 \otimes \ldots \otimes \pi_n$ as input word. More precisely $M_\Phi = \biguplus_{\vec{q} \in Q_B^{m-1}} M_{\vec{q}}$ where $M_{\vec{q}}$ is described as follows. Let $\vec{q} = (q_2,\ldots,q_m)$, the NFA $M_{\vec{q}}$ is defined with $Q_B^m$ as set of states, $(q_1,\ldots,q_m) \in Q_B^m$ where $q_1 \in I_B$ as initial state, $(q_2,\ldots,q_{m+1}) \in q_B^m$ where $q_{m+1} \in F_B$ as final state. When $M_{\vec{q}}$ read a symbol which is not an $n$-tuple of input letters, the current state does not change. When it reads $(a_1,\ldots,a_n) \in \Sigma^n$, the transition $(p_1,\ldots,p_m) \to (p'_1,\ldots,p'_m)$ can be triggered if $(p_j, a_i, p'_j)$ for all $1 \le j \le m$, $1 \le i \le n$ and $\pi'_j = \pi_i$. Note that $t \notin N$ can be denoted by $t \in \overline{N}$ and the size of the disjunct union remains exponential in $|\Phi|$. Finally, the NPA (here an NFA) $M_\Phi$ can be constructed with a number of states exponential in $|\Phi|$ and constant in $|T|$.
- $t_1 \not\sqsubseteq t_2$. We fix $t_i = \pi'_{i,1} \ldots \pi'_{i,m_i}$ with $i \in \{1,2\}$ and $\pi'_{i,j} \in X$. Remark, if $|t_1| \not\leq |t_2|$ then $t_1 \not\sqsubseteq t_2$ holds trivially. Thus, we assume w.l.o.g. that $|t_1| \le |t_2|$ (it has been already shown how to construct a NPA which checks this predicate as well as its negation). Under this hypothesis, it remains to construct a NPA which checks for existence of a mismatching position $h$ in $t_1$ and $t_2$, i.e. such that the $h$th letter of the two outputs differ. We describe the construction of $M_\Phi$ which reads $\pi_1 \otimes \ldots \otimes \pi_n$ as input word. The position $h$ may not occur at the same position in the input read by $M_\Phi$, and that is why we need to use counters to identify a position $h_1$ in the output defined by $t_1$, and a position $h_2$ in the output defined by $t_2$, remember the corresponding output labels (using states), and later check that the memorized output letters differ, and that $h_1 = h_2$ (using an atomic Presburger formula). The way the positions $h_1, h_2$ are chosen by $M_\Phi$ works as follows: it guesses an index $j_1 \in \{1,\ldots,m_1\}$ and an index $j_2 \in \{1,\ldots,m_2\}$, intended to verify the existence of a mismatch position in the output of $\pi'_{1,j_1}$ and $\pi'_{2,j_2}$. The counters will respectively count the length of the output produced on $\pi'_{1,1} \ldots \pi'_{1,j_1-1}$ and $\pi'_{2,1} \ldots \pi'_{2,j_2-1}$ (when processing the whole tuple of paths), and also the length of the output produced in respectively $\pi'_{1,j_1}$ and $\pi'_{2,j_2}$ up to some point non-deterministically chosen. Figure 7.1 intuitively shows how one of the two counters is handled. Overall it yields a NPA $M_\Phi$ which checks $t_1 \not\sqsubseteq t_2$ with a number of states and number of distinct weights polynomial in both $|T|$ and $|\Phi|$. ♦

**Lemma 7.3.2 – easiness of Theorem 6.3.1**

The model checking of transducers against $\mathsf{PL}_{\text{trans}}$ formulas is in PSpace. It is in NLogSpace when the formula is fixed.

**Proof** Let $\Psi$ be a $\mathsf{PL}_{\text{trans}}$ formula and $A$ be a transducer. The beginning is the same as it was for NFA. After a series of transformations of $\Psi$ (negation normal form, transformation into path formula, disjunction removing), we end up with the $\mathsf{PL}^{+}_{\text{paths}}[\{\not\sqsubseteq, \in N, \notin N, \leq, <\}]$ path formula $\Phi$ of the form $\bigwedge_{i=1}^{k} \ell_i \wedge \bigwedge_{j=1}^{k'} p_j$ where each $\ell_i$ are either state, input and path literals and all $p_j$ are output atoms[VI]. Now, for all $\ell_i$ we construct an NFA (which is a particular case of NPA) applying Lemma 7.2.1 and for all $p_j$ we construct a NPA applying Lemma 7.3.1. Then we define $M$ as the NPA such that:

$$L(M) = \text{Paths}^n(A) \cap \bigcap_{i=1}^{k} L(M_{\ell_i}) \cap \bigcap_{j=1}^{k'} L(M_{p_j})$$

The acceptance constraint of $M$ consists in a conjunction of atomic Presburger formulas (in particular this conjunction has a polynomial size in $|\Phi|$). Again, we have $L(M) \neq \varnothing$ iff $A \models \Psi$ and the emptiness of $M$ can be decided in PSpace by applying Lemma 2.2.3. When the pattern formula is fixed, the acceptance constraint and the set of weight vectors of $M$ becomes fixed as well. In that case we apply Theorem 2.2.1. to get the NLogSpace membership. ♦

## 7.4 Model-checking of $\mathsf{PL}_{\text{sum}}$ and $\mathsf{PL}^{\neq}_{\text{sum}}$: Proof of Theorems 6.4.1 and 6.4.2

The proof of the results below for $\mathsf{PL}_{\text{sum}}$ follows arguments that are similar to those developed for transducers in the proof of Theorem 6.3.1 (and then also for NFA in the proof of Theorem 6.2.1). However, the NLogSpace-C result for $\mathsf{PL}^{\neq}_{\text{sum}}$ requires to rely on the weak Parikh automata defines in Section 2.2.

### 7.4.1 Proof of Theorems 6.4.1 and 6.4.2

**Lemma 7.4.1**

Consider $X = \{\pi_1, \ldots, \pi_n\}$ be a finite set of path variables, arbitrarily ordered. Let $A$ be a sum-automaton and $\Phi$ be an atomic path formula over $X$ of the form $t_1 \neq t_2$ or $t_1 \leq t_2$ or $t_1 < t_2$ where $t, t_1, t_2 \in \text{Terms}(X, +, 0)$. One can construct a NPA $M_\Phi$ with a number of states and a number of distinct weight vectors polynomial in both $|A|$ and $|\Phi|$, an atomic acceptance constraint and such that:

$$\text{Paths}^n(A) \cap L(M_\Phi) = \{(\nu(\pi_1), \ldots, \nu(\pi_n)) : \nu \colon X \to \text{Paths}(A) \wedge \nu \in [\![\Phi]\!]_A\}$$

In addition, the NPA $M_\Phi$ is weak when $\Phi$ is $t_1 \neq t_2$.

**Proof** The construction is the same as the case of length comparison for transducers. We fix $t_i = \pi'_{i,1} \ldots \pi'_{i,m_i}$ with $i \in \{1,2\}$. By definition $t_1 \preceq t_2$ with $\preceq \in \{\neq, <, \leq\}$ holds if and only if $\sum_{j=1}^{m_1} \text{out}(\pi'_{1,j}) \preceq \sum_{j=1}^{m_2} \text{out}(\pi'_{2,j})$. Intuitively $M_\Phi$ computes for all $i \in \{1,2\}$ the value $c_i = \text{out}(\pi_j) \times k_{i,j}$ where $k_{i,j} \leq m_i$ is the number of occurrences of $\pi_j$ in $t_i$. For updating counters, the NPA treats each path of the input tuple separately and then can uses weight vectors belonging to $(\{0\} \cup D)^2$. For acceptance the NPA $M_\Phi$ tests whether $\sum_{j=1}^{n} c_{1,j} \preceq \sum_{j=1}^{n} c_{2,j}$ using an atomic Presburger formula. ♦

**Lemma 7.4.2 – easiness of Theorem 6.4.1**

The model checking of sum-automata against $\mathsf{PL}_{\text{sum}}$ formulas is in PSpace. It is in NP when the formula is fixed, and NLogSpace if in addition weights of the automaton belong to $\{0,1\}$.

[VI]The path formula $\Phi$ cannot have output literals since $\Psi$ belongs to the fragment where output predicates does not occur under an odd number of negations.

**Proof** Let $\psi$ be a $\mathsf{PL}_{\text{sum}}$ formula and $A$ be a sum-automaton. The proof is similar to the model-checking for transducers against $\mathsf{PL}_{\text{trans}}$. Again, we consider some transformations of $\psi$ which provides the path formula $\Phi$ of the form $\bigwedge_{i=1}^{k} \ell_i \wedge \bigwedge_{j=1}^{k'} \ell'_j$ where each $\ell_i$ is either state, input or path literals and all $\ell'_j$ are output literals. For all $\ell_i$ we construct an $\mathsf{NFA}$ (which is a particular case of $\mathsf{NPA}$) applying Lemma 7.2.1 and for all $\ell'_j$ we construct a $\mathsf{NPA}$ applying Lemma 7.4.1. Then we define $M$ as the $\mathsf{NPA}$ such that:

$$L(M) = \text{Paths}^n(A) \cap \bigcap_{i=1}^{k} L(M_{\ell_i}) \cap \bigcap_{j=1}^{k'} L(M_{\ell'})$$

The acceptance constraint of $M$ consists in a conjunction of atomic Presburger formulas (in particular this conjunction has a polynomial size in $|\Phi|$). The PSPACE and NLOGSPACE results come by applying Lemma 2.2.3 as for the proof of model-checking for transducers against $\mathsf{PL}_{\text{trans}}$. To get the NP upper bound, we effectively construct in polynomial time $M$. In fact, when values $n, k, k'$ are fixed then $M$ have a polynomial number of states. As shown in [FL15] the non-emptiness problem for $\mathsf{NPA}$ is in NP. ◆

**Lemma 7.4.3 – easiness of Theorem 6.4.2**

The model checking of sum-automata against $\mathsf{PL}_{\text{sum}}^{\neq}$ formulas is in PSPACE. It is NLOGSPACE when the formula is fixed (even if the values of the automaton are encoded in binary).

**Proof** The proof is the same as for $\mathsf{PL}_{\text{sum}}$ model-checking, except each call to Lemma 2.2.3 are replaced by Theorem 2.2.6. ◆

**Lemma 7.4.4 – hardness of Theorem 6.4.1**

The model checking of sum-automata against fixed $\mathsf{PL}_{\text{sum}}$ formulas is hard for NP.

**Proof** The proof goes by reduction from the *Set 2-Partition* problem [Pap94], which asks, given a multi-set $S = x_1, \ldots, x_k$ of natural numbers, whether there exists $I$ such that $\sum_{i \in I} x_i = \sum_{i \in S \setminus I} x_i$. Such set $I$ exists if the sum-automaton of Figure 7.2 satisfies the following $\mathsf{PL}_{\text{sum}}$ formula:

$$\exists \pi \colon q_I \xrightarrow{v} q_F,\ \exists \pi_0 \colon q_F \xrightarrow{v_0} q_F \quad \mathsf{init}(q_I) \wedge \mathsf{final}(q_F) \wedge v = v_0$$

The reduction is linear because the automata construction is linear and the formula is constant. ◆

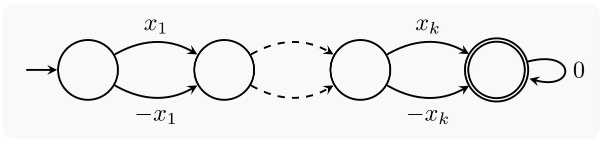


Figure 7.2: Sum-automaton used for encoding Set Partition

## 7.5 Summary and future works

We introduced several logics to reason about structural properties of regular automata, transducers and sum-automata.

### Summary

The complexity to decide whether an automaton with outputs in a monoid fulfills the specification given by a pattern formula, either constant or part of the input, have been published in [FMR18, FMR19]. We focus on how the three instances allow to easily recover known complexity results for deciding automata subclasses, as well as new results, such as for instance the NLOGSPACE upper bound of the $k$-valuedness problem for sum-automata (with a fixed $k$ and weights in binary). The following table summarizes the complexity results of the model-checking problem (for sum-automata, the weights are assumed to be encoded in binary):

| | $\mathsf{PL}_{\text{nfa}}$ | $\mathsf{PL}_{\text{trans}}$ | $\mathsf{PL}_{\text{sum}}$ | $\mathsf{PL}^{\neq}_{\text{sum}}$ |
|---|---|---|---|---|
| fixed | PSpace-C | PSpace-C | PSpace-C | PSpace-C |
| unfixed | NLogSpace-C | NLogSpace-C | NP-C | NLogSpace-C |

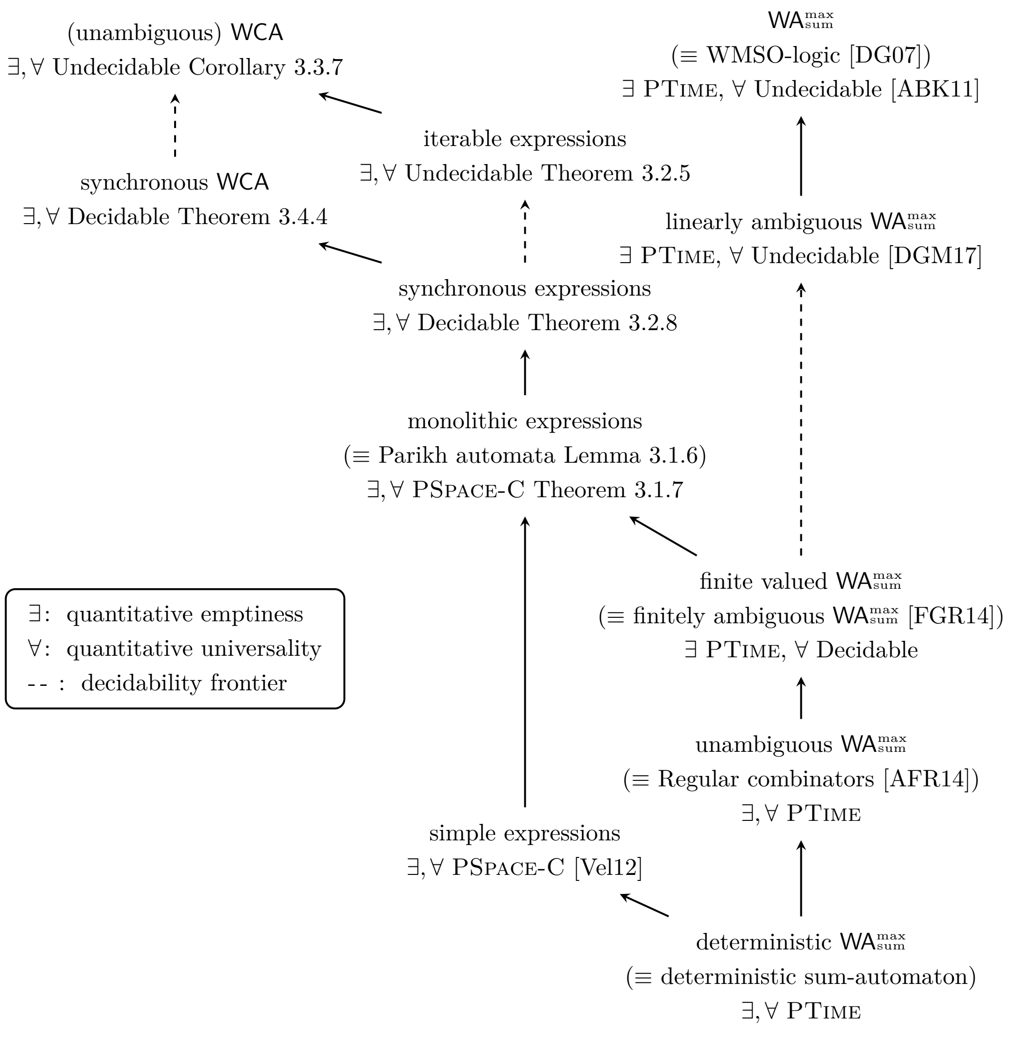
**Future works**

In [FMR18], we give sufficient conditions, on the output monoid, under which the problem of model-checking an automaton with outputs against a formula in the generic logic is decidable. Briefly, these conditions require the existence of a machine model accepting tuples of runs which satisfy the atomic predicates of the logic, is closed under union and intersection, and has decidable emptiness problem.

Other output monoids appear frequently in the literature and may share common techniques, for instance, discounted sum can be encoded as some operation of a monoid as proved in [FGR15]. There are many possible extensions that ought to be considered, such as tree automata with Presburger constraints [Won10], two-way automata with Presburger constraints [FGM19] and two-way visibly pushdown transducers and Parikh automata [DFRT16, DFT19]. For instance, there is a rich literature on tree transducers and two-way transducers where deciding structural patterns is an important problem, that would deserve to have a dedicated logic and model-checking algorithm. Based on examples of structural patterns from the literature, our objective is to design a logical syntax in which those patterns could be naturally expressed.

# Conclusion

This thesis lies in the general realm of formal methods and more precisely in quantitative extensions of verification and synthesis methods for reactive systems. The model-checking is, nowadays, a standardized approach to assure the design of reactive systems that are dependable, safe, and efficient. Its classical (Boolean) setting is well known and elegantly supported by regular automata-theoretic methods. However, the correct/incorrect abstraction level is coarse and some applications require to capture situations beyond regularity. Quantitative extensions of model-checking have been developed, although the landscape of this framework is not fully understood, and challenging questions remain open.



(unambiguous) WCA
∃, ∀ Undecidable Corollary 3.3.7
WA$^{\max}_{\text{sum}}$
(≡ WMSO-logic [DG07])
∃ PTime, ∀ Undecidable [ABK11]
iterable expressions
∃, ∀ Undecidable Theorem 3.2.5
synchronous WCA
∃, ∀ Decidable Theorem 3.4.4
linearly ambiguous WA$^{\max}_{\text{sum}}$
∃ PTime, ∀ Undecidable [DGM17]
synchronous expressions
∃, ∀ Decidable Theorem 3.2.8
monolithic expressions
(≡ Parikh automata Lemma 3.1.6)
∃, ∀ PSpace-C Theorem 3.1.7
finite valued WA$^{\max}_{\text{sum}}$
(≡ finitely ambiguous WA$^{\max}_{\text{sum}}$ [FGR14])
∃ PTime, ∀ Decidable
∃: quantitative emptiness
∀: quantitative universality
-- : decidability frontier
unambiguous WA$^{\max}_{\text{sum}}$
(≡ Regular combinators [AFR14])
∃, ∀ PTime
simple expressions
∃, ∀ PSpace-C [Vel12]
deterministic WA$^{\max}_{\text{sum}}$
(≡ deterministic sum-automaton)
∃, ∀ PTime

Figure 7.3: Comparison between the weighted expression formalisms and the weighted chop automata (WCA) introduced in Chapter 3, as well as classes of weighted automata (WA$^{\max}_{\text{sum}}$). The arrows denote strict inclusion between the classes of quantitative languages defined by the formalisms and dashed arrows highlights when the frontier of decidability is cross.

The contribution follows three main axes. The first part focuses on computational models that manipulate integer weights with arithmetical operations featuring a tread-off between the expressibility of quantitative properties and the feasibility of time/memory requirements. In particular, we introduced the programming languages monolithic and iterable expressions that can combine functions realized by weighted automata using the Presburger arithmetic. The second part, investigate error/noise formalisms and robustness synthesis to relax the standard model-checking approach with a parametric disturbance. Our approach provides a measure of confidence about the model-checking answer adapted for erroneous systems (as sensors) and specifications (from statistical observations for instance). The third part, introduce new specification languages that allow us to define and efficiently decide structural properties for popular automata classes (regular automata, transducers, weighted automata). In fact, to retain the feasibility and tractability of model-checking algorithms, the verification community introduces many pattern criteria expressible into this logical formalisms.

Figure 7.3 presents the main formalisms that have been studied from literature and from Part I. It orders them depending on their expressiveness (models on bottom are less expressive than the ones on top). It also recalls some structural pattern as finite valuedness or linear ambiguity which can be express in the pattern logic introduced in Part III. In Part II, we model noise with weighed transducers. In the case of Sum-transducer, we rely on automata weighed over $\mathbb{N}$ to solve the robust kernel synthesis Lemma 5.1.2. Note that, the monolithic expressions are closed under Presburger definable functions which allows us to express various word metrics and so, is a good candidate as model of noise.

For sack for fact, we explain here how the finite valuedness restriction on Sum-transducer over $\mathbb{Z}$ raise computability of the robust kernel. Formally, we consider the specification language $L$ given by a DFA $A$, we let $T$ be a $k$-valued Sum-transducer which model input noise and we take $\nu \in \mathbb{Z}$ as errors threshold. We show now how to represent the set of words $\texttt{Rob}_T(\nu, L)$, i.e. the robust kernel. First, we construct the finite valued sum-automaton $N$ from $T$ by restricting its output domain to belongs to $L$ and then we remove the output words.[VII] Thanks to [SdS10], we can decomposed $N$ into $k$ unambiguous sum-automata $N_1, \dots, N_k$ such that $[\![N]\!] = \bigcup_{i=1}^{k} [\![N_i]\!]$. Hence, we defined the monolithic expression $E = \min(N_1, \dots, N_k)$ that denotes the function from a word of $\texttt{dom}(T)$ to the cost of the minimal rewriting that belongs to $L$.[VIII] In order to represent $\texttt{Rob}_T(\nu, L)$, we need to bounded the images of $E$, which is doable since monolithic expressions are close by Presburger definable function, let $E|_\nu$ be restriction of $E$. The robust kernel is thus the domain of $E|_\nu$ which is regular by Proposition 3.2.4.

[VII]The construction is in a similar vain of the weighted transition system $G_{T,A}$ of Section 4.2 but input are preserved.

[VIII]Here, the combinator could be replaced by any existential Presburger functional formula $\varphi\colon \mathbb{Z}^k \to \mathbb{Z}$.

# Bibliography

[ABK11] Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In Tevfik Bultan and Pao-Ann Hsiung, editors, *Proceedings of the 9th International Symposium on Automated Technology for Verification and Analysis, ATVA'11*, volume 6996 of *Lecture Notes in Computer Science series*, pages 482–491. Springer, 2011.

Cited 4 times on pages 3, 59, 61, and 93.

[ACP$^+$19] Shaull Almagor, Nathann Cohen, Guillermo A. Pérez, Mahsa Shirmohammadi, and James Worrell. Coverability in 1-vass with disequality tests. *Computing Research Repository*, abs/1902.06576, 2019.

Cited 1 time on page 23.

[AFR14] Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In Thomas A. Henzinger and Dale Miller, editors, *Proceedings of the joint meeting of the 23th Annual Conference of the European Association for Computer Science Logic, CSL'14 and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'14*, pages 9:1–9:10. ACM, 2014.

Cited 5 times on pages 4, 29, 35, 45, and 93.

[AH15] Takumi Akazaki and Ichiro Hasuo. Time robustness in MTL and expressivity in hybrid system falsification. In Daniel Kroening and Corina S. Pasareanu, editors, *Part II of proceedings of the 27th International Conference on Computer Aided Verification, CAV'15*, volume 9207 of *Lecture Notes in Computer Science series*, pages 356–374. Springer, 2015.

Cited 1 time on page 58.

[AKTY13] Rajeev Alur, Sampath Kannan, Kevin Tian, and Yifei Yuan. On the complexity of shortest path problems on discounted cost graphs. In Adrian-Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *Proceedings of the 7th International Conference on Language and Automata Theory and Applications, LATA'13*, volume 7810 of *Lecture Notes in Computer Science series*, pages 44–55. Springer, 2013.

Cited 1 time on page 53.

[AM03] Cyril Allauzen and Mehryar Mohri. Efficient algorithms for testing the twins property. *Journal of Automata, Languages and Combinatorics*, 8(2):117–144, 2003.

Cited 1 time on page 8.

[AMR11] Cyril Allauzen, Mehryar Mohri, and Ashish Rastogi. General algorithms for testing the ambiguity of finite automata and the double-tape ambiguity of finite-state transducers. *International Journal of Foundations of Computer Science*, 22(4):883–904, 2011.

Cited 1 time on page 71.

[AR13] Rajeev Alur and Mukund Raghothaman. Decision problems for additive regular functions. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Part II of proceedings of the 40th International Colloquium on Automata, Languages, and Programming, ICALP'13*, volume 7966 of *Lecture Notes in Computer Science series*, pages 37–48. Springer, 2013.

Cited 1 time on page 44.

[BC02] Marie-Pierre Béal and Olivier Carton. Determinization of transducers over finite and infinite words. *Theoretical Computer Science*, 289(1):225–251, 2002.

Cited 1 time on page 8.

[BCB$^+$14] Nihat Baysal, Fraser Cameron, Bruce A. Buckingham, Darrell M. Wilson, H. Peter Chase, David M. Maahs, B. Wayne Bequette, and In Home Closed-Loop Study Group. A novel method to detect pressure-induced sensor attenuations (pisa) in an artificial pancreas. *Journal of diabetes science and technology*, 8(6):1091–1096, 2014.

Cited 1 time on page 58.

[BCPS03] Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science*, 292(1):45–63, 2003.

Cited 5 times on pages 8, 11, 38, 74, and 76.

[Ber77] Jean Berstel. Some recent results on recognizable formal power series. In Marek Karpinski, editor, *Proceedings of the 1st International Symposium on Fundamentals of Computation Theory, FCT'77*, volume 56 of *Lecture Notes in Computer Science*, pages 39–48. Springer, 1977.

Cited 1 time on page 2.

[Ber79] Jean Berstel. *Transductions and context-free languages*, volume 38 of *Teubner Studienbücher : Informatik series*. Teubner, 1979.

Cited 3 times on pages 5, 7, and 49.

[BHO15] Udi Boker, Thomas A. Henzinger, and Jan Otop. The target discounted-sum problem. In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'15*, pages 750–761. IEEE Computer Society, 2015.

Cited 1 time on page 64.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. The MIT Press, 2008.

Cited 1 time on page 2.

[Bon11] Rémi Bonnet. The reachability problem for vector addition system with one zero-test. In Filip Murlak and Piotr Sankowski, editors, *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science MFCS'11*, volume 6907 of *Lecture Notes in Computer Science series*, pages 145–157. Springer, 2011.

Cited 1 time on page 45.

[CDE$^+$10] Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. In Paul Gastin and François Laroussinie, editors, *Proceedings of the 21th International Conference on Concurrency Theory, CONCUR'10*, volume 6269 of *Lecture Notes in Computer Science series*, pages 269–283. Springer, 2010.

Cited 3 times on pages 3, 4, and 25.

[CDH10a] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Expressiveness and closure properties for quantitative languages. *Logical Methods in Computer Science*, 6(3), 2010.

Cited 2 times on pages 62 and 64.

[CDH10b] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Transactions on Computational Logic*, 11(4):23:1–23:38, 2010.

Cited 2 times on pages 2 and 61.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs*, volume 131 of *Lecture Notes in Computer Science series*, pages 52–71. Springer, 1981.

Cited 1 time on page 2.

[CFM12] Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Affine parikh automata. *Revue d'Automatique, d'Informatique et de Recherche OpÃľrationnelle – Theorical Informatics Applications*, 46(4):511–545, 2012.

Cited 1 time on page 23.

[CFM13] Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Unambiguous constrained automata. *International Journal of Foundations of Computer Science*, 24(7):1099–1116, 2013.

Cited 1 time on page 23.

[CGP01] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking.* The MIT Press, 2001.

Cited 2 times on pages 2 and 15.

[CHJS15] Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann, and Rohit Singh. Measuring and synthesizing systems in probabilistic environments. *Journal of the ACM*, 62(1):9:1–9:34, 2015.

Cited 1 time on page 64.

[Cho77a] Christian Choffrut. Applications séquentielles permutables. *Journal of Information Processing and Cybernetics*, 13(7/8):351–357, 1977.

Cited 1 time on page 2.

[Cho77b] Christian Choffrut. Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles. *Theoretical Computer Science*, 5(3):325–337, 1977.

Cited 3 times on pages 8, 74, and 76.

[CHO15] Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested weighted automata. In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'15*, pages 725–737. IEEE Computer Society, 2015.

Cited 1 time on page 44.

[CHVB18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking.* Springer, 2018.

Cited 1 time on page 69.

[Coo72] David C. Cooper. Theorem proving in arithmetic without multiplication. *Machine intelligence*, 7(91–99):300, 1972.

Cited 1 time on page 21.

[CS86] Christian Choffrut and Marcel Paul Schützenberger. Décomposition de fonctions rationnelles. In Burkhard Monien and Guy Vidal-Naquet, editors, *Proceedings of the 3rd Annual Symposium on Theoretical Aspects of Computer Science, STACS'86*, volume 210 of *Lecture Notes in Computer Science series*, pages 213–226. Springer, 1986.

Cited 3 times on pages 8, 74, and 77.

[DFRT16] Luc Dartois, Emmanuel Filiot, Pierre-Alain Reynier, and Jean-Marc Talbot. Two-way visibly pushdown automata and transducers. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'16*, pages 217–226. ACM, 2016.

Cited 1 time on page 92.

[DFT19] Luc Dartois, Emmanuel Filiot, and Jean-Marc Talbot. Two-way parikh automata with a visibly pushdown stack. In Mikolaj Bojanczyk and Alex Simpson, editors, *Proceedings of the 22nd International Conference on Foundations of Software Science and Computation Structures, FOSSACS'19, held as part of the European Joint Conferences on Theory and Practice of Software, ETAPS'19*, volume 11425 of *Lecture Notes in Computer Science series*, pages 189–206. Springer, 2019.

Cited 1 time on page 92.

[DG07] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380(1–2):69–86, 2007.

Cited 1 time on page 93.

[DGM17] Laure Daviaud, Pierre Guillon, and Glenn Merlet. Comparison of max-plus automata and joint spectral radius of tropical matrices. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science, MFCS'17*, volume 83 of *Leibniz International Proceedings in Informatics series*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

Cited 4 times on pages 25, 31, 33, and 93.

[DJRV17] Laure Daviaud, Ismaël Jecker, Pierre-Alain Reynier, and Didier Villevalois. Degree of sequentiality of weighted automata. In Javier Esparza and Andrzej S. Murawski, editors, *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures, FOSSACS'17, held as part of the European Joint Conferences on Theory and Practice of Software, ETAPS'17*, volume 10203 of *Lecture Notes in Computer Science series*, pages 215–230, 2017.

Cited 3 times on pages 8, 75, and 76.

[DKV09] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata.* Springer-Verlag Berlin Heidelberg, 2009.

Cited 6 times on pages 2, 4, 5, 9, 12, and 49.

[DM10] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Proceedings of the 8th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS'10*, volume 6246 of *Lecture Notes in Computer Science series*, pages 92–106. Springer, 2010.

Cited 1 time on page 58.

[DMP17] Jyotirmoy V. Deshmukh, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying conformance using the skorokhod metric. *Formal Methods System Design*, 50(2–3):168–206, 2017.

Cited 1 time on page 58.

[ES69] Samuel Eilenberg and Marcel-Paul Schützenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13(2):173–191, 1969.

Cited 1 time on page 41.

[ES06] Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Operations Research Letters*, 34(5):564–568, 2006.

Cited 1 time on page 16.

[FGH13] Alain Finkel, Stefan Göller, and Christoph Haase. Reachability in register machines with polynomial updates. In Krishnendu Chatterjee and Jirí Sgall, editors, *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science, MFCS'13*, volume 8087 of *Lecture Notes in Computer Science series*, pages 409–420. Springer, 2013.

Cited 1 time on page 23.

[FGM19] Emmanuel Filiot, Shibashis Guha, and Nicolas Mazzocchi. Two-way parikh automata. In Arkadev Chattopadhyay and Paul Gastin, editors, *Proceedings of the 39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'19*, volume 150 of *Leibniz International Proceedings in Informatics series*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

Cited 2 times on pages 23 and 92.

[FGR14] Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Finite-valued weighted automata. In Venkatesh Raman and S. P. Suresh, editors, *Proceedings of the 34th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS'14*, volume 29 of *Leibniz International Proceedings in Informatics series*, pages 133–145. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.

Cited 9 times on pages 4, 8, 25, 27, 33, 64, 65, 79, and 93.

[FGR15] Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Quantitative languages defined by functional automata. *Logical Methods in Computer Science*, 11(3), 2015.

Cited 7 times on pages 8, 23, 34, 64, 65, 79, and 92.

[FL15] Diego Figueira and Leonid Libkin. Path logics for querying graphs: Combining expressiveness and efficiency. In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'15*, pages 329–340. IEEE Computer Society, 2015.

Cited 3 times on pages 19, 20, and 91.

[FMR18] Emmanuel Filiot, Nicolas Mazzocchi, and Jean-François Raskin. A pattern logic for automata with outputs. In Mizuho Hoshi and Shinnosuke Seki, editors, *Proceedings of the 22nd International Conference on Developments in Language Theory, DLT'18*, volume 11088 of *Lecture Notes in Computer Science series*, pages 304–317. Springer, 2018.

Cited 2 times on pages 91 and 92.

[FMR19] Emmanuel Filiot, Nicolas Mazzocchi, and Jean-François Raskin. Decidable weighted expressions with presburger combinators. *Journal of Computer and System Sciences*, 106:1–22, 2019.

Cited 1 time on page 91.

[FP09] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

Cited 1 time on page 58.

[GI83] Eitan M. Gurari and Oscar H. Ibarra. A note on finitely-valued and finitely ambiguous transducers. *Mathematical Systems Theory*, 16(1):61–66, 1983.

Cited 2 times on pages 8 and 77.

[Gri68] Timothy V. Griffiths. The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. *Journal of the ACM*, 15(3):409–413, 1968.

Cited 1 time on page 8.

[GS64] Seymour Ginsburg and Edwin H. Spanier. Bounded algol-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964.

Cited 1 time on page 15.

[GS66] Seymour Ginsburg and Edwin H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

Cited 1 time on page 15.

[GS81] Fritz J. Grunewald and Daniel Segal. How to solve a quadratic equation in integers. *Mathematical Proceedings of the Cambridge Philosophical Society*, 89(1):1âĂŞ–5, 1981.

Cited 1 time on page 23.

[Gur85] Eitan M. Gurari. Decidable problems for powerful programs. *Journal of the ACM*, 32(2):466–483, 1985.

Cited 1 time on page 45.

[HO13] Thomas A. Henzinger and Jan Otop. From model checking to model measuring. In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *Proceedings of the 24th International Conference on Concurrency Theory, CONCUR'13*, volume 8052 of *Lecture Notes in Computer Science series*, pages 273–287. Springer, 2013.

Cited 1 time on page 64.

[HP84] David Harel and Amir Pnueli. On the development of reactive systems. In Krzysztof R. Apt, editor, *Proceedings of the International Conference on Logics and Models of Concurrent Systems*, volume 13 of *NATO Avanced Study Institute series*, pages 477–498. Springer, 1984.

Cited 1 time on page 1.

[HPR18] Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Looking at mean payoff through foggy windows. *Acta Informatica*, 55(8):627–647, 2018.

Cited 1 time on page 65.

[HZ19] Christoph Haase and Georg Zetzsche. Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'19*, pages 1–14. IEEE, 2019.

Cited 1 time on page 45.

[JF18] Ismaël Jecker and Emmanuel Filiot. Multi-sequential word relations. *International Journal of Foundations of Computer Science*, 29(2):271–296, 2018.

Cited 3 times on pages 8, 74, and 77.

[Kar78] Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978.

Cited 1 time on page 53.

[KLMP04] Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004.

Cited 4 times on pages 25, 26, 27, and 33.

[Koz77] Dexter Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science, FOCS'77*, pages 254–266. IEEE Computer Society, 1977.

Cited 3 times on pages 7, 60, and 88.

[KR03] Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Proceedings of the 30th International Colloquium on Automata, Languages and Programming, ICALP'03*, volume 2719 of *Lecture Notes in Computer Science series*, pages 681–696. Springer, 2003.

Cited 4 times on pages 4, 15, 18, and 23.

[Kro94] Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–426, 1994.

Cited 2 times on pages 3 and 25.

[KVW00] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

Cited 2 times on pages 2 and 15.

[Lin10] Anthony Widjaja Lin. *Model checking infinite-state systems: generic and specific approaches.* PhD thesis, University of Edinburg, 2010.

Cited 1 time on page 17.

[MCB$^+$14] David M. Maahs, Peter Calhoun, Bruce A. Buckingham, H. Peter Chase, Irene Hramiak, John Lum, Fraser Cameron, B. Wayne Bequette, Tandy Aye, Terri Paul, Robert Slover, R. Paul Wadwa, Darrell M. Wilson, Craig Kollman, Roy W. Beck, and In Home Closed Loop Study Group. A randomized trial of a home system to reduce nocturnal hypoglycemia in type 1 diabetes. *Diabetes care*, 37(7):1885–1891, 2014.

Cited 1 time on page 57.

[Moh02] Mehryar Mohri. Edit-distance of weighted automata. In Jean-Marc Champarnaud and Denis Maurel, editors, *Proceedings of the 7th International Conference on Implementation and Application of Automata, CIAA'02*, volume 2608 of *Lecture Notes in Computer Science series*, pages 1–23. Springer, 2002.

Cited 1 time on page 49.

[MRT11] Rupak Majumdar, Elaine Render, and Paulo Tabuada. Robust discrete synthesis against unspecified disturbances. In Marco Caccamo, Emilio Frazzoli, and Radu Grosu, editors, *Proceedings of the 14th ACM International Conference on Hybrid Systems on Computation and Control, HSCC'11*, pages 211–220. ACM, 2011.

Cited 1 time on page 64.

[Pap94] Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

Cited 1 time on page 91.

[Par66] Rohit Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

Cited 2 times on pages 15 and 17.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science, FOCS'77*, pages 46–57. IEEE Computer Society, 1977.

Cited 1 time on page 2.

[Pre29] Mojżesz Presburger. Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen. In *Sprawozdanie z I Kongresu metematyków słowiańskich*, pages 92–101 and 395, 1929.

Cited 2 times on pages 15 and 16.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Proceedings of the 5th Colloquium on International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science series*, pages 337–351. Springer, 1982.

Cited 1 time on page 2.

[Ros39] Barkley Rosser. The $n$-th prime is greater than $n \log n$. *Proceedings of the London Mathematical Society*, s2–45(1):21–44, 1939.

Cited 1 time on page 21.

[RS59] Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

Cited 1 time on page 2.

[Saa15] Aleksi Saarela. Systems of word equations, polynomials and linear algebra: A new approach. *European Journal of Combinatorics*, 47:1–14, 2015.

Cited 1 time on page 75.

[Sca84] Bruno Scarpellini. Complexity of subcases of presburger arithmetic. *Transactions of the American Mathematical Society*, 284(1):203–218, 1984.

Cited 2 times on pages 17 and 20.

[Sch61] Marcel Paul Schützenberger. On the definition of a family of automata. *Journal of Information and Control*, 4(2-3):245–270, 1961.

Cited 1 time on page 2.

[SdS08] Jacques Sakarovitch and Rodrigo de Souza. On the decidability of bounded valuedness for transducers. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science, MFCS'08*, volume 5162 of *Lecture Notes in Computer Science series*, pages 588–600. Springer, 2008.

Cited 2 times on pages 8 and 77.

[SdS10] Jacques Sakarovitch and Rodrigo de Souza. Lexicographic decomposition of $k$-valued transducers. *Theory of Computing Systems*, 47(3):758–785, 2010.

Cited 4 times on pages 4, 23, 77, and 94.

[SI85] Richard Edwin Stearns and Harry B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *Society for Industrial and Applied Mathematics Journal on Computing*, 14(3):598–611, 1985.

Cited 1 time on page 7.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, STOC'73*, pages 1–9. ACM, 1973.

Cited 1 time on page 7.

[Vel12] Yaron Velner. The complexity of mean-payoff automaton expression. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Proceedings part II of the 39th International Colloquium on Automata, Languages, and Programming, ICALP'12*, volume 7392 of *Lecture Notes in Computer Science series*, pages 390–402. Springer, 2012.

Cited 3 times on pages 4, 25, and 93.

[VSS05] Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational horn clauses. In Robert Nieuwenhuis, editor, *Proceedings of the 20th International Conference on Automated Deduction, CADE'05*, volume 3632 of *Lecture Notes in Computer Science series*, pages 337–352. Springer, 2005.

Cited 1 time on page 19.

[VW86] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32(2):183–221, 1986.

Cited 1 time on page 69.

[Web90] Andreas Weber. On the valuedness of finite transducers. *Acta Informatica*, 27(8):749–780, 1990.

Cited 1 time on page 77.

[Web93] Andreas Weber. Decomposing finite-valued transducers and deciding their equivalence. *Society for Industrial and Applied Mathematics Journal on Computing*, 22(1):175–202, 1993.

Cited 2 times on pages 8 and 77.

[WK95] Andreas Weber and Reinhard Klemm. Economy of description for single-valued transducers. *Information and Computation*, 118(2):327–340, 1995.

Cited 3 times on pages 8, 74, and 76.

[Won10] Karianto Wong. *Finite automata on unranked trees: extensions by arithmetical and equality constraints.* PhD thesis, RWTH Aachen University, 2010.

Cited 1 time on page 92.

[WS91] Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991.

Cited 3 times on pages 8, 71, and 72.

[YPA06] Jun Yuan, Carl Pixley, and Adnan Aziz. *Constraint-based verification.* Springer, 2006.

Cited 1 time on page 1.

# Abstract

*Reactive systems* are computer systems that maintain continuous interaction with the environment in which they operate. Such systems are nowadays part of our daily life: think about common yet critical applications like engine control units in automotive, aircraft autopilots, medical aided-devices, or micro-controllers in mass production. Clearly, any flaw in such critical systems can have catastrophic consequences. However, they exhibit several characteristics, like resource limitation constraints, real-time responsiveness, concurrency that make them difficult to implement correctly. To ensure the design of reactive systems that are dependable, safe, and efficient, researchers and industrials have advocated the use of so-called *formal methods*, that rely on mathematical models to express precisely and analyze the behaviors of these systems.

*Automata theory* provides a fundamental notion such as languages of program executions for which membership, emptiness, inclusion, and equivalence problems allow us to specify and verify properties of reactive systems. However, these Boolean notions yield the correctness of the system against a given property that sometimes, falls short of being satisfactory answers when the performances are prone to errors. As a consequence, a common engineering approach is not just to verify that a system satisfies a property, but whether it does so within a degree of quality and robustness.

This thesis investigates the expressibility, recognition, and robustness of quantitative properties for program executions.

- Firstly, we provide a survey on languages definable by regular automata with Presburger definable constraints on letter occurrences for which we provide descriptive complexity. Inspired by this model, we introduce an expression formalism that mixes formula and automata to define quantitative languages i.e. function from words to integers. These expressions use Presburger arithmetic to combine values given by regular automata weighted by integers. We show that quantitative versions of the classical decision problems such as emptiness, universality, inclusion, and equivalence are computable. Then we investigate the extension of our expressions with a “Kleene star” style operator. This allows us to iterate an expression over smaller fragments of the input word, and to combine the results by taking their iterated sum. The decision problems quickly turn out to be not computable, but we introduce a new subclass based on a structural restriction for which algorithmic procedures exist.
- Secondly, we consider a notion of robustness that places a distance on words, thus defining neighborhoods of program executions. A language is said to be robust if the membership satisfiability cannot differ for two “close” words, and that leads to robust versions of all the classical decision problems. Our contribution is to study robustness verification problems in the context of weighted transducers with different measures (sum, mean-payoff, and discounted sum). Here, the value associated by the transducer to rewrite a word into another denotes the cost of the noise that this rewriting induce. For each measure, we provide algorithms that determine whether a language is robust up to a given threshold of error and we solve the synthesis of the robust kernel for the sum measure. Furthermore, we provide case studies including modeling human control failures and approximate recognition of type-1 diabetes using robust detection of blood sugar level swings.
- Finally, we observe that algorithms for structural patterns recognition of automata often share similar techniques. So, we introduce a generic logic to express structural properties of automata with outputs in some monoid, in particular, the set of predicates talking about the output values is parametric. Then, we consider three particular automata models (regular automata, transducers, and automata weighted by integers) and instantiate the generic logic for each of them. We give tight complexity results for the three logics with respect to the pattern recognition problem. We study the expressiveness of our logics by expressing classical structural patterns characterizing for instance unambiguity and polynomial ambiguity in the case of regular automata, determinizability, and finite-valuedness in the case of transducers and automata weighted by integers. As a consequence of our complexity results, we directly obtain that these classical properties can be decided in logarithmic space.