New Perspectives on Classical Automata Constructions

PH.D. THESIS

Elena Gutiérrez Viedma

New Perspectives on Classical Automata Constructions

Submitted in partial fulfillment of the requirements for the degree of:
Doctor of Philosophy in Software, Systems and Computing

Author: Elena Gutiérrez Viedma
Double Degree in Mathematics and Computer Science
Universidad Autónoma de Madrid

Advisor: Prof. Pierre Ganty
Associate Research Professor
IMDEA Software Institute

Thesis committee

Emmanuel Filiot	Université Libre de Bruxelles
Lars-Åke Fredlund	Universidad Politécnica de Madrid
Fernando Rosa Velardo	Universidad Complutense de Madrid
César Sánchez Sánchez	IMDEA Software Institute
Hellis Tamm	Tallinn University of Technology

Abstract

The Theory of Automata is one of the most fundamental and longstanding mathematical theories which deals with the study of abstract machines, or automata, and their computational capabilities. In this thesis, we focus on the automata models of finite-state automata and pushdown automata, as well as context-free grammars. These models have been used in numerous applications: from the synthesis and formal verification of hardware and software systems to natural language processing applications or digital-image compression techniques. These applications strongly rely on language-theoretic notions where, still today, many questions are open. The goal of this thesis is to give new theoretic perspective on a collection of open problems, that are at the core of classical automata constructions and well-established algorithms. The underlying mathematical tool to approach these questions are equivalence relations on words as abstractions of languages.

First, we focus on algorithms for obtaining the deterministic automaton with the minimal number of states for some given regular language. This is an essential question that arises in many applications such as text processing, image analysis and program verification. While most minimization techniques rely on either the fusion of equivalent states or an iterative refinement of an initial partition of the set of states, like Hopcroft's or Moore's algorithm, the double-reversal method, proposed by Brzozowski, stands isolated from these methods as it simply combines two classical automata constructions twice in order to obtain the minimal deterministic automaton. In this thesis, we aim to understand the language-theoretic basis of the double-reversal method and its connection with the partition-based techniques, a longstanding question that, still today, attracts interest. As a result, we provide a uniform framework of deterministic automata constructions based on finite-index equivalences on words that allows us to give a new simple proof of the double-reversal method and shed light on the relation between this algorithm and the partition-based techniques.

Second, we address the question of comparing the descriptional complexity of pushdown automata against context-free grammars and finite-state automata for Parikh equivalence, a weaker notion than the standard language equivalence under which the ordering of symbols in the words is not important. The interest on this weaker notion of equivalence is in good part due to the celebrated Parikh's Theorem, which shows that every context-free language is Parikh-equivalent to some effectively computable regular language. Our main contribution is to provide an infinite family of pushdown automata defined over a singleton alphabet that allows us to give lower bounds on the number of grammar variables (resp. states) of the smallest context-free grammar (resp. finite-state automaton) that is language-equivalent. Since Parikh equivalence and language equivalence coincide when the alphabet is a singleton, we achieve our goal by deriving lower bounds for Parikh equivalence as well. As these lower bounds match existing upper bounds, we conclude the optimality of known translation procedures in the unary case and for Parikh equivalence, such as the textbook translation procedure that converts any pushdown automaton into a language-equivalent context-free grammar.

Finally, we address the question of extending Parikh's Theorem to the more general model of weighted automata. It is well-known that Parikh's Theorem does not hold for weighted languages. Thus, we study sufficient conditions under which the so-called Parikh property holds for weighted automata. We show that every nonexpansive weighted context-free grammar over a commutative semiring satisfies the Parikh property. Furthermore, we give a decision procedure for the Parikh property for weighted context-free grammars over the rational numbers that relies on the use of Groebner bases.

Resumen

La Teoría de Autómatas es la teoría matemática encargada del estudio de máquinas abstractas, o autómatas, y sus capacidades computacionales. En esta tesis nos centramos en los modelos de autómatas de estados finitos y de pila, así como en las gramáticas libres de contexto. Estos modelos encuentran aplicación, por ejemplo, en el diseño y verificación formal de software y hardware, en técnicas de proceso de lenguaje natural o en compresión de imágenes digitales. Todas estas aplicaciones están basadas en nociones de Teoría de Lenguajes y Autómatas sobre las que existen numerosas cuestiones sin resolver a día de hoy. El objetivo de esta tesis es dar una nueva perspectiva teórica a un conjunto de cuestiones abiertas que tratan fundamentalmente sobre construcciones clásicas de autómatas y algoritmos conocidos. La herramienta matemática subyacente para resolver estas cuestiones son las relaciones de equivalencia sobre palabras, interpretadas como abstracciones del lenguaje.

Primero, estudiamos algoritmos de minimización de autómatas de estados finitos, esto es, métodos para obtener el autómata determinista con el mínimo número de estados dado un lenguaje regular. Estos algoritmos juegan un papel crucial en aplicaciones de procesado de texto y diálogo, análisis de imágenes y verificación de programas. Mientras que la mayoría de las técnicas de minimización se basan en fusionar estados equivalentes o refinar iterativamente una partición inicial del conjunto de estados del autómata, como los algoritmos de Hopcroft o Moore, el conocido algoritmo de Brzozowski se aleja de estas técnicas, ya que simplemente combina dos conocidas operaciones sobre autómatas para obtener el autómata mínimo. En esta tesis, buscamos entender la base teórica a nivel del lenguaje del algoritmo de Brzozowski y su conexión con los algoritmos que se basan en refinar una partición inicial de estados, una cuestión que a día de hoy sigue despertando interés. Nuestra contribución principal es ofrecer un marco uniforme de construcciones de autómatas deterministas definidas a partir de equivalencias sobre palabras que permite dar una prueba más simple del algoritmo de Brzozowski y clarificar la relación entre este método y las

otras técnicas de minimización.

En segundo lugar, comparamos los autómatas de pila con las gramáticas libres de contexto y los autómatas de estados finitos en cuanto a su complejidad descriptiva cuando todos estos formalismos describen lenguajes Parikh-equivalentes. La equivalencia de Parikh es una noción más débil que la usual equivalencia de lenguajes bajo la cual el orden de los símbolos en las palabras no es importante. Su interés se debe al célebre Teorema de Parikh que establece que todo lenguaje libre de contexto es Parikh-equivalente a un lenguaje regular. Nuestra contribución principal es dar una familia infinita de autómatas de pila definidos sobre un alfabeto con un único símbolo que permite dar cotas inferiores en el número de variables (resp. de estados) de la gramática (resp. autómata) más pequeña con el mismo lenguaje. Como la equivalencia de Parikh coincide con la de lenguajes cuando el alfabeto sólo tiene un símbolo, cumplimos el objetivo planteado obteniendo cotas inferiores para equivalencia de Parikh también. Al comparar estas cotas con cotas superiores conocidas, concluimos que algoritmos ya existentes de conversión entre estos formalismos son óptimos.

Por último, buscamos extender el Teorema de Parikh al modelo de autómatas con pesos. Es bien conocido que este teorema no se cumple para lenguajes con pesos. Por ello, estudiamos condiciones suficientes bajo las cuales la propiedad de Parikh se cumpla. Demostramos que toda gramática libre de contexto con pesos sobre un semianillo conmutativo que sea no-expansiva satisface la propiedad de Parikh. Además, damos un procedimiento de decisión para dicha propiedad sobre el semianillo de los racionales que se basa en el uso de bases de Groebner.

A mis padres

Acknowledgements

Estas líneas son un modesto agradecimiento a todos los que habéis hecho este proyecto posible y divertido.

En primer lugar, quiero agredecerle esta oportunidad a la persona más crucial en esta causa, mi supervisor Pierre Ganty. Thank you for trusting on me, for sharing your time *any* time, and for your always easy-going attitude. I am very grateful for all I have learnt from you on Automata Theory and research methodology since my summer internships until the end of my Ph.D. También quiero agradecer todo el tiempo compartido a mi compañero de despacho, co-autor y gran amigo, Pedro. Tu espíritu trabajador y tenaz me ha estimulado siempre, y sin duda, esta tesis no habría sido la misma sin ti.

I would like to thank to Ichiro Hasuo the valuable and unforgettable opportunity of doing a 6-months internship in Tokyo. I am also grateful to Niccolas Mazzocchi and Emmanuel Filiot for having me in Bruxelles, it was great to meet you and your group; and to Roberto Giacobazzi for a great course in Computability. And thank you both, Emmanuel and Roberto, for taking the time to read a manuscript of this thesis and write a detailed evaluation.

During these 4 years at IMDEA I have had the pleasure to meet excellent people and friends: Álex, Álvaro Feal, Álvaro García, Antonio, Artem, Bogdan, Borja, Chana, Felipe, Ignacio, Isabel, Kyveli, Nataliia, Paloma, Pedro, Platon and Srdjan. You really are a *cool* group. Me gustaría mostrar mi agradecimiento de forma más especial a Ignacio, por su constante predisposición a escuchar y ayudar, y por haberse leído la introducción de esta tesis sin rechistar; y a Chana, for being always a touch of fresh air and for having gathered us together so many times out of the office. I will always have a beautiful memory of your internship in Madrid.

El tiempo que dedicas a no pensar en la tesis es casi tan importante como el que le dedicas, y en ese sentido me siento afortunada de tener unos amigos de infancia extraordinarios que lo han llenado siempre de

momentos memorables. Guti y Dani, Marina y Laura, Layos, Carlos, Sara y Víctor, Marta y la otra Marta, Irene y David, ... Vosotros sí que hacéis de esto un gran viaje.

Gracias a mis amigos de la universidad, Álvaro, Cris, Edu, los Guilles, Parra, Pedro y Víctor. Por interesaros por el estado de este documento, por los buenos momentos que pasamos juntos, y por recordarme siempre lo bueno que está el brócoli. Gracias especialmente a Guridi, por hablarme del instituto y por compartir conmigo la primera estancia.

Esta tesis está dedicado a mis padres. A ellos y a mis hermanos, Adrián y Álvaro, les agradezco, entre tantas cosas, el ejemplo que me han dado y me dan siempre.

Reservo las últimas líneas a mi gran compañero. Miguel, gracias por tu tiempo, tu ingenio, tus mil y una formas de sorprenderme siempre y tu cariño.

Contents

List of Figures

List of Publications

This thesis comprises 3 published papers in peer-reviewed academic conferences.

- Parikh Image of Pushdown Automata
  Pierre Ganty and Elena Gutiérrez.
  In Proceedings of FCT 2017. [36]

- The Parikh Property of Weighted Context-Free Grammars
  Pierre Ganty and Elena Gutiérrez.
  In Proceedings of FSTTCS 2018. [37]

- A Congruence-based Perspective on Automata Minimization Algorithm
  Pierre Ganty, Elena Gutiérrez and Pedro Valero.
  In Proceedings of MFCS 2019. [38]

Other conference articles co-authored during the term of my Ph.D. program, but not included in this dissertation are:

- Genetic Algorithm for the Weight Maximization Problem on Weighted Automata
  Elena Gutiérrez, Takamasa Okudono, Masaki Waga and Ichiro Hasuo.
  In Proceedings of GECCO 2020. [48]

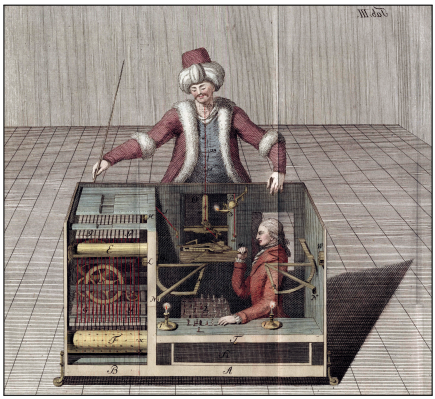
- A Quasiorder-based Perspective on Residual Automata
  Pierre Ganty, Elena Gutiérrez and Pedro Valero.
  In Proceedings of MFCS 2020. [39]

1 Introduction

The Theory of Automata is one of the most fundamental and longstanding mathematical theories, primarily incepted for modeling the capabilities of computer systems. Nowadays, automata pervade numerous tasks from the design and verification of hardware and software systems to text and speech recognition or digital image processing. Still numerous problems on classical automata constructions of great relevance in practice are open. The goal of this dissertation is to give new language-theoretic perspective on a collection of open questions on classical automata constructions and methods, whose study is motivated by the use of these models in practice.

1.1 Background

The human ambition for constructing self-operating machines has materialized recurrently along history, from ancient times, when the first programmable machines appeared [85], to, undeniably, our days. One intriguing example is *The Turk*, also known as *The Automaton Chess Player*, a chess-playing machine created in 1770 that was able to compete with a strong game and defeated a number of challengers, including Napoleon Bonaparte [69]. Any computer scientist, or enthusiastic of the chess game, might be confused about the capabilities of The Turk to reason about such a complex puzzle at that early stage of history. Certainly, the Turk had no such ability: nor the resources neither the knowledge had yet been developed for that sophisticated task. The only explanation, and indeed the true version of the facts, is that this machine was actually a hoax, as it was physically operated by a small, yet talented, player hidden inside.



Source: Wikimedia Commons

Figure 1.1: The *real* Automaton Chess Player.

There exist other examples along history of *real* automata that were designed to automatize mechanical tasks, such as the *Differencial Engine*, devised by the computer pioneer Charles Babbage during the 1820's to tabulate polynomial functions. Even though this is not the concept of automata formalized by the Theory of Automata, where automata are abstract "machines", both models point to a common aspiration: to build objects that can model complex functions by following a predetermined sequence of instructions automatically. What makes the formal notion of automata more appealing is that they allow to reason about both *how* these functions are computed and *what* we can solve by performing sequences of steps automatically.

Automata are mathematical objects described as systems that consist of states and labeled transitions with a simple ultimate purpose: *accept* or *reject words*. Consequently, they define *formal languages*, or just *languages*. Despite of their simplicity, they are very useful to describe what we can do and expect from hardware and software. As a matter of fact, nowadays automata stand as a fundamental building block in computer science with a wide range of applications in the design, analysis and verification of hardware and software systems.

There exist several types of automata that range from the simplest finite-state automaton to the more sophisticated Turing machines. In this dissertation we will focus on *finite-state automata*, *pushdown automata* and

the equivalent grammar model, i.e., *context-free grammars*. We will also be interested in the more general model of *weighted context-free grammars*.

Finite-state automata (NFAs[1]) are transition systems simply described by a finite set of *states* and *transitions* between them, which are labeled over a finite *alphabet* of symbols. These labels can represent program instructions, inputs or outputs, phones or words in a natural language, etc. In addition, a finite-state automaton specifies an *initial* and *final* configuration, which are distinguished states that allow to define how words are accepted and rejected by the automaton. Namely, a *computation* is a sequence of consecutive transitions from an initial to a final configuration that, as a result, reads a finite sequence of symbols, i.e., a *word*. In this case, we say that the word is *accepted* by the automaton. If the computation does not start from an initial state or ends in a final one, the word is *rejected*. Figure 1.2 illustrates the latter explanation with an example.

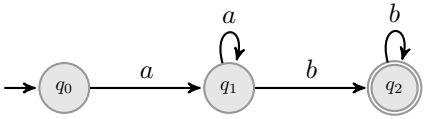


Figure 1.2: A finite-state automaton accepting the regular language $\{a^n b^m \mid n, m \geq 1\}$. The unlabeled incoming arrow indicates the initial state and the doubly circled state corresponds to the final. Note that this automaton is *deterministic*, since there is one single initial state and no two transitions labeled with the same symbol leave the same state.

On the other hand, *pushdown automata* (PDAs) are finite-state automata which are additionally provided with an auxiliary memory that obeys a LIFO policy, also known as *stack*, that can store an arbitrary number of symbols from a finite *stack alphabet*. In consequence, each configuration of a pushdown automaton is given by a state and a stack content. One use of the stack may be the storage and retrieval of return addresses during function calls in programs with procedures. Figure 1.3 shows an example of a pushdown automaton.

In a nutshell, both types of automata accept (possibly infinite) sets of words, or languages, by means of computations, sequences of labeled transitions from an initial to a final configuration. Since pushdown

[1]The *N* stands for *nondeterministic* finite-state automata.

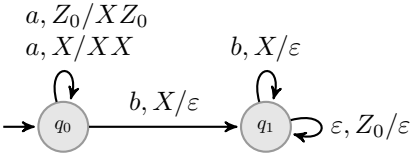


Figure 1.3: A pushdown automaton accepting the context-free language $\{a^n b^n \mid n \geq 1\}$. At each transition and after the comma we indicate the actions on the stack. The symbol before the bar indicates the symbol popped from the top of the stack, and the sequence of symbols after it denotes the symbols pushed into the stack. Note that the leftmost symbol in the pushed sequence corresponds to the topmost symbol in the stack. The symbol ε denotes the empty string, and Z_0, the initial stack symbol. In this case, the final configuration is reached when the stack is empty.

automata rely on a more complex description, they accept a class of language that strictly contains the languages accepted by a finite-state automaton. More pointedly, while NFAs accept the *regular languages*, PDAs recognize the more general class of *context-free languages* (CFLs).

The latter languages are also defined as those that can be *generated* by a *context-free grammar* (CFG). This is a model with a recursive structure that consists of a finite set of *rules* that indicate how to derive words of the language by means of replacements of *variables* by sequences of variables and symbols from the alphabet. In this setting, the counterpart of computations are grammar *derivations* that start from an *initial variable* and end up in a sequence of symbols that constitutes the generated word. While CFGs are equivalent to PDAs in the sense that both represent the context-free languages, *regular* CFGs, a restricted class of CFGs, are equivalent to NFAs as both represent the regular languages.

Finally, in this dissertation we also explore grammar constructions where rules are augmented with a *weight*. In that sense, *weighted context-free grammars* (WCFGs) are a generalization of the classical ones, where every word is not just generated but it also carries a weight. In general, we assume that the weights lie over an algebraic structure called *semiring*, such as the natural or rational numbers. The weight of a derivation is defined as the semiring product of the weights of all rules used along the derivation, and the weight of a word is the semiring sum of the weights of all possible derivations of the word. Thus, these models generate *weighted languages*. It is worth to remark that, as in the unweighted case, WCFGs

are equivalent to weighted PDAs (WPDAs) and regular WCFGs are equivalent to weighted NFAs (WFAs), in the sense that these models represent weighted context-free languages and weighted regular languages, respectively. Figure 1.4 shows an example of a WCFG over the natural numbers.

$$\begin{array}{ll} X \to aXX & \mathbf{1} \\ X \to a & \mathbf{1} \end{array}$$

Figure 1.4: A weighted context-free grammar over the natural numbers. We show the weight of each rule next to it in bold. The language generated by this grammar is $\{a^{2n+1} \mid n \geq 0\}$. The reader can check that, for instance, the weight of the words a, $aaa = a^3$ and $aaaaa = a^5$ is 1, 1 and 2, respectively. Since the weight of each rule is 1, the weight of each word in this grammar can be interpreted as its multiplicity.

In practice, these automata pervade numerous tasks in classical and modern computer science. On one hand, finite-state automata have been widely applied for the design of switching circuits, the implementation of the lexical analyzer that conforms compilers and the construction of regular expression engines. They have also been proved useful for approaching problems in additive number theory [79], for digital image processing and compression [58, 89] and in the development of regular language learning techniques [44, 3] that enable applications such as the abstraction of recurrent neural networks [90]. In the ambit of program verification, finite-state automata are useful tools for performing model checking verification [6, 24] as well as for representing the set of reachable stack configurations of pushdown systems [13, 35].

On the other hand, pushdown automata, and specially, context-free grammars, play an important role in the definition of the structure of programming languages, the design of parsers and the description of document formats such as XML. CFGs producing one single word, i.e., *straight-line grammars*, have been successfully used for data compression, also known as *grammar-based compression* [74, 60]. PDAs also allow to reason about programs with first-order recursion [13, 35, 80], like asynchronous [40] and multithreaded programs [84].

Finally, the number of applications grows when considering the richer weighted model. Weighted context-free grammars, in particular *stochastic* context-free grammars, have been used for modeling RNA secondary

structure [61] and in natural language processing applications [70]. In formal verification, WFAs and WPDAs are used for quantitative verification of programs [22, 29, 83], for interprocedural dataflow analysis [80] and to reason about probabilistic systems [5, 15]. Other modern applications of WFAs are speech processing [71], optical character recognition [62], digital image compression [4, 49] and weighted automata learning [7], which has been used to obtain richer abstractions of recurrent neural networks [75].

1.2 Motivation

All the aforementioned applications are strongly founded on language-theoretic notions around which, still today, many questions are open.

Notice that the description of finite-state automata, pushdown automata and context-free grammars always involves *finite* sets. Therefore, these devices constitute a *finite* representation of the languages they recognize, which enables the implementation of the systems they model with a fixed set of resources. However, a finite representation of a language does not guarantee having an effective procedure to solve certain problems about them. For instance, context-free languages have a finite representation as PDAs, as well as CFGs, but there is no algorithm that returns always the right answer to the question of whether the intersection of two CFLs is empty or not, for *any* given pair of them. In other words, the empty intersection of context-free languages is an *undecidable* problem.

Similarly, WFAs, WPDAs and WCFGs are finite representations of the weighted languages they accept. In the weighted case, the domain of weights is important when determining whether a problem is decidable or not. For instance, given a weighted finite-state automaton and a threshold value in the domain of weights, the *threshold reachability problem*, i.e., the problem of determining whether there exists a word such that its weight is greater than or equal to the threshold, is undecidable with respect to the tropical semiring[2] with domain in the integers [64], but it becomes decidable when the tropical semiring has its domain over the naturals [65].

Indeed, many language-theoretic questions of great practical interest in modern applications are still open and solving them is a challenging goal of active theoretic research. For instance, one central issue in the application

[2]For the tropical semiring, the weight of a computation is the *sum* of the weights of all transitions used along the sequence, and the weight of a word is the *minimum* of the weights of all computations reading the word.

of weighted finite-state automata to text and speech recognition [72] is that the WFAs used during the search stage are highly redundant. This means that the same word can be accepted by several different computations with distinct weights or probabilities. This nondeterminism drastically affects the speed of large vocabulary speech recognition tasks. However, while in the unweighted case, for each NFA there exists a deterministic one that accepts the same language, this convenient property is not true for all WFAs. Partial decidability results regarding this property are known in certain cases but the general question remains open [2].

Notably, these questions also persist in the ambit of classical automata constructions and well-established algorithms that manipulate them. The study of these longstanding notions is still today a line of active research in Automata Theory that aims to better understand their language-theoretic basis and provide practitioners solid ground to explore algorithmic improvements and new applications.

One clear example is the efforts that have been made to give a common theoretic basis to well-known *automata minimization* methods, i.e., algorithms that aim to build the *deterministic finite-state automaton* (DFA) with the least possible number of states that accepts some given regular language. [20, 9, 1, 87, 18, 12, 43]. Getting the minimal DFA is an essential problem that arises in many applications such as text processing, image analysis and program verification. The majority of the minimization methods in the literature rely on building a partition of the set of states of the automaton, either by an iterative refinement, like Hopcroft's or Moore's algorithm [51, 73], or by aggregating equivalent states [52]. On the other hand, the minimization method proposed by Brzozowski [16] simply alternates two well-known automata constructions twice to build the minimal DFA for any input NFA. These two operations are the *determinization* and *reverse* constructions[3], and thus this method is also known as the *double-reversal* method. Despite its exponential worst-case time complexity, the simplicity of the double-reversal method has motivated the study of this algorithm in the past few decades from different theoretic views [25, 18, 87, 12], as well as its connection with the partition-based methods [20, 9, 1, 43], with the goal of providing more efficient versions of it.

Another example is the study of efficient methods that translate

[3]Given an NFA, the *reverse* automaton results from flipping the source and destination state of every transition, and the initial and final states, while the *determinization* operation refers to the classical *subset construction* [52].

pushdown automata into context-free grammars. If one is interested in producing a CFG generating the same language as a given PDA, then Goldstine et al. [45] showed that the standard conversion procedure that appears in textbooks [52] is optimal, in the sense that, no other algorithm produces always a language-equivalent grammar with fewer variables.

However, there exist other notions of equivalence between languages that are of great interest in practice. One example is the so-called *Parikh equivalence*, a weaker notion than the usual equivalence of words: two words are *Parikh-equivalent* if each symbol occurs equally often in both words but not necessarily at the same positions, such as the words *aba* and *baa*. The interest on this notion of equivalence is in good part due to a celebrated result in the Theory of Formal Languages known as *Parikh's Theorem* [76]. This result shows that for every context-free language there is an effectively computable regular language that is Parikh-equivalent. For instance, the CFL $\{a^n b^n \mid n \geq 1\}$ is Parikh-equivalent to the regular language $\{(ab)^n \mid n \geq 1\}$, since for each word in the first language there is a word in the second that is Parikh-equivalent, and viceversa.

Among other applications, Parikh's Theorem has been used in the analysis of multithreaded asynchronous programs with procedures [40, 84]. Under this paradigm, each program thread may send asynchronous messages or tasks to other threads during their execution. Note that the language of all sequences of messages that can be sent by each thread during its executions is a context-free language. However, since the ordering in which these tasks are dispatched and processed from the bag of messages is irrelevant in the idiom of asynchronous programming, given a pushdown automaton representing the language of possible message sequences, a Parikh-equivalent finite-state automaton can be effectively computed. As a result, the control state reachability problem of programs following this paradigm is decidable [84].

In the context of program verification, the notion of Parikh equivalence has also been used for under approximating the reachable state space of multithreaded procedural programs and recursive counter programs [41], and for upper approximating the set of reachable paths of concurrent programs with procedures [14]. In both cases, the corresponding Parikh abstraction of context-free languages representing the reachable program configurations makes emptiness of the intersection decidable.

Additionally, this equivalence has been used to study the complexity of decision problems for Petri nets, commutative grammars, unary grammars and semilinear sets [54, 55, 56, 31], and to establish complexity bounds

on verification problems for counter machines [46] and equational Horn clauses [88].

Going back to the problem of efficient conversion algorithms from PDAs to language-equivalent CFGs, previous works [23, 78] have proved that when the alphabet is *unary*, i.e., is a singleton set, and the input pushdown automaton is deterministic, there exist translation methods that are more efficient than the texbook procedure [52] and return grammars with fewer variables. Whether efficient conversion procedures exist from general PDAs to Parikh-equivalent CFGs or to Parikh-equivalent NFAs are open questions.

In this dissertation, we address a collection of language-theoretic questions relative to our models of interest that are at the core of several well-known algorithms or have great interest for their application in the design and implementation of computer systems, and we give a novel perspective to solve them. The common denominator of this study is the use of *equivalence relations over words* as language abstractions. More precisely, we will consider two types of equivalences: *congruences* with a *finite* number of equivalence classes, and the so-called *Parikh equivalence*. Despite of sharing the same nature, namely, they are both types of *congruences*, these equivalences are fundamentally different. In particular, as opposed to the first type of congruences, Parikh equivalence defines infinitely many equivalence classes. Their concrete properties will make these equivalences on words suitable for the study of different classes of automata.

1.3 Research Questions

Automata minimization methods. Regular langauges are those that can be recognized by a finite-state automaton. On the other hand, one of the early results in the theory of finite-state automata, the so-called *Nerode's theorem* (see for instance [27]), offers an equivalent characterization of regular languages in terms of an equivalence relation over words. More precisely, this equivalence is a *congruence*.

Broadly speaking, a *congruence* is an equivalence relation over words with "good" properties with respect to the concatenation of symbols. Namely, *right* congruences behave well w.r.t. concatenation of symbols to the right, while *left* congruences behave well w.r.t. left concatenation.

The equivalence relation described in Nerode's theorem is a right congruence defined in terms of a language L and states that two words,

u and v, are *equivalent* if they are not *distinguishable* by L, i.e., uw is in L if and only if vw is in L, for every $w \in \Sigma^*$. Concretely, Nerode's theorem states that a language L is regular if and only if the *index*, i.e., the number of equivalence classes, of the latter relation is *finite*. An interesting consequence of this theorem is that the proof provides an automata construction which turns out to be the *minimal deterministic automaton* for the language (for details on this construction see for instance [59]). The intuition is that, given a regular language L, the language read from the initial to each state of the minimal deterministic finite-state automaton for L coincides with the equivalence classes of Nerode's right congruence.

It is not difficult to observe that the construction of the minimal DFA from Nerode's right congruence can be generalized to obtain a DFA construction for any given regular language by using any right congruence of finite index that precisely[4] represents the language. In this dissertation, we are interested in using this idea to explore the relation between well-known minimization and determinization operations. This way, we aim to give a new perspective on the double-reversal method and its connection with the partition-based methods. This is the (rather general) question we are interested in.

Q1. *Can we use congruences of finite-index to give new insights on the double-reversal method and its connection with the state-partition-based techniques?*

It is worth to remark that, Brzozowski and Tamm [18] also propose a language-theoretic view when they address a more general version of the double-reversal method. In this case, they do not use congruences on words, but the notion of *atoms* of the language, i.e., sets of suffixes of the language of an special kind. Other uniform views to the double-reversal method have been proposed from a category-theoretic point of view [1, 12] as well as using the geometry of rectangular decompositions of relations on words [25]. On the other hand, most of the previous attempts to study the connection between the double-reversal method and the other minimization techniques rather focus on low-level automata aspects [20, 9, 43].

[4] A congruence *precisely* represents a language if the language can be represented as a union of equivalence classes of the congruence.

Conciseness between PDAs and CFGs for Parikh equivalence. In a pushdown automaton, the presence of a stack means that, unlike finite-state automata, these machines are able to "remember" an arbitrary amount of information, and therefore the number of possible configurations is infinite. This way, they recognize the class of context-free languages, which strictly contains the regular languages.

CFLs are also defined as the languages that can be generated by a context-free grammar. Thus, both representations are equivalent, in the sense that, given a CFG there exists a PDA that accepts the same language, and viceversa. In this dissertation, we are interested in comparing the descriptional complexity of both models. The notion of *descriptional complexity*, or *size*, of a formal structure (PDA or CFG) refers to the number of symbols needed to write down its description. Put in other words, the length of the string that results from writing down all the entries of its transition function, in the case of a PDA, or its set of rules, in the case of a CFG.

To be more precise, we are interested in comparing PDAs against CFGs when they represent Parikh-equivalent languages. However, to provide some context, let us first consider the problem of comparing both models when they represent the same language. In this sense, there exist standard translation procedures that convert one formalism into the other [52].

On one hand, the classical textbook conversion procedure from a CFG into a PDA [52] relies on simulating the derivations of the grammar, i.e., the sequences of replacements of a variable by a list of variables and terminals, using the stack of a PDA. This way, the grammar variables and the terminals define the stack alphabet of the PDA, which has one single state. Thus, for a CFG with $v \geq 1$ grammar variables and $|\Sigma| \geq 1$ terminals, this procedure produces a PDA with 1 state and $v + |\Sigma|$ stack symbols. Relying on this construction, it is easy to see that for every CFG generating some given CFL, there exists an equivalent PDA of the same size up to some constant. Thus, we conclude that CFGs are never more concise[5] than PDAs, for language equivalence.

On the other hand, the standard counterpart method [52] defines an equivalent grammar by identifying each variable with a "short" computation of the PDA that is represented by the source and final state of the computation and the topmost symbol on the stack when the computation started. The procedure also defines an extra variable which will be the initial. In consequence, for a PDA with $n \geq 1$ states and $p \geq 1$ stack

[5]The term *concise* is w.r.t. the size of the description.

symbols, the CFG that results from applying the conversion has at most $n^2p + 1$ variables if $n > 1$ (p if $n = 1$).

One natural question is whether there exist PDAs for which the smallest CFG needs that apparently large number of variables, and thus whether there exist context-free languages that can be defined much more concisely by PDAs than by CFGs. This problem was addressed by Goldstine et al. [45] in a paper where they introduced an infinite family of context-free languages whose representation by a pushdown automaton is more concise than by a context-free grammar. Incidentally, this result shows that the classical conversion procedure [52] of a PDA into a CFG is optimal in general, in the sense that, there is no other algorithm that produces a grammar with a fewer number of variables.

As we mentioned in the previous section, there exist more efficient methods [23, 78] to translate PDAs into equivalent CFGs when the PDA is deterministic and defined over a unary alphabet. In that case, the number of grammar variables is proportional to the product of the number of states and stack symbols of the PDA.

In this dissertation, we are interested in comparing PDAs against CFGs when language equivalence is relaxed to Parikh equivalence. Generally speaking, these are the question we are interested in.

> **Q2.** *What is the relation between the conciseness of PDAs and CFGs when language equivalence is relaxed to Parikh equivalence?*

Observe that, in the unary case, language and Parikh equivalence coincide, since if the alphabet only contains one element there is no notion of ordering in the symbols of the words. It follows that question Q2 can be answered solving the following stronger problem.

> **Q3.** *What is the relation between the conciseness of PDAs and CFGs when the alphabet is unary?*

It is worth to note that the family of Goldstine et al. [45] consists of deterministic pushdown automata defined over an alphabet of non-constant size (greater than one). In consequence, their family cannot be used to answer Q3.

Finally, Parikh's Theorem allows us to compare PDAs and NFAs for Parikh-equivalent languages.

Q4. *What is the relation between the conciseness of PDAs and NFAs for Parikh equivalence?*

Parikh's Theorem in the weighted case. Continuing our work under the Parikh equivalence assumption, we are interested in the problem of extending Parikh's Theorem to the weighted case. Broadly speaking, this problem asks, given a weighted pushdown automaton $\mathcal{P}$, whether there exists always a weighted finite-state automaton $\mathcal{A}$ that accepts a Parikh-equivalent language such that, for each word w, the sum of the weights of all words Parikh-equivalent to w in $\mathcal{P}$ coincides with that of all Parikh-equivalent words to w in $\mathcal{A}$.

This generalization has the potential of expanding the applications of this result on the analysis of multi-threaded asynchronous programs with procedures to systems where transitions are augmented with a weight that may represent the cost of performing the transition or the probability of an event associated to it. Finding a weighted finite-state automaton that is Parikh-equivalent to the original program and preserves the costs enables quantitative [22, 29, 83] and probabilistic analysis [5] of programs following this paradigm.

Petre [77] showed that Parikh's Theorem does not hold in the weighted case by means of a counterexample that defines[6] the following weighted context-free language over the alphabet $\{a\}$ with weights over the naturals:

$$\left\{(a^{2n+1}, C_n) \mid n \geq 0 \text{ and } C_n \text{ is the } n\text{-th Catalan number } C_n \stackrel{\text{def}}{=} \frac{1}{n+1}\binom{2n}{n}\right\} ,$$

for which no weighted finite-state automaton over the naturals exists recognizing the same set of words with the same weights.[7] The reader may check that this is the weighted language generated by the WCFG from Figure 1.4.

Thus, we are interested in exploring the following question:

Q5. *Under which conditions does Parikh's Theorem hold in the weighted case?*

and, more generally:

[6] Here we use a notation for weighted languages as sets of pairs $(w, x) \in \Sigma^* \times \mathbb{S}$ where $\mathbb{S}$ denotes the semiring of weights. In Chapter 5 we will define a more convenient notation for our purpose.

[7] Note that, this is enough since the alphabet they use is unary, i.e., Parikh equivalence and language equivalence coincide.

Q6. *Is the problem of determining if a WPDA satisfies Parikh's Theorem decidable?*

In the following section we give an overview of the state of the art of the proposed questions and a (rather detailed) description of the main contributions of this dissertation, including some references to the rest of the document, with the goal of guiding the reader through the key aspects of this thesis.

1.4 Contributions of This Thesis

1.4.1 A Perspective Through Congruences

In this dissertation we propose the use of equivalence relations over words on the alphabet Σ that describe finite partitions over Σ^* to approach the study of automata minimization algorithms. By interpreting these equivalences as language abstractions, we will devise the connection between the double-reversal method and Moore's algorithm, a partition-based minimization method. This way, we give a positive answer to question Q1.

First, given a right congruence of finite index and a language L that is precisely represented by the congruence, i.e., L is a union of equivalence classes, we will provide a DFA construction (Definition 3.2.1) recognizing L. By instantiating these automata constructions on two concrete right congruences (Definitions 3.3.1 and 3.3.3), we will identify the minimal DFA for a given regular language and the determinization operation for a given NFA. We will also define counterpart automata constructions for the left version of these congruences. From a left congruence of finite index, we will give an automata construction (Definition 3.2.3) that is *co-deterministic* (co-DFA), i.e., an automaton whose reverse is deterministic. As a result, we provide a framework of finite-state constructions based on congruences that offers new insights in the connection between the double-reversal method and Moore's algorithm, a partition-based method.

The contributions of this study come as follows.

A congruence-based perspective on the double-reversal method and its later generalization. The double-reversal alternates a reverse and a determinization operation twice relying on the fact that determinizing a co-DFA yields to the minimal DFA for the language. Recently, Brzozowski

and Tamm proposed a generalization of this method [18]. More pointedly, they showed a sufficient and necessary condition (having a co-DFA is only sufficient) that guarantees that determinizing an NFA $\mathcal{N}$ yields to the minimal DFA for the language of $\mathcal{N}$.

We show that the latter condition can be formulated in simple terms, interpreting congruences as language abstractions (Theorem 3.4.2). Using this formulation we give a simple and clean alternative proof of the double-reversal method by Brzozowski [16] in terms of congruences. In this regard, Figure 3.1 shows an overview of the method within our framework of congruence-based automata constructions. In the light of the proof it is easy to see how to generalize (Theorem 3.5.2) and improve the double-reversal method by producing intermediate co-DFAs with possibly fewer states.

A congruence-based perspective on Moore's algorithm. Using Theorem 3.4.2, we relate the iterations of Moore's partition refinement algorithm, which works on the states of the input DFA, to the iterations of the greatest fixpoint algorithm that builds Nerode's partition over words (Theorem 3.5.23).

An overview of other related automata constructions. Brzozowski and Tamm [18] showed that every regular language defines a unique NFA, the so-called *átomaton*. We locate this NFA in our framework of automata constructions (see Figure 3.3) and relate their results based on the so-called *atoms* [18] of the language within our congruence-based setting. The *atoms* of a language are non-empty intersections of complemented and uncomplemented sets of suffixes[8] of the language.

1.4.2 A Perspective Through Parikh Equivalence

Goldstine et al. [45] addressed the question of whether there exist PDAs for which the smallest language-equivalent CFG needs the seemingly large number of variables given by the standard textbook procedure [52]. In this dissertation, we revisit this question from the Parikh-equivalence perspective.

PDAs can be polynomially more concise than CFGs in the unary case and for Parikh equivalence. We explore question Q2 by looking

[8] More precisely, the sets of suffixes are the so-called *left quotients* of the language, i.e., the set of all suffixes of the language that have some given word as a prefix.

at question Q3. We define an infinite family of CFLs as Goldstine et al. [45] did but our family differs drastically from theirs. Given $n \geq 1$ and $k \geq 1$, each member of our family is given by a PDA with n states, $p = k+2n+1$ stack symbols and *one* input symbol (Definition 4.3.1). We show that, for each PDA of the family, every equivalent CFG has $\Omega(n^2(p-2n-4))$ variables (Theorem 4.3.4).

If the alphabet is a singleton, language equivalence and Parikh equivalence coincide. Therefore, we conclude that there is a family of CFLs that can be represented more concisely by a PDA than by a CFG, even if we are interested just in Parikh equivalence. This way, we extend the work of Goldstine et al. concluding that the textbook translation of a PDA into a language-equivalent CFG is *optimal*[9] in the unary case and for Parikh equivalence.

New polynomial time conversion method from unary deterministic PDAs to CFGs. We investigate the special case of deterministic PDAs over a unary alphabet. For this class of PDAs there exist more efficient translation procedures [23, 78], i.e., methods that produce CFGs with a number of variables proportional to the product of the number of states and stack symbols of the input PDA.

Now we give a new definition of an equivalent context-free grammar given a unary deterministic PDA that achieves the best known bounds [23] by optimizing the standard conversion algorithm (Theorems 4.3.7 and 4.3.8).

PDAs can be exponentially more concise than NFAs in the unary case and for Parikh-equivalence. Parikh's Theorem motivates the comparison of PDAs against NFAs. We use the same family of PDAs to derive an exponential lower bound on the number of states of every Parikh-equivalent NFA (Theorem 4.4.1). This answers question Q4. Moreover, relying on this lower bound, we show that a procedure chaining two existing constructions yields optimal results in the number of states of the resulting NFA.

1.4.3 Parikh Equivalence in The Weighted Case

Petre [77] showed that Parikh's Theorem does not hold in the weighted case. In consequence, we explore under which conditions the so-called Parikh property holds, as well as whether the property is decidable or

[9] We will precise the meaning of *optimal* in Chapter 3.

not. For these purposes, we adopt the grammar model, as opposed to the automata model. In the weighted setting, both representations are also equivalent, in the sense that WPDAs and WCFGs generate the same class of languages of weighted words. However, working with WCFGs allows us to exploit their connection with *algebraic systems of equations* to give more simple and convincing proofs of our results.

In the weighted setting, two WCFGs $\mathcal{G}_1$ and $\mathcal{G}_2$ are Parikh-equivalent if and only if for each Parikh equivalence class the semiring sum of the weights of all the words that are Parikh-equivalent w.r.t. $\mathcal{G}_1$ and $\mathcal{G}_2$ coincide. Thus, we say that a WCFG $\mathcal{G}_1$ satisfies the Parikh property if and only if there exists a *regular* WCFG $\mathcal{G}_2$ that is *Parikh-equivalent* to $\mathcal{G}_1$.

We propose an extension on known results regarding questions Q5 and Q6 as follows.

A sufficient condition for the Parikh property. It is well-known that the Parikh property holds if the semiring is commutative and idempotent[10] [10, 66, 68]. Furthermore, Luttenberger et al. [68] showed that a sufficient condition for a WCFG over the naturals to satisfy the Parikh property is that the WCFG is *nonexpansive*. This property depends on the structure of the grammar (as opposed to idempotence, which is a property on the weight domain) and, intuitively, implies that every word generated by the grammar can be processed with a stack of bounded depth. Baron and Kuich [8], who gave a similar characterization of nonexpansive grammars using *rational power series* to that of Luttenberger et al., conjectured that nonexpansiveness was also a necessary condition for the Parikh property. Recently, Bhattiprolu et al. [10] also explored the question of giving a sufficient condition, providing a class of WCFGs over the unary alphabet that always satisfies the condition.

We advance on question Q5 in the following way. We rely on the result of Luttenberger et al. to give a Parikh-equivalent regular WCFG construction for a given nonexpansive WCFG defined over *any* commutative semiring (Theorem 5.2.2). Moreover, we show that the latter property is sufficient but not necessary by means of a counterexample (Example 5.2.3). This shows that the conjecture formulated by Baron and Kuich is false, even when the alphabet is unary. Finally, we prove that the class described by Bhattiprolu et al. is strictly contained in the class of nonexpansive grammars (see Section 5.6.2).

[10] A semiring $\mathbb{S}$ is *idempotent* iff $x + x = x$, for all $x \in \mathbb{S}$

A decision procedure for the Parikh Property over the rationals. To the best of our knowledge, question Q6 on the decidability of the Parikh property is open. However, it implicitly follows from a result by Kuich and Salomaa [67] that, when we equivalently formulate the property in terms of formal power series, it is decidable over the semiring of rational numbers. Their proof relies on a cumbersome elimination procedure which is hard to perform even on toy examples.

We give a decision procedure that sidesteps this problem by applying an alternative standard technique: *Groebner bases* (Theorem 5.3.14). This method allows to illustrate the algorithm on examples with the support of mainstream open-source computer algebra systems.

1.5 Thesis Structure

First, we introduce common notation and preliminary definitions in Section 2. More specific notation might be deferred to the beginning of each chapter. We give a congruence-based perspective on finite-state automata minimization methods in Section 3. In Section 4 we compare pushdown automata against context-free grammars and finite-state automata for the Parikh image of context-free languages. We continue our work under the Parikh equivalence assumption in Section 5 where we address the weighted extension of Parikh's Theorem. Finally, we conclude this dissertation and discuss about future work in Section 6. At the end of the document we collect a list of acronyms used in this thesis. A reference to their first use in each section is included.

2 Preliminaries

In this section, we introduce the general notation and definitions that we will use throughout this dissertation. In some cases, we defer more specific notation to the beginning of the chapter where it is used. We start with the fundamental elements of Automata Theory: alphabets, words and languages.

2.1 Alphabets, Words and Languages

An *alphabet* is a nonempty finite set of symbols, generally denoted by Σ. When Σ is a singleton, we say that the alphabet is *unary*.

A *word* is a finite sequence of symbols over the alphabet Σ. If the sequence is empty, we denote the word by ε, the *empty string*. Otherwise, the finite sequence is given by $w \stackrel{\text{def}}{=} a_1 \cdots a_n$ where $a_i \in \Sigma$, for each $i \in \{1, \ldots, n\}$ and $n \geq 1$. We say that n is the *length* of w, and we denote it by $|w| = n$. If $w = \varepsilon$ then $|w| = 0$. We use the notation $(w)_i$ to denote the i-th symbol in the sequence $w \in \Sigma^*$, if $1 \leq i \leq |w|$; otherwise $(w)_i = \varepsilon$.

The set of all words over Σ including the empty string is denoted by Σ^*. Sometimes we use Σ^+ to denote the set $\Sigma^* \setminus \{\varepsilon\}$. We define a *language* L over Σ as any set of words over the alphabet Σ, i.e., any subset of Σ^*. The *size* of L is the cardinality of the set L, denoted by $|L|$.

Given a word $w = a_1 \cdots a_n \in \Sigma^*$, we define the *reverse* of w as $w^R \stackrel{\text{def}}{=} a_n \cdots a_1$. Note that $\varepsilon^R = \varepsilon$. Consequently, given a language L, we define the *reverse* language of L as $L^R \stackrel{\text{def}}{=} \{w^R \mid w \in L\}$. Finally, we define the *complement* language of L as $L^c \stackrel{\text{def}}{=} \{w \mid w \notin L\}$.

Next, we define the algebraic structure used for the domain of the weights of weighted finite-state automata, weighted pushdown automata

and weighted context-free grammars.

2.2 Semirings

A *semiring* is an algebraic structure $(\mathbb{S}, +, \cdot, 0_{\mathbb{S}}, 1_{\mathbb{S}})$ where $(\mathbb{S}, +, 0_{\mathbb{S}})$ is a commutative monoid, i.e., a set equipped with an associative and commutative binary operation $+$ with $0_{\mathbb{S}}$ as *identity* for this operation; $(\mathbb{S}, \cdot, 1_{\mathbb{S}})$ is a monoid with identity $1_{\mathbb{S}}$; the operation $\cdot$ distributes over $+$, and $0_{\mathbb{S}}$ satisfies that $x \cdot 0_{\mathbb{S}} = 0_{\mathbb{S}} \cdot x = 0_{\mathbb{S}}$, for all $x \in \mathbb{S}$. A semiring is *commutative* iff $x \cdot y = y \cdot x$, for every $x, y \in \mathbb{S}$. In this dissertation, we will always assume that semirings are commutative, for a reason that will be clear when we define weighted context-free grammars (see Remark 2.4.11). A semiring is *idempotent* iff $x + x = x$, for all $x \in \mathbb{S}$.

A *ring* is a semiring where $(\mathbb{S}, +, 0_{\mathbb{S}})$ is a commutative *group*, i.e., every element in $\mathbb{S}$ has an additive inverse. Recall that the *additive inverse* of an element $x \in \mathbb{S}$ is another element $y \in \mathbb{S}$ such that $x + y = 0_{\mathbb{S}}$.

A *field* is a ring where $(\mathbb{S} \setminus \{0_{\mathbb{S}}\}, \cdot, 1_{\mathbb{S}})$ is a commutative group, i.e., every element has multiplicative inverse. Recall that the *multiplicative inverse* of an element $x \in \mathbb{S} \setminus \{0_{\mathbb{S}}\}$ is another element $y \in \mathbb{S}$ such that $x \cdot y = 1_{\mathbb{S}}$.

We will sometimes use $\mathbb{S}$ for both the structure and the underlying set when the meaning is clear from the context. We will abuse notation and use $+$ and $\cdot$ to denote the ordinary sum and product in $\mathbb{N}$, $\mathbb{Q}$ and $\mathbb{R}$.

Some classical examples of commutative semirings are $(\mathbb{N}, +, \cdot, 0, 1)$, $(\mathbb{Q}, +, \cdot, 0, 1)$ and $(\mathbb{R}, +, \cdot, 0, 1)$. The two latter examples are also fields. Common examples of commutative and idempotent semirings are the *Boolean semiring* defined as $\mathbb{B} = (\{0, 1\}, +, \cdot, 0, 1)$ and satisfying the equation $1 + 1 = 1$; and the *tropical semiring* over the naturals defined as $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$.

2.3 Automata

In this section, we introduce basic definitions of different automata representations that we will use throughout this dissertation. All these concepts are standard and covered in several books on Automata Theory [81, 52, 30]. Let us start with the notion of (nondeterministic) *finite-state automaton*.

Definition 2.3.1 (Finite-state automaton)**.** A (nondeterministic) *finite-state automaton* (NFA), or simply, *automaton*, is a 5-tuple $\mathcal{N} = (Q, \Sigma, \delta, I, F)$

where Q is a nonempty finite set of *states*, Σ is an alphabet of symbols, $I \subseteq Q$ are the *initial* states, $F \subseteq Q$ are the *final* states, and δ is a finite subset of $Q \times \Sigma \times Q$ called the *transition* relation.

Each element $(q, a, q') \in \delta$ is called a *transition*, where $q, q' \in Q$ are the *source state* and *destination state* of the transition, respectively, and $a \in \Sigma$ is the symbol the transition *reads*. We often use the alternative notation $q' \in \delta(q, a)$ to denote the fact $(q, a, q') \in \delta$.

A *path* $p = (q_0, a_1, q_1)(q_1, a_2, q_2) \cdots (q_{n-1}, a_n, q_n)$ of $\mathcal{N}$, with $n \geq 1$, is defined as a nonempty sequence of *consecutive* transitions of δ, i.e., the destination state of every transition in the sequence (except for the last transition) coincides with the source state of the next transition. We define the *source* of p as $\mathrm{s}(p) \stackrel{\text{def}}{=} q_0$, the *destination* of p as $\mathrm{d}(p) \stackrel{\text{def}}{=} q_n$, and we say that p *reads* the word $a_1 \cdots a_n$. We denote by $\mathrm{paths}_{\mathcal{N}}(w)$ the set of all paths of $\mathcal{N}$ reading w.

Given two sets of states $S, D \subseteq Q$, we define the language of all words read by a path of $\mathcal{N}$ with source state in S and destination state in D as follows. Define $W^{\mathcal{N}}_{S,D} \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid \exists p \in \mathrm{paths}_{\mathcal{N}}(w) : \mathrm{s}(p) \in S \text{ and } \mathrm{d}(p) \in D\}$. When S or D are singletons $\{q\}$ we write q instead of $\{q\}$. In particular, we define the *right language* of the state q as $W^{\mathcal{N}}_{q,F}$ and the *left language* of q as $W^{\mathcal{N}}_{I,q}$. We say that a state q is *empty* iff $W^{\mathcal{N}}_{q,F} = \varnothing$, and *unreachable* when $W^{\mathcal{N}}_{I,q} = \varnothing$.

We also define the set of states in $\mathcal{N}$ that are reached from S when reading the word w, and the set of states from which S is reached when reading w. Formally, $\mathrm{post}^{\mathcal{N}}_w(S) \stackrel{\text{def}}{=} \{q \in Q \mid \exists p \in \mathrm{paths}_{\mathcal{N}}(w) : \mathrm{s}(p) \in S \text{ and } q = \mathrm{d}(p)\}$ and $\mathrm{pre}^{\mathcal{N}}_w(S) \stackrel{\text{def}}{=} \{q \in Q \mid \exists p \in \mathrm{paths}_{\mathcal{N}}(w) : q = \mathrm{s}(p) \text{ and } \mathrm{d}(p) \in S\}$. We will omit the superscript $\mathcal{N}$ from all the terms defined above when it is clear from the context.

Finally, we define the *language* of an NFA $\mathcal{N}$ as $\mathcal{L}(\mathcal{N}) \stackrel{\text{def}}{=} \bigcup_{q \in I} W_{q,F} = \bigcup_{q \in F} W_{I,q} = W_{I,F}$. We usually say that $\mathcal{N}$ *recognizes* or *accepts* the language $\mathcal{L}(\mathcal{N})$, while we say that every word in $\mathcal{L}(\mathcal{N})^c$ is *rejected* by $\mathcal{N}$. If two NFAs recognize the same language, we say they are *language-equivalent*, or simply equivalent.

There exists a class of NFAs that enjoys *determinism*, i.e., they only have one initial state and there exists exactly one transition to a destination state for each pair of source state and alphabet symbol. Thus, the transition relation of deterministic automata can be interpreted as a total function from $Q \times \Sigma$ to Q. It is well-known that this class of automata is equivalent to the nondeterministic automata, namely, every NFA is equivalent to a

deterministic automaton [52]. We define *deterministic finite-state automata* as follows.

Definition 2.3.2 (Deterministic finite-state automaton)**.** A *deterministic finite-state automaton* (DFA) $\mathcal{D} = (Q, \Sigma, \delta, I, F)$ is an NFA such that $I = \{q_0\}$ and, for every $q \in Q$ and $a \in \Sigma$, there exists exactly one $q' \in Q$ such that $(q, a, q') \in \delta$.

According to this definition, DFAs are always *complete*, i.e., for each source state $q \in Q$ and alphabet symbol $a \in \Sigma$, there exists always a destination state $q' \in Q$ such that $(q, a, q') \in \delta$.

Let us recall here the finite-state automaton given in Figure 1.2. Note that it is deterministic, and there is no other DFA that accepts the same language and has fewer states. Therefore, it is the *minimal* DFA for the language $\{a^n b^m \mid n, m \geq 1\}$.

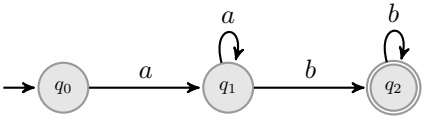


Figure 2.1: The minimal DFA accepting the regular language $\{a^n b^m \mid n, m \geq 1\}$.

Now we recall the *subset automata construction* for a given NFA $\mathcal{N}$ which yields a DFA for the language of $\mathcal{N}$ [52].

Definition 2.3.3 (Subset construction)**.** Given an NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$, define the DFA $\mathcal{N}^D \stackrel{\text{def}}{=} (Q_D, \Sigma, \delta_D, I_D, F_D)$ where $Q_D \stackrel{\text{def}}{=} \{S \mid S \subseteq Q\}$, $I_D \stackrel{\text{def}}{=} \{q \mid q \in I\}$, $F_D \stackrel{\text{def}}{=} \{S \subseteq Q \mid S \cap F \neq \varnothing\}$ and $\delta_D(S, a) \stackrel{\text{def}}{=} \bigcup_{q \in S} \delta(q, a)$, for each $S \in Q_D$ and $a \in \Sigma$.

It follows from the definition that the states of $\mathcal{N}^D$ correspond to the *power set* of Q, also denoted by $\wp(Q)$. Nevertheless, we will only consider as states of $\mathcal{N}^D$ the elements of $\wp(Q)$ that are *reachable* from its initial state. Thus, $\mathcal{N}^D$ possibly contains empty states, but no state is unreachable.

By reversing all the *arrows* of an automaton $\mathcal{N}$, i.e., by flipping the source and destination state of every transition; and by switching the initial and final states, we obtain the so-called *reverse* automaton of $\mathcal{N}$.

Definition 2.3.4 (Reverse construction)**.** Given an NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$, *the reverse NFA* of $\mathcal{N}$ is the NFA defined as $\mathcal{N}^R \stackrel{\text{def}}{=} (Q, \Sigma, \delta^R, F, I)$ where $(q, a, q') \in \delta^R$ iff $(q', a, q) \in \delta$.

Every word in the language of $\mathcal{N}^R$ is the reverse of the corresponding word in the language of $\mathcal{N}$, and thus $\mathcal{L}(\mathcal{N}) = (\mathcal{L}(\mathcal{N}))^R$ [81]. Below these lines we show the reverse automaton of that of Figure 2.1.

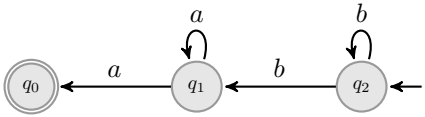


Figure 2.2: Reverse automaton of the DFA given in Figure 2.1. Note that it accepts the language $\{b^n a^m \mid n, m \geq 1\}$, which is the reverse language of $\{a^n b^m \mid n, m \geq 1\}$, the language of the automaton in Figure 1.2.

The notion of reverse automaton allows us to define *co-deterministic automata* as follows.

Definition 2.3.5 (Co-deterministic finite-state automaton)**.** An NFA $\mathcal{N}$ is a *co-deterministic finite-state automaton* (co-DFA) iff $\mathcal{N}^R$ is deterministic.

In this case, co-DFAs are always *co-complete*, i.e., for each target state $q' \in Q$ and $a \in \Sigma$, there exists always a source state $q \in Q$ such that $(q, a, q') \in \delta$. Observe that the NFA in Figure 2.2 is a co-DFA since its reverse is deterministic.

While nondeterministic automata (and equivalently, deterministic ones) recognize the class of *regular* languages, the more general class of *context-free languages* requires a more sophisticated class of automata to be finitely represented, namely, *pushdown automata*. Roughly speaking, pushdown automata are finite-state automata augmented with a *stack*. Whereas the size of the stack alphabet is finite, the number of symbols pushed onto the stack can be arbitrarily large.

Definition 2.3.6 (Pushdown automaton)**.** A (nondeterministic) *pushdown automaton* (PDA) is a 6-tuple $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ where Q is a finite nonempty set of *states* including q_0, the *initial* state, Σ is the *input* alphabet, Γ is the *stack alphabet* including Z_0, the initial stack symbol; and δ is a finite subset of $Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \times Q \times \Gamma^*$ called the *actions*.

We write $(q, X) \hookrightarrow_a (q', \beta)$ to denote an action $(q, X, a, q', \beta) \in \delta$, where q is the *source* state of the action, q' the *target* state, X the symbol the action *pops* and β, the (possibly empty) sequence of symbols the action *pushes*. We sometimes omit the subscript of the arrow.

An *instantaneous description* (ID) of a PDA is a pair (q, β) where $q \in Q$ and $\beta \in \Gamma^*$. We call the first component of an ID the *state* and the second component, the *stack content*. The *initial* ID consists of the initial state and the initial stack symbol. When reasoning formally, we use the functions state$(\cdot)$ and stack$(\cdot)$ which, given an ID, returns its state and the stack content, respectively.

We say that an action $(q, X) \hookrightarrow_a (q', \beta)$ is *enabled* at ID I iff state$(I) = q$ and $(\,\text{stack}(I)\,)_1 = X$. Given an ID $(q, X\gamma)$ enabling $(q, X) \hookrightarrow_a (q', \beta)$, define the *successor ID* to be $(q', \beta\gamma)$. We denote this fact by $(q, X\gamma) \vdash_a (q', \beta\gamma)$, and call it a *move* that *consumes* a from the input.[1] We sometimes omit the subscript of $\vdash$ when the symbol consumed is not important. A *move sequence* $I_0 \vdash_{a_1} \vdash \cdots \vdash_{a_n} I_n$ is a finite sequence of IDs $I_0 I_1 \cdots I_n$ with $n \geq 1$ such that $I_i \vdash_{a_{i+1}} I_{i+1}$, with $0 \leq i \leq n-1$. We say that the move sequence *consumes* the word $w = a_1 \cdots a_n$ from the input. We concisely denote this fact as $I_0 \vdash .^w. \vdash I_n$. A move sequence $I \vdash \cdots \vdash I'$ is a *quasi-run* when $|\text{stack}(I)| = 1$ and $|\text{stack}(I')| = 0$; and a *run* when, furthermore, I is the initial ID. Finally, we define the *language* of a PDA $\mathcal{P}$ as $\mathcal{L}(\mathcal{P}) \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid \mathcal{P} \text{ has a run consuming } w\}$.

We have assumed that a PDA accepts its input by consuming it and emptying its stack at the same time. This assumption is called *acceptance by empty stack*. Let us recall here the PDA from Figure 1.3 which accepts by empty stack.

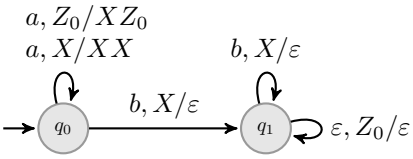


Figure 2.3: A pushdown automaton accepting the context-free language $\{a^n b^n \mid n \geq 1\}$.

Remark 2.3.7. There is a second approach to define the language of a PDA, the so-called *acceptance by final states*, that establishes that a PDA

[1] When $a = \varepsilon$ the move does not consume input.

accepts its input by consuming it and entering a state from a subset $F \subseteq Q$ of *final* states. Both modes of acceptance are equivalent, in the sense that every context-free language L has a PDA that accepts it by empty stack if and only if L has a PDA that accepts it by final states (see Theorems 6.9 and 6.11 in [52]). We give an example of a PDA accepting by final states in Figure 2.4.

As for finite-state automata, one can define the class of *deterministic* pushdown automata, where no more than one move is enabled at each ID.

Definition 2.3.8 (Deterministic pushdown automata)**.** A *deterministic pushdown automaton* (DPDA) is a PDA $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ satisfying the two following properties:

1. for every $q \in Q, X \in \Gamma$ and $a \in \Sigma \cup \{\varepsilon\}$, $|\delta(q, a, X)| \leq 1$ and,
2. for every $q \in Q$ and $X \in \Gamma$, if $\delta(q, \varepsilon, X) \neq \varnothing$ then $\delta(q, a, X) = \varnothing$ for every $a \in \Sigma$.

Note that the PDA given in Figure 1.3 is deterministic, as it satisfies the two conditions above.

Remark 2.3.9. Unlike general PDAs, the two modes of acceptance of DPDAs, i.e., by empty stack and by final state, are not the same. In fact, both automata models define different proper subclasses of context-free languages. To be precise, the class of languages accepted by a DPDA by final states strictly includes the regular languages and is strictly included in the context-free languages, while the class of languages accepted by a DPDA by empty stack is strictly included in that of the languages accepted by a DPDA by final state.[2] We define the class of *deterministic context-free languages* (DCFLs) as the languages that can be accepted by a DPDA by final states. A classical example of a context-free language that cannot be accepted by a DPDA by final states is the language of even-length palindromes over the alphabet $\Sigma = \{a, b\}$, i.e., $L \stackrel{\text{def}}{=} \{w^R w \mid w \in \Sigma^*\}$ [52].

[2]More pointedly, the class of languages accepted by DPDAs by empty stack is exactly the class of languages accepted by DPDAs by final states that are *prefix-free*, i.e., no word in the language is the prefix of another word in the language. For instance, the language a^* is not prefix-free, and thus no DPDA by empty stack accepts it. Since a^* is regular, the languages accepted by DPDAs by empty stack do not include the regular languages.

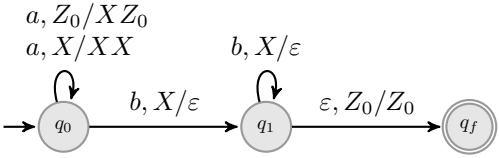


Figure 2.4: A PDA that accepts by final states the same language as that of Figure 1.3, i.e., $\{a^n b^n \mid n \geq 1\}$. In this case, its set of final states is $F = \{q_f\}$. Note that it is also deterministic. Therefore, $\{a^n b^n \mid n \geq 1\}$ is a DCFL.

2.4 Grammars

Regular languages and the more general class of context-free languages have also a finite recursive representation as grammars. We will first introduce the definition of context-free grammars and then, we will define regular grammars as a particular case.

Definition 2.4.1 (Context-free grammar)**.** A *context-free grammar* (CFG) is a 4-tuple $\mathcal{G} = (V, \Sigma, S, R)$ where V is a finite set of *variables* including S, the *start* variable, Σ is the *alphabet* or set of *terminals* and $R \subseteq V \times (V \cup \Sigma)^*$ is a finite set of *rules*.

We will denote the rules as $X \to \alpha$, with $X \in V$ and $\alpha \in (V \cup \Sigma)^*$, where X and α are the *head* and the *body* of R, respectively. We will represent all the rules of the grammar sharing the same head by listing the head variable once, followed by all the bodies of rules with that variable separated by a vertical line. For instance, if $X \to \alpha$ and $X \to \beta$ are all the rules with head X in a given grammar, then we will write these two rules compactly as $X \to \alpha \mid \beta$.

CFGs *generate* strings by means of *derivations*, i.e., finite sequences of steps that, starting from the start variable, replace any variable of the current string by the body of one of its rules until the resulting string contains only terminals. To describe each step of a derivation we use the relation symbol $\Rightarrow$.

Formally, given a CFG $\mathcal{G} = (V, \Sigma, R, S)$ and a rule $\pi = X \to \alpha \in R$, we write $\beta\, X\, \gamma \overset{\pi}{\Rightarrow} \beta\, \alpha\, \gamma$ to denote one *derivation step* of $\mathcal{G}$, with $\beta, \gamma \in (V \cup \Sigma)^*$. We usually omit the superscript of $\Rightarrow$ when it is not important. We say that $\beta\, X\, \gamma$ and $\beta\, \alpha\, \gamma$ are *derivation sentences* of $\mathcal{G}$. A *derivation sequence* is a sequence $\alpha_1 \overset{\pi_1}{\Rightarrow} \alpha_2 \overset{\pi_2}{\Rightarrow} \ldots \overset{\pi_n}{\Rightarrow} \alpha_n$ where each $\alpha_i \overset{\pi_i}{\Rightarrow} \alpha_{i+1}$ is a

derivation step. We use the symbol $\Rightarrow^*$ to denote zero or more steps of a derivation sequence, and $\Rightarrow^+$ to denote one or more steps of a derivation sequence. Finally, we define the language *generated* by a CFG $\mathcal{G}$ as $\mathcal{L}(\mathcal{G}) \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid S \Rightarrow_{\mathcal{G}}^+ w\}$.

Example 2.4.2. Consider the CFG $\mathcal{G} = (\{X\}, \{a, b\}, R, X)$ where the set of rules R is given as:

$$X \to aXb \mid ab \ .$$

This CFG generates the same language as the PDAs from Figures 1.3 and 2.4, i.e., $\{a^n b^n \mid n \geq 1\}$. An example of a derivation sequence of this grammar is $X \Rightarrow aXb \Rightarrow aaXbb \Rightarrow aaabbb$. ◀

In order to represent derivation sequences, it is common to use *parse trees* [52], i.e., tree structures that describe how the terminals of a string are grouped into substrings, each generated by one variable of the grammar.

We will write parse trees as *labeled trees*. Since this is not the only use we will do of labeled trees (see Section 4.2), we will give here a formal definition and some related notions.

Definition 2.4.3 (Labeled tree)**.** A *labeled tree* $c(\tau_1, \ldots, \tau_n)$ $(n \geq 0)$ is a finite tree whose nodes are labeled, where c is the label of the root and $\tau_1, \ldots, \tau_n$ are labeled trees, the children of the root.

When $n = 0$ we prefer to write c instead of $c()$. Each labeled tree τ defines a sequence, denoted $\overline{\tau}$, obtained by removing the symbols '(', ')' or ',' when interpreting τ as a string, e.g., $\overline{c(c_1, c_2(c_{21}))} = c\, c_1\, c_2\, c_{21}$. The *size* of a labeled tree τ, denoted $|\tau|$, is given by $|\overline{\tau}|$, and it coincides with the number of nodes in τ.

We recall the notion of *dimension* of a labeled tree [33] and we relate dimension and size of labeled trees in Lemma 2.4.6.

Definition 2.4.4 (Dimension of a labeled tree)**.** The *dimension* of a labeled tree τ, denoted as $d(\tau)$, is inductively defined as follows. $d(\tau) = 0$ if $\tau = c$, otherwise we have $\tau = c(\tau_1, \ldots, \tau_k)$ for some $k > 0$ and

$$d(t) = \begin{cases} max_{i \in \{1,\ldots,k\}} d(\tau_i) & \text{if there is a unique maximum,} \\ max_{i \in \{1,\ldots,k\}} d(\tau_i) + 1 & \text{otherwise.} \end{cases}$$

Example 2.4.5. Consider the labeled tree $\tau = c_1(c_2(c_3, c_3), c_4)$, depicted in Figure 2.5. The sequence $\bar{t}$ is $c_1\, c_2\, c_3\, c_3\, c_4$ and $|t| = 5$. The annotation $\overset{d(\tau)}{\tau}(\ldots)$ shows that τ has dimension 1:

$$\overset{1}{c_1}\,(\overset{1}{c_2}\,(\overset{0}{c_3}, \overset{0}{c_3}), \overset{0}{c_4})\ .$$

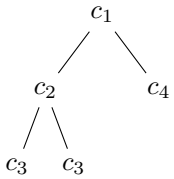


Figure 2.5: Depiction of the tree $c_1(c_2(c_3, c_3), c_4)$.

◀

Lemma 2.4.6. *Let τ be a labeled tree. Then, $|\tau| \geq 2^{\,d(\tau)}$.*

Proof. By induction on $|\tau|$.
Base case. Since $|\tau| = 1$ necessarily $\tau = a$ and $d(\tau) = 0$. Hence $1 \geq 2^0$.
Inductive case. Let $\tau = a(a_1(\ldots), \ldots, a_r(\ldots))$ with $r \geq 1$. We study two cases. Suppose there is a unique subtree $\tau_x = a_x(\ldots)$ of τ with $x \in \{1, \ldots, r\}$ such that $d(\tau_x) = d(\tau)$. As $|\tau_x| < |\tau|$, the induction hypothesis shows that $|\tau_x| \geq 2^{\,d(\tau_x)} = 2^{\,d(\tau)}$, hence $|\tau| \geq 2^{\,d(\tau)}$.

Next, let $r \geq 2$ and suppose there are at least two subtrees $\tau_x = a_x(\ldots)$ and $\tau_y = a_y(\ldots)$ of τ with $x, y \in \{1, \ldots, r\}$ and $x \neq y$ such that $d(\tau_x) = d(\tau_y) = d(\tau) - 1$. As $|\tau_x| < |\tau|$, the induction hypothesis shows that $|\tau_x| \geq 2^{\,d(\tau_x)}$. Applying the same reasoning to τ_y we conclude from $|\tau| \geq |\tau_x| + |\tau_y|$ that $|\tau| \geq 2^{\,d(\tau_x)} + 2^{\,d(\tau_y)} = 2 \cdot 2^{\,d(\tau)-1} = 2^{\,d(\tau)}$.

Thus, we write a parse tree as the labeled tree $\tau = \pi(\tau_1, \ldots, \tau_n)$ to denote that the topmost level of τ is induced by the grammar rule π and has exactly n children nodes which root (from left to right) the parse trees $\tau_1, \ldots, \tau_n$. This means that the body of π contains n grammar variables

where the i-th (from the left) is derived according to τ_i. We define the *yield* of a parse tree $\tau = \pi(\tau_1, \ldots, \tau_n)$, denoted as $\mathcal{Y}(\tau)$ inductively as follows. If $n = 0$, then $\mathcal{Y}(\tau) = \alpha$ where π is of the form $X \to \alpha$ and $\alpha \in (V \cup \Sigma)^*$. Otherwise, $\mathcal{Y}(\tau) = \alpha_1 \mathcal{Y}(\tau_1) \ldots \alpha_n \mathcal{Y}(\tau_n) \alpha_{n+1}$ where π is of the form $X \to \alpha_1 X_1 \ldots \alpha_n X_n \alpha_{n+1}$ with $\alpha_i \in (V \cup \Sigma)^*$, and each X_i corresponds to the head of the rule in the root of τ_i.

Note that, for each parse tree, there might be several ways to construct a derivation sequence if no restriction is imposed on the number of choices of variables to be replaced throughout the derivation. Thus, we will assume that the derivation policy for CFGs, i.e., the derivation strategy that determines the next variable to replace at each derivation step, defines one unique derivation sequence for each parse tree. One example of such a derivation policy is the so-called *left-most derivation policy*, which always replaces the *leftmost* variable in the derivation sentence at each step of the derivation.

Note also that there might be several parse trees that yield to the same word. Thus, we define the multiplicity or *ambiguity* of a word as the number of different parse trees that yield to the word. If a CFG generates every word with ambiguity one, we say that the CFG is *unambiguous*. Otherwise, it is *ambiguous*.

Additionally, we will assume that CFGs are *cycle-free*, i.e., no derivation sequence is of the form $X \Rightarrow^+ X$, with $X \in V$. This guarantees the convenient property that the ambiguity of every word generated by a CFG is always *finite*.

Context-free languages are the class of languages generated by context-free grammars. On the other hand, the regular languages are those generated by *regular* context-free grammars.

Definition 2.4.7 (Regular context-free grammar)**.** A *regular context-free grammar* is a CFG $\mathcal{G} = (V, \Sigma, S, R)$ such that, for every rule $X \to \beta \in R$, $\beta \in \Sigma^+(V \cup \{\varepsilon\})$.

In the above definition we assume that regular CFGs are *right-regular*, as opposed to *left-regular*[3] CFGs. Both formalisms are equivalent as they both describe the class of regular languages. Sometimes we refer to regular CFGs simply as *regular grammars*.

[3]Left-regular CFGs are context-free grammars such that, for every rule $X \to \beta \in R$, $\beta \in (V \cup \{\varepsilon\})\,\Sigma^+$.

Example 2.4.8. Consider the CFG $\mathcal{G} = (\{X, Y, Z\}, \{a, b\}, R, X)$ where the set of rules R is given as:

$$\begin{aligned} X &\to aY \\ Y &\to aY \mid bZ \mid b \\ Z &\to bZ \mid b \ . \end{aligned}$$

First, note it is a regular CFG as no right-hand side of a rule contains more than one variable, and if so, it occurs at the right of the terminal symbol. This CFG generates the same regular language as the DFA from Figure 1.2, i.e., $\{a^n b^m \mid n, m \geq 1\}$. ◀

Finally, we recall the classical textbook *translation procedure of a PDA into a CFG* [52] generating the same language that the PDA accepts by *empty stack*.

Definition 2.4.9 (Standard translation procedure from a PDA to a CFG)**.** Given a PDA $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$, define the $\mathcal{G} = (V, \Sigma, S, R)$ where:

- The set V of variables — often called the *triples* — is given by

$$\{[qXq'] \mid q, q' \in Q, X \in \Gamma\} \cup \{S\} \ . \tag{2.1}$$

- The set R of production rules is given by

$$\begin{aligned} &\{S \to [q_0 Z_0 q] \mid q \in Q\} \\ \cup\ &\{[qXr_d] \to b[q'(\beta)_1 r_1] \dots [r_{d-1}(\beta)_d r_d] \\ &\qquad \mid (q, X) \hookrightarrow_b (q', \beta), d = |\beta|, r_1, \dots, r_d \in Q\} \end{aligned} \tag{2.2}$$

For a proof of correctness, see Theorem 6.14 in [52]. The previous definition easily translates into a *conversion algorithm*. Observe that the runtime of such algorithm depends polynomially on $|Q|$ and $|\Gamma|$, but exponentially on $|\beta|$.

2.4.1 Weighted case

We introduce the more general grammar model of *weighted context-free grammars*, where a weight from a semiring is assigned to each rule of the grammar. The notion of weight is extended from rules to parse trees by multiplying the weights of the rules used along a tree, and from parse trees to words by adding the weights of all the possible parse trees that yield

to a word. Note that WCFGs are a generalization of CFGs in the sense that every *unweighted* CFG, as given in Definition 2.4.1, is a weighted context-free grammar with weights over the Boolean semiring $\mathbb{B}$.

Definition 2.4.10 (Weighted context-free grammar)**.** A *weighted context-free grammar* (WCFG) over the semiring $(\mathbb{S}, +, \cdot, 0_{\mathbb{S}}, 1_{\mathbb{S}})$ is a pair $(\mathcal{G}, W_{\mathcal{G}})$ where $\mathcal{G} = (V, \Sigma, S, R)$ is a CFG and $W_{\mathcal{G}}$ is a *weight function* defined as $W_{\mathcal{G}} : R \to \mathbb{S}$ that assigns a weight from the semiring $\mathbb{S}$ to each rule in R.

We will often omit the subscript of the weight function when it is clear from the context. We extend the definition of W from rules to derivation sequences by assigning to each derivation sequence ψ a weight which is the product of the weights of the rules applied in ψ.

Remark 2.4.11. In this dissertation we assume that the $\cdot$ operation of $\mathbb{S}$ is commutative, i.e., we are interested in commutative semirings. Otherwise, the weight of a derivation sequence would depend on the choice of the derivation policy.

We define the *weight of a parse tree* $\tau = \pi(\tau_1, \ldots, \tau_n)$ inductively as follows:

$$W(\tau) \stackrel{\text{def}}{=} W(\pi) \cdot \prod_{i=1}^{n} W(\tau_i) \ .$$

If $n = 0$, then $W(\tau) \stackrel{\text{def}}{=} W(\pi)$.

We define $\mathcal{T}_{\mathcal{G}}$, or simply $\mathcal{T}$, as the set of all parse trees of a CFG $\mathcal{G}$. Then, we define the *weight of a word* $w \in \Sigma$ as follows:

$$W(w) \stackrel{\text{def}}{=} \sum_{\substack{\mathcal{Y}(\tau) = w \\ \tau \in \mathcal{T}}} W(\tau) \ .$$

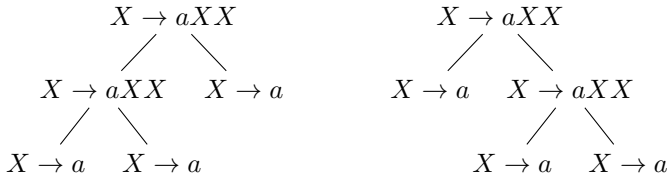
If for some $w \in \Sigma^*$, the set $\{\tau \mid \mathcal{Y}(\tau) = w, \tau \in \mathcal{T}\} = \varnothing$ then $W(w) = 0_{\mathbb{S}}$.

Finally, we define the *semantics* of a WCFG $(\mathcal{G}, W)$ as the function $[\![\mathcal{G}]\!]_W : \Sigma^* \to \mathbb{S}$ such that $[\![\mathcal{G}]\!]_W(w) \stackrel{\text{def}}{=} W(w)$.

Example 2.4.12. Recall the WCFG $(\mathcal{G}, W)$ given in Figure 1.4, where $\mathcal{G} = (\{X\}, \{a\}, R, X)$ and the weight function $W : R \to \mathbb{N}$ is given as:

$$\begin{array}{ll} X \to aXX & \mathbf{1} \\ X \to a & \mathbf{1} \end{array}$$

Recall also that the weight of the word a^5 is 2. Let us show here how to compute it. First, a^5 is the yield of 2 parse trees τ_1 and τ_2 in the grammar (see Figure 2.6). The weight of each parse is $W(\tau_1) = W(\tau_2) = 1$,



(a) Parse tree τ_1. (b) Parse tree τ_2.

Figure 2.6: Parse trees that yield to a^5.

since the weight of every rule is 1. Thus, the weight of the word a^5 is $W(a^5) = W(\tau_1) + W(\tau_2) = 2$. Since the weight of each rule is 1, the weight of each word in this grammar corresponds to the number of distinct trees that yield to that word, i.e., its ambiguity. ◀

Finally, we introduce *equivalence relations* and other related notions.

2.5 Equivalence Relations

Given a nonempty set X, an *equivalence relation* $\sim$ on X is a binary relation that is (*i*) reflexive: $x \sim x$; (*ii*) symmetric: $x \sim y \Rightarrow y \sim x$; and (*iii*) transitive: $x \sim y$ and $y \sim z \Rightarrow x \sim z$, for all $x, y, z \in X$.

Every equivalence relation $\sim$ induces a *partition* $P_\sim$ of X, i.e., a family $P_\sim \stackrel{\text{def}}{=} \{B_i\}_{i \in \mathcal{I}} \subseteq \wp(X)$ of subsets of X, with $\mathcal{I} \subseteq \mathbb{N}$, such that: (*i*) $B_i \neq \varnothing$ for all $i \in \mathcal{I}$; (*ii*) $B_i \cap B_j = \varnothing$, for all $i, j \in \mathcal{I}$ with $i \neq j$; and (*iii*) $X = \bigcup_{i \in \mathcal{I}} B_i$.

We say that a partition is *finite* when $\mathcal{I}$ is finite. Then, if $P_\sim$ is a finite partition, $\sim$ is an equivalence relation of *finite index*, i.e., $\sim$ describes a finite number of equivalence classes. We will sometimes refer to equivalences of finite index as *finite* equivalences. Given two equivalence relations $\sim_1$, $\sim_2$, we say that $\sim_1$ is *finer or equal to* $\sim_2$ when $\sim_1 \subseteq \sim_2$. Sometimes, we also say that $\sim_2$ is *coarser or equal to* $\sim_1$.

Each B_i is called a *block* of the partition. Given $u \in X$, then $P_\sim(u)$ denotes the unique block that contains u and corresponds to the *equivalence class* u w.r.t. $\sim$, $P_\sim(u) \stackrel{\text{def}}{=} \{v \in X \mid u \sim v\}$. This definition is naturally

extended to sets as follows. Given $S \subseteq X$, define $P_\sim(S) \stackrel{\text{def}}{=} \bigcup_{u \in S} P_\sim(u)$. We say that the congruence $\sim$ *represents precisely* S iff $P_\sim(S) = S$.

Finally, define $\text{Part}(X) \subseteq \wp(X)$ as the set of partitions of X. We will use the standard refinement ordering $\preceq$ between partitions: let $P_1, P_2 \in \text{Part}(X)$, then P_1 is *finer or equal to* P_2, denoted by $P_1 \preceq P_2$, iff for every $B_1 \in P_1$, there exists $B_2 \in P_2$ such that $B_1 \subseteq B_2$. Define the *coarsest common refinement* between P_1 and P_2, denoted by $P_1 \curlywedge P_2$, as the coarsest partition $P \in \text{Part}(X)$ that is finer than both P_1 and P_2. Similarly, define the *finest common coarsening* between P_1 and P_2, denoted by $P_1 \curlyvee P_2$, as the finest partition P that is coarser than both P_1 and P_2. Recall that $(\text{Part}(X), \preceq, \curlyvee, \curlywedge)$ is a complete lattice where the top (coarsest) element is $\{X\}$ and the bottom (finest) element is $\{\{x\} \mid x \in X\}$.

In this dissertation, we will use equivalence relations over words, i.e., $X \stackrel{\text{def}}{=} \Sigma^*$. Specifically, we will define *congruences* of finite index and *Parikh equivalence*.

2.5.1 Congruences

Definition 2.5.1 (Right and left congruences)**.** An equivalence relation $\sim$ is a *right congruence* iff for all $u, v \in \Sigma^*$, we have that $u \sim v \Rightarrow ua \sim va$, for all $a \in \Sigma$. Similarly, an equivalence relation $\sim$ is a *left congruence* iff for all $u, v \in \Sigma^*$, we have that $u \sim v \Rightarrow au \sim av$, for all $a \in \Sigma$.

We will denote right congruences by $\sim_r$ and left congruences by $\sim_\ell$. In the following lemma we give a characterization of right and left congruences in terms of their induced partitions.

Lemma 2.5.2. *The following properties hold:*

1. $\sim^r$ *is a right congruence iff* $P_{\sim^r}(v)u \subseteq P_{\sim^r}(vu)$*, for all* $u, v \in \Sigma^*$*.*
2. $\sim^\ell$ *is a left congruence iff* $uP_{\sim^\ell}(v) \subseteq P_{\sim^\ell}(uv)$*, for all* $u, v \in \Sigma^*$*.*

Proof.

1. $\sim^r$ is a right congruence iff $P_{\sim^r}(v)u \subseteq P_{\sim^r}(vu)$, for all $u, v \in \Sigma^*$.
   To simplify the notation, we denote $P_{\sim^r}$, the partition induced by $\sim^r$, simply by P.
   ($\Rightarrow$). Let $x \in P(v)u$, i.e., $x = \tilde{v}u$ with $P(\tilde{v}) = P(v)$ (hence

$v \sim^r \tilde{v}$). Since $\sim^r$ is a right congruence and $v \sim^r \tilde{v}$ then $vu \sim^r \tilde{v}u$. Therefore, $x \in P(vu)$.
($\Leftarrow$). By hypothesis, for each $u, v \in \Sigma^*$ and $\tilde{v} \in P(v)$, $\tilde{v}u \in P(vu)$. Therefore, $v \sim^r \tilde{v} \Rightarrow \tilde{v}u \sim^r vu$.

2. $\sim^\ell$ is a left congruence iff $uP_{\sim^\ell}(v) \subseteq P_{\sim^\ell}(uv)$, for all $u, v \in \Sigma^*$. To simplify the notation, we denote $P_{\sim^\ell}$, the partition induced by $\sim^\ell$ simply by P.
($\Rightarrow$). Let $x \in uP(v)$, i.e., $x = u\tilde{v}$ with $P(\tilde{v}) = P(v)$ (hence $v \sim^\ell \tilde{v}$). Since $\sim^\ell$ is a left congruence and $v \sim^\ell \tilde{v}$ then $uv \sim^\ell u\tilde{v}$. Therefore, $x \in P(uv)$.
($\Leftarrow$). By hypothesis, for each $u, v \in \Sigma^*$ and $\tilde{v} \in P(v)$, $u\tilde{v} \in P(uv)$, for all $u \in \Sigma^*$. Therefore, $v \sim^\ell \tilde{v} \Rightarrow u\tilde{v} \sim^\ell uv$.

2.5.2 Parikh Equivalence

We first introduce the notion of *Parikh image* of a word. Broadly speaking, the Parikh image of a word ignores the ordering of the symbols in its sequence, capturing only the information given by the number of occurrence of each symbol.

Definition 2.5.3 (Parikh image of a word)**.** Let $\Sigma = \{a_1, \ldots, a_n\}$ be an alphabet of symbols. The *Parikh image of a word* w over Σ, denoted by $\lbag w \rbag$, is the vector $(x_1, \ldots, x_n) \in \mathbb{N}^n$ where x_i is the number of occurrences of the symbol a_i in w, for each $i \in \{1, \ldots, n\}$.

Definition 2.5.4 (Parikh image of a language)**.** Let $\Sigma = \{a_1, \ldots, a_n\}$ be an alphabet of symbols. The *Parikh image of a language* $L \subseteq \Sigma^*$, is defined as $\lbag L \rbag \stackrel{\text{def}}{=} \{\lbag w \rbag \mid w \in L\}$.

Example 2.5.5. Consider the alphabet $\Sigma = \{a, b\}$. The Parikh image of $w = aba$ is the vector $(2, 1)$, and the Parikh image of the language $\{a^n b^n \mid n \geq 1\}$ is the set $\{(n, n) \mid n \geq 1\}$. ◀

Now we are ready to define the notion of *Parikh equivalence*.

Definition 2.5.6 (Parikh equivalence of words)**.** Given two words $w_1, w_2 \in \Sigma^*$, w_1 is *Parikh-equivalent* to w_2 iff $\lbag w_1 \rbag = \lbag w_2 \rbag$.

Finally, we lift the definition of Parikh equivalence from words to languages in a natural way.

Definition 2.5.7 (Parikh equivalence of languages)**.** Given two languages $L_1, L_2 \subseteq \Sigma^*$, L_1 is *Parikh-equivalent* to L_2 iff $\lbag L_1 \rbag = \lbag L_2 \rbag$.

Example 2.5.8. Consider the alphabet $\Sigma = \{a, b\}$ again. The words aba and baa are Parikh-equivalent since $\lbag aba \rbag = \lbag baa \rbag = (2, 1)$, while aba and bab are not, since $\lbag bab \rbag = (1, 2)$. On the other hand, the language $L_1 = \{a^n b^n \mid n \geq 1\}$ and $L_2 = \{(ab)^n \mid n \geq 1\}$ are Parikh-equivalent languages, since $\lbag L_2 \rbag = \{(n, n) \mid n \geq 1\} = L_1$. ◀

When we relax language-equivalence by Parikh-equivalence it turns out that the class of context-free languages and the class of regular languages are equivalent, in the sense that, every context-free language has the same Parikh image as some regular language. This is a consequence of a classical theorem in the Theory of Formal Languages proved in 1966 by Rohit Parikh [76].[4]

Theorem 2.5.9 (Parikh's Theorem [76])**.** *Every context-free language is* Parikh-equivalent *to some regular language.*

Note that the context-free language $\{a^n b^n \mid n \geq 1\}$ is Parikh-equivalent to the *regular* language $\{(ab)^n \mid n \geq 1\}$.

[4] More precisely, Parikh proved that the Parikh image of every context-free language is *semilinear*. Since the Parikh image of every regular language is semilinear, the formulation we give here is equivalent to the original result.

3 Finite Automata Constructions Based on Congruences

In this chapter, we propose a framework of finite-state constructions based on congruences of finite index over words to provide new insights on the connection between well-known methods for constructing the minimal deterministic automaton of a language.

3.1 Introduction

The problem of building the deterministic finite-state automaton with the least possible number of states recognizing a given regular language is a classical issue that arises in many different areas in computer science such as program verification, regular expression searching and natural language processing.

There exists a number of methods, such as Hopcroft's [51] and Moore's algorithms [73], that receive as input a deterministic finite-state automaton generating a language and build the minimal DFA for that language. In general, these methods rely on computing a partition of the set of states of the input DFA which is then used as the set of states of the minimal DFA.

On the other hand, Brzozowski [16] proposed the double-reversal method for building the minimal DFA for the language generated by an input non-deterministic automaton. This algorithm alternates a reverse operation and a determinization operation twice, relying on the fact that, for any given NFA $\mathcal{N}$, if the reverse automaton of $\mathcal{N}$ is deterministic then the determinization operation yields the minimal DFA for the language of $\mathcal{N}$.

This method has been recently generalized by Brzozowski and Tamm [18].

Despite of the fact that all these approaches aim to compute Nerode's equivalence relation for the given language, it is evident that the double-reversal method and its later generalization stand isolated from the state-partition-based methods such as Hopcroft's and Moore's algorithm. This has lead to different approaches to study the double-reversal method [25, 18, 87, 12] and its connection with other minimization algorithms [20, 9, 1, 43].

In this chapter, we propose the use of left and right congruences over Σ^* to draw a connection between these independent techniques to build the minimal DFA. We will require these congruences are of finite index, i.e., they define finite partitions on Σ^*. Given a right (resp. left) congruence satisfying the latter conditions and that represents precisely the language L, i.e., L is a union of equivalence classes, we will adapt the well-known minimal DFA construction based on right Nerode's congruence to obtain a DFA (resp. co-DFA) recognizing the language L [53, 19, 59].

Then, we will instantiate two particular right and left congruences. First, we will define so-called *right language-based congruences* whose definition relies on a regular language. This congruence is also known as the *right Nerode's congruence*. It is well-known [59] that applying our automata constructions to the right language-based congruence w.r.t. a regular language L yields to an automaton that is isomorphic to the minimal DFA for L. Second, we will define the right *automata-based congruences*, whose definition is in terms of a finite representation of the language, i.e., an NFA. Given an NFA $\mathcal{N}$, our automata construction applied to the automata-based congruence w.r.t. $\mathcal{N}$ yields to a determinized version of $\mathcal{N}$, in particular, it corresponds to the subset automata construction for $\mathcal{N}$. We will also give counterpart automata constructions for the left language-based and the left automata-based congruences, that will yield to the minimal co-DFA for the input language and a co-DFA for the input NFA, respectively.

We will show that there exists a left-right duality between our left and right congruences through the reverse operation on words. This left-right duality allows us to interpret the double-reversal method in simple terms (see Figure 3.1). It is worth to remark that this notion of duality has been observed before by Courcelle et al. [25] through their geometrical perspective on the determinization and minimization operations of finite-state automata, and similarly by Bonchi et al. [12] in the context of algebras and coalgebras. Now we exploit this duality to reformulate a

sufficient and necessary condition that guarantees that determinizing and automaton yields the minimal DFA [18], in terms of language abstractions given by congruences. This formulation allows us to relate the double-reversal method and Moore's algorithm.

3.1.1 Notation

Let us introduce some specific notation to this chapter. Given a language $L \subseteq \Sigma^*$ and a word $u \in \Sigma^*$, the *left quotient* of L by u is defined as the language $u^{-1}L \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid uw \in L\}$. We sometimes refer to the left quotient of L simply as *quotient* of L. Similarly, the *right quotient* of L by u is defined as the language $Lu^{-1} \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid wu \in L\}$.

Given an NFA $\mathcal{N}$, we say that a DFA for the language $\mathcal{L}(\mathcal{N})$ is the *minimal*, denoted by $\mathcal{N}^{DM}$ iff it has no unreachable states and no two states have the same right language. The minimal DFA for a regular language is unique modulo isomorphism.

Finally, recall that $\mathcal{N}^D$ and $\mathcal{N}^R$ denote the subset automata construction (see Definition 2.3.3) and the reverse automata construction (Definition 2.3.4) applied to the NFA $\mathcal{N}$, respectively.

3.2 Automata Constructions From Congruences

Given a finite right congruence $\sim^r$ and a regular language $L \subseteq \Sigma^*$ such that $\sim^r$ represents precisely L, i.e., $P_{\sim^r}(L) = L$, the following automata construction recognizes exactly the language L [19].

Definition 3.2.1 (Automata construction $\mathsf{H}^r(\sim^r, L)$)**.** Let $\sim^r$ be a right congruence and let $P_{\sim^r}$ be the partition induced by $\sim^r$. Let $L \subseteq \Sigma^*$ be a language. Define the automaton $\mathsf{H}^r(\sim^r, L) = (Q, \Sigma, \delta, I, F)$ where $Q = \{P_{\sim^r}(u) \mid u \in \Sigma^*\}$, $I = \{P_{\sim^r}(\varepsilon)\}$, $F = \{P_{\sim^r}(u) \mid u \in L\}$, and $\delta(P_{\sim^r}(u), a) = P_{\sim^r}(v)$ iff $P_{\sim^r}(u)a \subseteq P_{\sim^r}(v)$, for all $u, v \in \Sigma^*$ and $a \in \Sigma$.

Lemma 3.2.2. *Let $\sim^r$ be a right congruence and let $L \subseteq \Sigma^*$ be a language such that $P_{\sim^r}(L) = L$. Then $\mathcal{L}(\mathsf{H}^r(\sim^r, L)) = L$.*

> ***Proof.*** To simplify the notation, we denote $P_{\sim^r}$, the partition induced by $\sim^r$, simply by P. Let $\mathsf{H} = \mathsf{H}^r(\sim^r, L) = (Q, \Sigma, \delta, I, F)$. First, we prove that
>
> $$W^{\mathsf{H}}_{I,P(u)} = P(u), \quad \text{for each } u \in \Sigma^* \ . \tag{3.1}$$

($\subseteq$). We show that, for all $w \in \Sigma^*$, $w \in W^{\mathsf{H}}_{I,P(u)} \Rightarrow w \in P(u)$. The proof goes by induction on length of w.

- *Base case:* Let $w = \varepsilon$ and $\varepsilon \in W^{\mathsf{H}}_{I,P(u)}$. Note that the only initial state of H is $P(\varepsilon)$. Then, $P(u) = \delta(P(\varepsilon), \varepsilon)$, and thus $P(u) = P(\varepsilon)$. Hence, $\varepsilon \in P(u)$.

  Let $w = a$ with $a \in \Sigma$ and $a \in W^{\mathsf{H}}_{I,P(u)}$. Then, $P(u) = \delta(P(\varepsilon), a)$. By Definition 3.2.1, $P(\varepsilon)a \subseteq P(u)$. Therefore, $a \in P(u)$.

- *Inductive step:* Now we assume by hypothesis of induction that, if $|w| = n$ $(n > 1)$ then $w \in W^{\mathsf{H}}_{I,P(u)} \Rightarrow w \in P(u)$. Let $|w| = n + 1$ and $w \in W^{\mathsf{H}}_{I,P(u)}$. Assume w.l.o.g. that $w = xa$ with $x \in \Sigma^*$ and $a \in \Sigma$. Then, there exists a state $q \in Q$ such that $x \in W^{\mathsf{H}}_{I,q}$ and $P(u) = \delta(q, a)$. Since x satisfies the induction hypothesis, we have that $x \in q$, i.e., q denotes the state $P(x)$. On the other hand, by Definition 3.2.1, we have that $P(x)a \subseteq P(u)$. Therefore, $xa \in P(u)$.

($\supseteq$). We show that, for all $w \in \Sigma^*$, $w \in P(u) \Rightarrow w \in W^{\mathsf{H}}_{I,P(u)}$. Again, the proof goes by induction on length of w.

- *Base case:* Let $w = \varepsilon$ and $\varepsilon \in P(u)$. Then, $P(u) = P(\varepsilon)$. By Definition 3.2.1, $P(\varepsilon)$ is the initial state of H. Then, $\varepsilon \in W^{\mathsf{H}}_{I,P(\varepsilon)}$.

  Let $w = a$ with $a \in \Sigma$ and $a \in P(u)$. Then $P(u) = P(a)$. Since P is a partition induced by a right congruence, by Lemma 2.5.2, we have that $P(\varepsilon)a \subseteq P(a)$. Therefore, by Definition 3.2.1, $P(a) = \delta(P(\varepsilon), a)$. Since $P(\varepsilon)$ is the initial state of H, we have that $a \in W^{\mathsf{H}}_{I,P(a)}$, i.e., $w \in W^{\mathsf{H}}_{I,P(u)}$.

- *Inductive step:* Now we assume by hypothesis of induction that, if $|w| = n$ $(n > 1)$ then $w \in P(u) \Rightarrow w \in W^{\mathsf{H}}_{I,P(u)}$. Let $|w| = n + 1$ and $w \in P(u)$. Assume w.l.o.g. that $w = xa$ with $x \in \Sigma^*$ and $a \in \Sigma$. Then $P(xa) = P(u)$. Since P is a partition induced by a right congruence, by Lemma 2.5.2, we have that $P(x)a \subseteq P(xa)$. Since $x \in P(x)$, by induction hypothesis, $x \in W^{\mathsf{H}}_{I,P(x)}$. On the other hand, by Definition 3.2.1, $P(xa) = \delta(P(x), a)$. Hence $xa \in W^{\mathsf{H}}_{I,P(xa)}$, i.e., $w \in W^{\mathsf{H}}_{I,P(u)}$.

We conclude this proof by showing that $\mathcal{L}(\mathsf{H}) = L$.

$$\begin{aligned}
\mathcal{L}(\mathsf{H}) &= && \text{[By definition of } \mathcal{L}(\mathsf{H})\text{]} \\
\bigcup_{q \in F} W^{\mathsf{H}}_{I,q} &= && \text{[By Definition 3.2.1]} \\
\bigcup_{\substack{P(w) \in Q \\ w \in L}} W^{\mathsf{H}}_{I,P(w)} &= && \text{[By Equation (3.1)]} \\
\bigcup_{w \in L} P(w) &= && \text{[By hypothesis, } P(L) = L\text{]} \\
L \ . & &&
\end{aligned}$$

Now we give an automata construction for the language L that uses a finite left congruence $\sim^\ell$ such that $P_{\sim^\ell}(L) = L$.

Definition 3.2.3 (Automata construction $\mathsf{H}^\ell(\sim^\ell, L)$)**.** Let $\sim^\ell$ be a left congruence and let $P_{\sim^\ell}$ be the partition induced by $\sim^\ell$. Let $L \subseteq \Sigma^*$ be a language. Define the automaton $\mathsf{H}^\ell(\sim^\ell, L) = (Q, \Sigma, \delta, I, F)$ where $Q = \{P_{\sim^\ell}(u) \mid u \in \Sigma^*\}$, $I = \{P_{\sim^\ell}(u) \mid u \in L\}$, $F = \{P_{\sim^\ell}(\varepsilon)\}$, and $P_{\sim^\ell}(v) \in \delta(P_{\sim^\ell}(u), a)$ iff $aP_{\sim^\ell}(v) \subseteq P_{\sim^\ell}(u)$, for all $u, v \in \Sigma^*$ and $a \in \Sigma$.

Remark 3.2.4. Note that both $\mathsf{H}^r(\sim^r, L)$ and $\mathsf{H}^\ell(\sim^\ell, L)$ are *finite* since we assume $\sim^r$ and $\sim^\ell$ are of finite index. $\mathsf{H}^r(\sim^r, L)$ is also a complete *deterministic* finite-state automaton since, for each $u \in \Sigma^*$ and $a \in \Sigma$, there exists *exactly one* block $P_{\sim^r}(v)$ such that $P_{\sim^r}(u)a \subseteq P_{\sim^r}(v)$, which is $P_{\sim^r}(ua)$. Furthermore, observe that it possibly contains empty states but no state is unreachable.

On the other hand, $\mathsf{H}^\ell(\sim^\ell, L)$ is a co-complete *co-deterministic* finite-state automaton since, for each $v \in \Sigma^*$ and $a \in \Sigma$, there exists *exactly one* block $P_{\sim^\ell}(u)$ such that $aP_{\sim^\ell}(v) \subseteq P_{\sim^\ell}(u)$, which is $P_{\sim^\ell}(av)$. Finally, it possibly contains unreachable states but no state is empty.

Lemma 3.2.5. *Let $\sim^\ell$ be a left congruence and let $L \subseteq \Sigma^*$ be a language such that $P_{\sim^\ell}(L) = L$. Then $\mathcal{L}(\mathsf{H}^\ell(\sim^\ell, L)) = L$.*

Proof. To simplify the notation, we denote $P_{\sim^\ell}$, the partition induced by $\sim^\ell$, simply by P. Let $\mathsf{H} = \mathsf{H}^\ell(\sim^\ell, L) = (Q, \Sigma, \delta, I, F)$. First, we

prove that

$$W^{\mathsf{H}}_{P(u),F} = P(u), \quad \text{for each } u \in \Sigma^* \ . \tag{3.2}$$

($\subseteq$). We show that, for all $w \in \Sigma^*$, $w \in W^{\mathsf{H}}_{P(u),F} \Rightarrow w \in P(u)$. The proof goes by induction on length of w.

- *Base case:* Let $w = \varepsilon$ and $\varepsilon \in W^{\mathsf{H}}_{P(u),F}$. Note that the only final state of H is $P(\varepsilon)$. Then, $P(\varepsilon) \in \delta(P(u), \varepsilon)$, and thus $P(u) = P(\varepsilon)$. Hence, $\varepsilon \in P(u)$.

  Let $w = a$ with $a \in \Sigma$ and $a \in W^{\mathsf{H}}_{P(u),F}$. Then, $P(\varepsilon) \in \delta(P(u), a)$. By Definition 3.2.3, $aP(\varepsilon) \subseteq P(u)$. Therefore, $a \in P(u)$.

- *Inductive step:* Now we assume by hypothesis of induction that, if $|w| = n$ $(n > 1)$ then $w \in W^{\mathsf{H}}_{P(u),F} \Rightarrow w \in P(u)$. Let $|w| = n + 1$ and $w \in W^{\mathsf{H}}_{P(u),F}$. Assume w.l.o.g. that $w = ax$ with $a \in \Sigma$ and $x \in \Sigma^*$. Then, there exists a state $q \in Q$ such that $x \in W^{\mathsf{H}}_{q,F}$ and $q \in \delta(P(u), a)$. Since x satisfies the induction hypothesis, we have that $x \in q$, i.e., q denotes the state $P(x)$. On the other hand, by Definition 3.2.3, we have that $aP(x) \subseteq P(u)$. Therefore, $ax \in P(u)$.

($\supseteq$). We show that, for all $w \in \Sigma^*$, $w \in P(u) \Rightarrow w \in W^{\mathsf{H}}_{P(u),F}$. Again, the proof goes by induction on length of w.

- *Base case:* Let $w = \varepsilon$ and $\varepsilon \in P(u)$. Then, $P(u) = P(\varepsilon)$. By Definition 3.2.1, $P(\varepsilon)$ is the final state of H. Then, $\varepsilon \in W^{\mathsf{H}}_{P(u),F}$.

  Let $w = a$ with $a \in \Sigma$ and $a \in P(u)$. Then $P(u) = P(a)$. Since P is a partition induced by a left congruence, by Lemma 2.5.2, we have that $aP(\varepsilon) \subseteq P(a)$. Therefore, by Definition 3.2.3, $P(\varepsilon) \in \delta(P(a), a)$. Since $P(\varepsilon)$ is the final state of H, we have that $a \in W^{\mathsf{H}}_{P(a),F}$, i.e., $w \in W^{\mathsf{H}}_{P(u),F}$.

- *Inductive step:* Now we assume by hypothesis of induction that, if $|w| = n$ $(n > 1)$ then $w \in P(u) \Rightarrow w \in W^{\mathsf{H}}_{P(u),F}$. Let $|w| = n + 1$ and $w \in P(u)$. Assume w.l.o.g. that $w = ax$ with $a \in \Sigma$ and $x \in \Sigma^*$. Then $P(ax) = P(u)$. Since P is a partition induced by a left congruence, by Lemma 2.5.2, we have that

$aP(x) \subseteq P(ax)$. Since $x \in P(x)$, by induction hypothesis, $x \in W^{\mathsf{H}}_{P(x),F}$. On the other hand, by Definition 3.2.3, $P(x) \in \delta(P(ax), a)$. Hence $ax \in W^{\mathsf{H}}_{P(ax),F}$, i.e., $w \in W^{\mathsf{H}}_{P(u),F}$.

We conclude this proof by showing that $\mathcal{L}(\mathsf{H}) = L$.

$$
\begin{aligned}
\mathcal{L}(\mathsf{H}) = & \quad \text{[By definition of } \mathcal{L}(\mathsf{H})\text{]} \\
\bigcup_{q \in I} W^{\mathsf{H}}_{q,F} = & \quad \text{[By Definition 3.2.1]} \\
\bigcup_{\substack{P(w) \in Q \\ w \in L}} W^{\mathsf{H}}_{P(w),F} = & \quad \text{[By Equation (3.2)]} \\
\bigcup_{w \in L} P(w) = & \quad \text{[By hypothesis, } P(L) = L\text{]} \\
L \ . &
\end{aligned}
$$

The following lemma shows that, when $\sim^\ell$ and $\sim^r$ satisfy the notion of duality given by Equation 3.3, then the automata constructions H^ℓ and H^r are related through the reverse operation.

Lemma 3.2.6. *Let $\sim^r$ and $\sim^\ell$ be a right and left congruence respectively, and let $L \subseteq \Sigma^*$ be a language. If the following property holds*

$$u \sim^r v \Leftrightarrow u^R \sim^\ell v^R \tag{3.3}$$

then $\mathsf{H}^r(\sim^r, L)$ is isomorphic to $\left(\mathsf{H}^\ell(\sim^\ell, L^R)\right)^R$.

Proof. Let us define $\mathsf{H}^r(\sim^r, L) = (Q, \Sigma, \delta, I, F)$ and $(\mathsf{H}^\ell(\sim^\ell, L^R))^R = (\widetilde{Q}, \Sigma, \widetilde{\delta}, \widetilde{I}, \widetilde{F})$. We will show that $\mathsf{H}^r(\sim^r, L)$ and $(\mathsf{H}^\ell(\sim^\ell, L^R))^R$ are isomorphic.

Let $\varphi : Q \to \widetilde{Q}$ be a mapping assigning to each state $P_{\sim^r}(u) \in Q$ with $u \in \Sigma^*$, the state $P_{\sim^\ell}(u^R) \in \widetilde{Q}$. We show that φ is an NFA isomorphism between $\mathsf{H}^r(\sim^r, L)$ and $(\mathsf{H}^\ell(\sim^\ell, L^R))^R$.

The initial state $P_{\sim^r}(\varepsilon)$ of $\mathsf{H}^r(\sim^r, L)$ is mapped to $P_{\sim^\ell}(\varepsilon)$ which is the final state of $\mathsf{H}^\ell(\sim^\ell, L^R)$, i.e., the initial state of $(\mathsf{H}^\ell(\sim^\ell, L^R))^R$.

Each final state $P_{\sim^r}(u)$ of $\mathsf{H}^r(\sim^r, L)$ with $u \in L$ is mapped to $P_{\sim^\ell}(u^R)$, where $u^R \in L^R$. Therefore, $P_{\sim^\ell}(u^R)$ is an initial state of

$\mathsf{H}^\ell(\sim^\ell, L^R)$, i.e., a final state of $(\mathsf{H}^\ell(\sim^\ell, L^R))^R$.

Now, note that, by Definition 3.2.3, $\mathsf{H}^\ell(\sim^\ell, L^R)$ is a co-DFA, therefore $(\mathsf{H}^\ell(\sim^\ell, L^R))^R$ is a DFA. Let us show that $q' = \delta(q, a)$ if and only if $\varphi(q') = \widetilde{\delta}(\varphi(q), a)$, for all $q, q' \in Q$ and $a \in \Sigma$. Assume that $q = P_{\sim^r}(u)$ for some $u \in \Sigma^*$, and $q' = \delta(q, a)$ with $a \in \Sigma$. By Definition 3.2.1, we have that $q' = P_{\sim^r}(ua)$. Then, $\varphi(q) = P_{\sim^\ell}(u^R)$ and $\varphi(q') = P_{\sim^\ell}(au^R)$. Since $\sim^\ell$ is a left congruence, using Lemma 2.5.2 we have that $aP_{\sim^\ell}(u^R) \subseteq P_{\sim^\ell}(au^R)$. Then, there is a transition in $\mathsf{H}^\ell(\sim^\ell, L^R)$ from state $\varphi(q') = P_{\sim^\ell}(au^R)$ to state $\varphi(q) = P_{\sim^\ell}(u^R)$ reading a. Hence, there exists the reverse transition in $(\mathsf{H}^\ell(\sim^\ell, L^R))^R$, i.e., $\varphi(q') = \widetilde{\delta}(\varphi(q), a)$.

Assume now that $\widetilde{q} = P_{\sim^\ell}(u^R)$ for some $u \in \Sigma^*$, and $\widetilde{q'} = \widetilde{\delta}(\widetilde{q}, a)$ with $a \in \Sigma$. By Definition 3.2.3, we have that $\widetilde{q'} = P_{\sim^\ell}(au^R)$. Consider a state $q \in Q$ such that $\varphi(q) = \widetilde{q}$, then q is of the form $P_{\sim^r}(u)$. Likewise, consider a state $q' \in Q$ such that $\varphi(q') = \widetilde{q'}$, then q' is of the form $P_{\sim^r}(ua)$. Since $P_{\sim^r}$ is a partition induced by a right congruence, using Lemma 2.5.2, we have that $P_{\sim^r}(u)a \subseteq P_{\sim^r}(ua)$, and thus $q' = \delta(q, a)$.

3.3 Language-Based Congruences and Their Approximation Using NFAs

We instantiate two pairs of right and left congruences. First, we define the right *language-based* congruence, also known as *right Nerode's congruence* (e.g., see [59]). We also give its left counterpart.

Definition 3.3.1 (Language-based congruence)**.** Let $u, v \in \Sigma^*$ and let $L \subseteq \Sigma^*$ be a language. Define:

$$u \sim_L^r v \overset{\text{def}}{\Leftrightarrow} u^{-1}L = v^{-1}L \qquad \textit{Right} \text{ language-based congruence} \tag{3.4}$$

$$u \sim_L^\ell v \overset{\text{def}}{\Leftrightarrow} Lu^{-1} = Lv^{-1} \qquad \textit{Left} \text{ language-based congruence} \tag{3.5}$$

We give a proof of the fact that the equivalences defined above are indeed congruences in Section 3.8 (Lemma 3.8.1). Recall that, when L is a regular language, $\sim_L^r$ and $\sim_L^\ell$ are of *finite index* [19, 59]. Since we are interested in congruences of finite index (or equivalently, finite partitions), we will always assume that L is a regular language over Σ.

The following result states that, given a language L, the right Nerode's congruence induces the coarsest partition of Σ^* which is a right congruence and represents precisely L.

Lemma 3.3.2 (de Luca and Varricchio [27])**.** *Let $L \subseteq \Sigma^*$ be a regular language. Then,*

$$P_{\sim_L^r} = \bigwedge \{P_{\sim^r} \mid \sim^r \text{ is a right congruence and } P_{\sim^r}(L) = L\} \ .$$

In a similar way, one can prove that the same property holds for the left Nerode's congruence. Therefore, as we shall see, applying the construction H to these equivalences yields minimal automata. However, computing them is unpractical since languages are possibly infinite, even if they are regular. Thus, we will consider congruences based on the states of the NFA representation of the language, which induce finer partitions of Σ^* than Nerode's congruence. In this sense, we say that the automata-based congruences *approximate* Nerode's congruences.

Definition 3.3.3 (Automata-based congruences)**.** Let $u, v \in \Sigma^*$ and let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA. Define:

$$u \sim_{\mathcal{N}}^r v \overset{\text{def}}{\Leftrightarrow} \text{post}_u^{\mathcal{N}}(I) = \text{post}_v^{\mathcal{N}}(I) \qquad \textit{Right} \text{ automata-based congruence} \tag{3.6}$$

$$u \sim_{\mathcal{N}}^{\ell} v \overset{\text{def}}{\Leftrightarrow} \text{pre}_u^{\mathcal{N}}(F) = \text{pre}_v^{\mathcal{N}}(F) \qquad \textit{Left} \text{ automata-based congruence} \tag{3.7}$$

A proof of the fact that the equivalences defined above are indeed congruences can be found in Section 3.8 (Lemma 3.8.2). Furthermore, they are of *finite index* since each equivalence class corresponds to a subset of states of $\mathcal{N}$.

The following lemma shows the relation between the automata-based and the language-based congruences.

Lemma 3.3.4. *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an automaton with $L = \mathcal{L}(\mathcal{N})$. Then, $\sim_{\mathcal{N}}^r \subseteq \sim_L^r$.*

Proof.

$$
\begin{aligned}
u \sim^r_{\mathcal{N}} v \Leftrightarrow & \quad \text{[By definition (3.6)]} \\
\text{post}^{\mathcal{N}}_u(I) = \text{post}^{\mathcal{N}}_v(I) \implies & \\
W^{\mathcal{N}}_{\text{post}^{\mathcal{N}}_u(I),F} = W^{\mathcal{N}}_{\text{post}^{\mathcal{N}}_v(I),F} \Leftrightarrow & \quad \text{[def. of quotient of } L\text{]} \\
u^{-1}L = v^{-1}L \Leftrightarrow & \quad \text{[By definition (3.7)]} \\
u \sim^r_L v \ . &
\end{aligned}
$$

In consequence, we give a sufficient and necessary condition for the right language-based and the automata-based congruences to coincide.

Corollary 3.3.5. *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an automaton with $L = \mathcal{L}(\mathcal{N})$. Then, the following conditions are equivalent:*

1. $\sim^r_L = \sim^r_{\mathcal{N}}$.
2. $\forall u, v \in \Sigma^*,\ W^{\mathcal{N}}_{\text{post}^{\mathcal{N}}_u(I),F} = W^{\mathcal{N}}_{\text{post}^{\mathcal{N}}_v(I),F} \Leftrightarrow \text{post}^{\mathcal{N}}_u(I) = \text{post}^{\mathcal{N}}_v(I)$.

3.3.1 Automata Constructions

In what follows, we will use Min and Det to denote the construction H when applied, respectively, to the language-based congruences induced by a regular language and the automata-based congruences induced by an NFA.

Definition 3.3.6. Let $\mathcal{N}$ be an NFA generating the language $L = \mathcal{L}(\mathcal{N})$. Define:

$$
\begin{aligned}
\mathsf{Min}^r(L) &\stackrel{\text{def}}{=} \mathsf{H}^r(\sim^r_L, L) & \qquad \mathsf{Det}^r(\mathcal{N}) &\stackrel{\text{def}}{=} \mathsf{H}^r(\sim^r_{\mathcal{N}}, L) \\
\mathsf{Min}^\ell(L) &\stackrel{\text{def}}{=} \mathsf{H}^\ell(\sim^\ell_L, L) & \qquad \mathsf{Det}^\ell(\mathcal{N}) &\stackrel{\text{def}}{=} \mathsf{H}^\ell(\sim^\ell_{\mathcal{N}}, L) \ .
\end{aligned}
$$

Given an NFA $\mathcal{N}$ generating the language $L = \mathcal{L}(\mathcal{N})$, all constructions in the above definition yield automata generating L. However, while the constructions using the right congruences result in DFAs, the constructions relying on left congruences result in co-DFAs. Furthermore, since the pairs of congruences (3.4)-(3.5) and (3.6)-(3.7), from Definition 3.3.1 and 3.3.3 respectively, are dual, i.e., they satisfy the hypothesis of Lemma 3.2.6, it follows that $\mathsf{Min}^\ell(L)$ is isomorphic to $(\mathsf{Min}^r(L^R))^R$ and $\mathsf{Det}^\ell(\mathcal{N})$ is isomorphic to $(\mathsf{Det}^r(\mathcal{N}^R))^R$.

On the other hand, since Min^r relies on the language-based congruences, the resulting DFA is minimal, which is not guaranteed to occur with Det^r. This easily follows from the fact that the states of the automata constructions are the equivalence classes of the given congruences and there is no right congruence, representing L precisely, that is coarser than the right Nerode's congruence (see Lemma 3.3.2).

Finally, since every co-deterministic automaton (with no empty states) satisfies the second condition of Corollary 3.3.5, it follows that determinizing (Det^r) a co-deterministic automaton ($\mathsf{Det}^\ell(\mathcal{N})$) results in the minimal DFA ($\mathsf{Min}^r(\mathcal{L}(\mathcal{N}))$), as already proven by Sakarovitch [81, Proposition 3.13].

We formalize all these notions in Theorem 3.3.7. Finally, Figure 3.1 summarizes all these well-known connections between the automata constructions given in Definition 3.3.6.

Theorem 3.3.7. *Let $\mathcal{N}$ be an NFA generating language $L = \mathcal{L}(\mathcal{N})$. Then the following properties hold:*

1. $\mathcal{L}(\mathsf{Min}^r(L)) = \mathcal{L}(\mathsf{Min}^\ell(L)) = L = \mathcal{L}(\mathsf{Det}^r(\mathcal{N})) = \mathcal{L}(\mathsf{Det}^\ell(\mathcal{N}))$.
2. $\mathsf{Min}^r(L)$ *is isomorphic to the minimal deterministic automaton for* L.
3. $\mathsf{Det}^r(\mathcal{N})$ *is isomorphic to* $\mathcal{N}^D$.
4. $\mathsf{Min}^\ell(L)$ *is isomorphic to* $(\mathsf{Min}^r(L^R))^R$.
5. $\mathsf{Det}^\ell(\mathcal{N})$ *is isomorphic to* $(\mathsf{Det}^r(\mathcal{N}^R))^R$.
6. $\mathsf{Det}^r(\mathsf{Det}^\ell(\mathcal{N}))$ *is isomorphic to* $\mathsf{Min}^r(L)$.

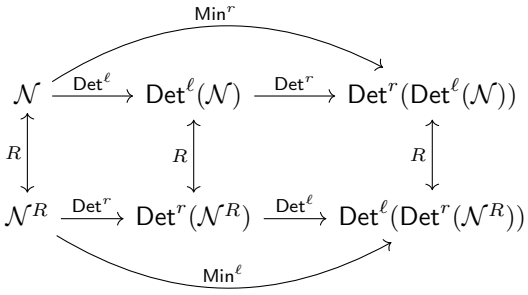


Figure 3.1: Relations between the constructions Det^ℓ, Det^r, Min^ℓ and Min^r.

Note that constructions Min^r and Min^ℓ in the diagram on the previous page are applied to the language generated by the automaton in the origin of the labeled arrow, while constructions Det^r and Det^ℓ are applied directly to the automaton. The upper part of the diagram follows from Theorem 3.3.7 (6). Both squares follow from Theorem 3.3.7 (5), which states that $\mathsf{Det}^\ell(\mathcal{N})$ is isomorphic to $(\mathsf{Det}^r(\mathcal{N}^R))^R$. Finally, the bottom curved arc follows from Theorem 3.3.7 (4). Incidentally, the diagram shows a new relation which follows from the left-right dualities between $\sim_L^\ell$ and $\sim_L^r$, and $\sim_{\mathcal{N}}^\ell$ and $\sim_{\mathcal{N}}^r$: $\mathsf{Min}^\ell(\mathcal{L}(\mathcal{N}^R))$ is isomorphic to $\mathsf{Det}^\ell(\mathsf{Det}^r(\mathcal{N}^R))$.

Let us give a proof of this theorem.

Proof.

1. $\mathcal{L}(\mathsf{Min}^r(L)) = \mathcal{L}(\mathsf{Min}^\ell(L)) = L = \mathcal{L}(\mathsf{Det}^r(\mathcal{N})) = \mathcal{L}(\mathsf{Det}^\ell(\mathcal{N}))$.

   By Definition 3.3.6, $\mathsf{Min}^r(L) = \mathsf{H}^r(\sim_L^r, L)$ and $\mathsf{Det}^r(\mathcal{N}) = \mathsf{H}^r(\sim_{\mathcal{N}}^r, L)$. By Lemma 3.2.2, $\mathcal{L}(\mathsf{H}^r(\sim_L^r, L)) = L = \mathcal{L}(\mathsf{H}^r(\sim_{\mathcal{N}}^r, L))$. Thus, $\mathcal{L}(\mathsf{Min}^r(L)) = \mathsf{Det}^r(\mathcal{N}) = L$. The proof of $\mathcal{L}(\mathsf{Min}^\ell(L)) = L = \mathcal{L}(\mathsf{Det}^\ell(\mathcal{N}))$ goes similarly using Lemma 3.2.5.

2. $\mathsf{Min}^r(L)$ is isomorphic to the minimal deterministic automaton for L. Let P be the partition induced by $\sim_L^r$. Recall that the automaton $\mathsf{Min}^r(L) = (Q, \Sigma, \delta, I, F)$ is a complete DFA (see Remark 3.2.4). Recall also that the *quotient DFA* of L, defined as $\mathcal{D} = (\widetilde{Q}, \Sigma, \eta, \widetilde{q}_0, \widetilde{F})$ where $\widetilde{Q} = \{u^{-1}L \mid u \in \Sigma^*\}$, $\eta(u^{-1}L, a) = a^{-1}(u^{-1}L)$ for each $a \in \Sigma$, $\widetilde{q}_0 = \varepsilon^{-1}L = L$ and $\widetilde{F} = \{u^{-1}L \mid \varepsilon \in u^{-1}L\}$, is the minimal DFA for L. We will show that $\mathsf{Min}^r(L)$ is isomorphic to $\mathcal{D}$.

   Let $\varphi : \widetilde{Q} \to Q$ be the mapping assigning to each state $\widetilde{q}_i \in \widetilde{Q}$ of the form $u^{-1}L$, the state $P(u) \in Q$, with $u \in \Sigma^*$. Note that, in particular, if $\widetilde{q}_i \in \widetilde{Q}$ is the empty set, then φ maps $\widetilde{q}_i$ to the block in P that contains all the words that are not prefixes of L. We show that φ is a DFA isomorphism between $\mathcal{D}$ and $\mathsf{Min}^r(L)$.

   The initial state $\widetilde{q}_0 = \varepsilon^{-1}L$ of $\mathcal{D}$ is mapped to the state $P(\varepsilon)$ which, by definition, is the unique initial state of $\mathsf{Min}^r(L)$. Each final state $u^{-1}L \in \widetilde{F}$ is mapped to the state $P(u)$ with $u \in L$ which, by definition, is a final state of $\mathsf{Min}^r(L)$.

   We now show that $\widetilde{q}_j = \eta(\widetilde{q}_i, a)$ if and only if $\varphi(\widetilde{q}_j) = \delta(\varphi(\widetilde{q}_i), a)$,

for all $\widetilde{q}_i, \widetilde{q}_j \in \widetilde{Q}, a \in \Sigma$. Assume that $\widetilde{q}_i = u^{-1}L$ for some $u \in \Sigma^*$ and $\widetilde{q}_j = \eta(\widetilde{q}_i, a)$ where $\widetilde{q}_j = a^{-1}(u^{-1}L)$ and $a \in \Sigma$. Note that $a^{-1}(u^{-1}L) = \{x \in \Sigma^* \mid uax \in L\}$. Then, $\varphi(\widetilde{q}_i) = P(u)$ and $\varphi(\widetilde{q}_j) = P(ua)$. Since P is a partition induced by a right congruence, using Lemma 2.5.2, we have that $P(u)a \subseteq P(ua)$. Therefore, $\varphi(\widetilde{q}_j) = \delta(\varphi(\widetilde{q}_i), a)$.

Assume now that $P(ua) = \delta(P(u), a)$ for some $u \in \Sigma^*$ and $a \in \Sigma$. Consider $\widetilde{q}_i \in \widetilde{Q}$ such that $\varphi(\widetilde{q}_i) = P(u)$, then $\widetilde{q}_i = u^{-1}L$. Likewise, consider $\widetilde{q}_j \in \widetilde{Q}$ such that $\varphi(\widetilde{q}_j) = P(ua)$, then $\widetilde{q}_j = (ua)^{-1}L = a^{-1}(u^{-1})L$. Therefore, $\widetilde{q}_j = \eta(\widetilde{q}_i, a)$.

3. $\mathsf{Det}^r(\mathcal{N})$ is isomorphic to $\mathcal{N}^D$. Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$. Recall that $\mathcal{N}^D$ denotes the DFA that results from applying the subset construction to $\mathcal{N}$ and removing all states that are not reachable. Thus, $\mathcal{N}^D$ possibly contains empty states but no state is unreachable. Let $\mathcal{N}^D = (Q_d, \Sigma, \delta_d, \{I\}, F_d)$ and let $\mathsf{Det}^r(\mathcal{N}) = (\widetilde{Q}, \Sigma, \widetilde{\delta}, \widetilde{I}, \widetilde{F})$. Let P be the partition induced by $\sim^r_{\mathcal{N}}$ and let $\varphi : \widetilde{Q} \to Q_d$ be the mapping assigning to each state $P(u) \in \widetilde{Q}$, the set $\mathrm{post}^{\mathcal{N}}_u(I) \in Q_d$ with $u \in \Sigma^*$. Note that if $u \in \Sigma^*$ is not a prefix of $\mathcal{L}(\mathcal{N})$, then φ maps $P(u)$ to $\mathrm{post}^{\mathcal{N}}_u(I) = \varnothing$. We show that φ is a DFA isomorphism between $\mathsf{Det}^r(\mathcal{N})$ and $\mathcal{N}^D$.

The initial state of $\mathsf{Det}^r(\mathcal{N})$, $P(\varepsilon)$, is mapped to $\mathrm{post}^{\mathcal{N}}_\varepsilon(I) = \{I\}$. Therefore, φ maps the initial state of $\mathsf{Det}^r(\mathcal{N})$ to the initial state of $\mathcal{N}^D$. Each final state of $\mathsf{Det}^r(\mathcal{N})$, $P(u)$ with $u \in L$, is mapped to $\mathrm{post}^{\mathcal{N}}_u(I)$. Since $\mathrm{post}^{\mathcal{N}}_u(I) \cap F \neq \varnothing$, $\mathrm{post}^{\mathcal{N}}_u(I) \in \widetilde{F}$.

Now note that, by Remark 3.2.4, $\mathsf{Det}^r(\mathcal{N})$ is a complete DFA, and by construction, so is $\mathcal{N}^D$. Let us show that $\widetilde{q'} = \widetilde{\delta}(\widetilde{q}, a)$ iff $\varphi(\widetilde{q'}) = \delta_d(\varphi(\widetilde{q}), a)$, for all $\widetilde{q}, \widetilde{q'} \in \widetilde{Q}$ and $a \in \Sigma$. Assume that $\widetilde{q} = P(u)$, for some $u \in \Sigma^*$, and $\widetilde{q'} = \widetilde{\delta}(\widetilde{q}, a)$, with $a \in \Sigma$. By Definition 3.2.1, we have that $\widetilde{q'} = P(ua)$. Then, $\varphi(\widetilde{q}) = \mathrm{post}^{\mathcal{N}}_u(I)$ and $\varphi(\widetilde{q'}) = \mathrm{post}^{\mathcal{N}}_{ua}(I) = \mathrm{post}^{\mathcal{N}}_a(\mathrm{post}^{\mathcal{N}}_u(I))$. Therefore, $\varphi(\widetilde{q'}) = \delta_d(\varphi(\widetilde{q'}), a)$.

Assume now that $\delta_d(\mathrm{post}^{\mathcal{N}}_u(I), a) = \mathrm{post}^{\mathcal{N}}_{ua}(I)$. Consider $\widetilde{q} \in \widetilde{Q}$ such that $\varphi(\widetilde{q}) = \mathrm{post}^{\mathcal{N}}_u(I)$, then $\widetilde{q} = P(u)$. Likewise, consider $\widetilde{q'} \in \widetilde{Q}$ such that $\varphi(\widetilde{q'}) = \mathrm{post}^{\mathcal{N}}_{ua}(I)$, then $\widetilde{q'} = P(ua)$. Since P is a partition induced by a right congruence, using Lemma 2.5.2,

we have that $P(u)a \subseteq P(ua)$. Therefore, $\widetilde{q'} = \widetilde{\delta}(\widetilde{q}, a)$.

4. $\mathsf{Min}^{\ell}(L)$ is isomorphic to $(\mathsf{Min}^{r}(L^R))^R$Observe that, for each $u \in \Sigma^*$:

$$(u^{-1}L)^R = \{x^R \in \Sigma^* \mid ux \in L\} = \{x^R \in \Sigma^* \mid x^R u^R \in L^R\} = \{x' \in \Sigma^* \mid x' u^R \in L^R\} = L^R(u^R)^{-1} \ . \quad (3.8)$$

Therefore,

$$\begin{aligned} u \sim_L^{\ell} v \Leftrightarrow & \quad [\text{By Definition (3.5)}] \\ u^{-1}L = v^{-1}L \Leftrightarrow & \quad [x = y \Leftrightarrow x^R = y^R] \\ (u^{-1}L)^R = (v^{-1}L)^R \Leftrightarrow & \quad [\text{By Equation (3.8)}] \\ L^R(u^R)^{-1} = L^R(v^R)^{-1} \Leftrightarrow & \quad [\text{By Definition (3.4)}] \\ u^R \sim_{L^R}^{r} v^R \ . & \end{aligned}$$

Finally, it follows from Lemma 3.2.6 that $\mathsf{Min}^{\ell}(L)$ is isomorphic to $(\mathsf{Min}^{r}(L^R))^R$.

5. $\mathsf{Det}^{\ell}(\mathcal{N})$ is isomorphic to $(\mathsf{Det}^{r}(\mathcal{N}^R))^R$. For each $u, v \in \Sigma^*$:

$$\begin{aligned} u \sim_{\mathcal{N}^R}^{\ell} v \Leftrightarrow & \quad [\text{By Definition 3.3.3}] \\ \mathrm{pre}_u^{\mathcal{N}^R}(F) = \mathrm{pre}_v^{\mathcal{N}^R}(F) \Leftrightarrow & \quad [q \in \mathrm{pre}_x^{\mathcal{N}^R}(F) \text{ iff } q \in \mathrm{post}_{x^R}^{\mathcal{N}}(I)] \\ \mathrm{post}_{u^R}^{\mathcal{N}}(I) = \mathrm{post}_{v^R}^{\mathcal{N}}(I) \Leftrightarrow & \quad [\text{By Definition 3.3.3}] \\ u^R \sim_{\mathcal{N}}^{\ell} v^R \ . & \end{aligned}$$

It follows from Lemma 3.2.6 that $\mathsf{Det}^{\ell}(\mathcal{N})$ is isomorphic to $\mathsf{Det}^{r}(\mathcal{N}^R))^R$.

6. $\mathsf{Det}^{r}(\mathsf{Det}^{\ell}(\mathcal{N}))$ is isomorphic to $\mathsf{Min}^{r}(L)$.

   By Theorem 3.3.7 (1), $\mathsf{Det}^{\ell}(\mathcal{N})$ is a co-deterministic automaton generating the language $\mathcal{L}(\mathcal{N})$. Since $\mathsf{Det}^{\ell}(\mathcal{N})$ is a co-DFA with no empty states, it satisfies the second condition of Corollary 3.3.5. Therefore, $\mathsf{Det}^{r}(\mathsf{Det}^{\ell}(\mathcal{N}))$ is isomorphic to $\mathsf{Min}^{r}(\mathcal{L}(\mathsf{Det}^{\ell}(\mathcal{N}))) = \mathsf{Min}^{r}(\mathcal{L}(\mathcal{N}))$.

3.4 Congruences as Language Abstractions

In this section we use the right language-based congruence $\sim_L^r$ as a language abstraction to give a necessary and sufficient condition on an NFA so that determinizing it yields the minimal DFA. More precisely, this condition asks whether $\sim_L^r$ represents precisely the left languages of the states of the NFA. First, we give a preliminary result.

Lemma 3.4.1. *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA with $L = \mathcal{L}(\mathcal{N})$ and $\sim_L^r = \sim_{\mathcal{N}}^r$. Then $\forall q \in Q,\ P_{\sim_L^r}(W_{I,q}^{\mathcal{N}}) = W_{I,q}^{\mathcal{N}}$.*

Proof.

$$
\begin{aligned}
P_{\sim_L^r}(W_{I,q}^{\mathcal{N}}) = \quad & \text{[By definition]} \\
\{w \in \Sigma^* \mid \exists u \in W_{I,q}^{\mathcal{N}},\ w^{-1}L = u^{-1}L\} = \quad & [\sim_L^r = \sim_{\mathcal{N}}^r] \\
\{w \in \Sigma^* \mid \exists u \in W_{I,q}^{\mathcal{N}},\ \mathrm{post}_w^{\mathcal{N}}(I) = \mathrm{post}_u^{\mathcal{N}}(I)\} \ .
\end{aligned}
$$

Since, for all $u \in \Sigma^*$ and $q \in Q$: $u \in W_{I,q}^{\mathcal{N}} \Leftrightarrow q \in \mathrm{post}_u^{\mathcal{N}}(I)$, we have the following set inclusion:

$$
\begin{aligned}
\{w \in \Sigma^* \mid \exists u \in W_{I,q}^{\mathcal{N}},\ \mathrm{post}_w^{\mathcal{N}}(I) = \mathrm{post}_u^{\mathcal{N}}(I)\} \subseteq & \\
\{w \in \Sigma^* \mid q \in \mathrm{post}_w^{\mathcal{N}}(I)\} = \quad & \text{[By def. of } W_{I,q}^{\mathcal{N}}] \\
W_{I,q}^{\mathcal{N}} \ . &
\end{aligned}
$$

By reflexivity of $\sim_L^r$, we conclude that $P_{\sim_L^r}(W_{I,q}^{\mathcal{N}}) = W_{I,q}^{\mathcal{N}}$.

Theorem 3.4.2. *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA with $L = \mathcal{L}(\mathcal{N})$. Then $\mathsf{Det}^r(\mathcal{N})$ is the minimal DFA for L iff $\forall q \in Q,\ P_{\sim_L^r}(W_{I,q}^{\mathcal{N}}) = W_{I,q}^{\mathcal{N}}$.*

Proof. Assume $\mathsf{Det}^r(\mathcal{N})$ is minimal. Then $P_{\sim_{\mathcal{N}}^r}(u) = P_{\sim_L^r}(u)$ for all $u \in \Sigma^*$, i.e. $\sim_L^r = \sim_{\mathcal{N}}^r$. It follows from Lemma 3.4.1 that $P_{\sim_L^r}(W_{I,q}^{\mathcal{N}}) = W_{I,q}^{\mathcal{N}}$.

Now, assume that $P_{\sim_L^r}(W_{I,q}^{\mathcal{N}}) = W_{I,q}^{\mathcal{N}}$, for each $q \in Q$. Then, for

every $u \in \Sigma^*$,

$$
\begin{aligned}
P_{\sim^r_{\mathcal{N}}}(u) = & \quad \text{[By def. of } P_{\sim^r_{\mathcal{N}}}\text{]} \\
\{v \in \Sigma^* \mid \mathrm{post}^{\mathcal{N}}_u(I) = \mathrm{post}^{\mathcal{N}}_v(I)\} = & \quad \text{[By def. of set equality]} \\
\{v \in \Sigma^* \mid \forall q \in Q : q \in \mathrm{post}^{\mathcal{N}}_u(I) \Leftrightarrow q \in \mathrm{post}^{\mathcal{N}}_v(I)\} = & \quad [q \in \mathrm{post}^{\mathcal{N}}_v \Leftrightarrow v \in W^{\mathcal{N}}_{I,q}] \\
\{v \in \Sigma^* \mid \forall q \in Q : q \in \mathrm{post}^{\mathcal{N}}_u(I) \Leftrightarrow v \in W^{\mathcal{N}}_{I,q}\} = & \quad \text{[By def. of intersection]} \\
\bigcap_{q \in \mathrm{post}^{\mathcal{N}}_u(I)} W^{\mathcal{N}}_{I,q} \cap \bigcap_{q \notin \mathrm{post}^{\mathcal{N}}_u(I)} (W^{\mathcal{N}}_{I,q})^c = & \quad [P_{\sim^r_L}(W^{\mathcal{N}}_{I,q}) = W^{\mathcal{N}}_{I,q}] \\
\bigcap_{q \in \mathrm{post}^{\mathcal{N}}_u(I)} P_{\sim^r_L}(W^{\mathcal{N}}_{I,q}) \cap \bigcap_{q \notin \mathrm{post}^{\mathcal{N}}_u(I)} (P_{\sim^r_L}(W^{\mathcal{N}}_{I,q}))^c & \ .
\end{aligned}
$$

From the last equality we conclude that $P_{\sim^r_{\mathcal{N}}}(u)$ is a *union* of blocks of $P_{\sim^r_L}$. Recall that $\sim^r_L$ induces the coarsest right congruence such that $P_{\sim^r_L}(L) = L$ (Lemma 3.3.2). Since $\sim^r_{\mathcal{N}}$ is a right congruence satisfying $P_{\sim^r_{\mathcal{N}}}(L) = L$ then $P_{\sim^r_{\mathcal{N}}} \subseteq P_{\sim^r_L}$. Thus, $P_{\sim^r_{\mathcal{N}}}(u)$ necessarily corresponds to one single block of $P_{\sim^r_L}$, namely, $P_{\sim^r_L}(u)$. Since $P_{\sim^r_{\mathcal{N}}}(u) = P_{\sim^r_L}(u)$ for each $u \in \Sigma^*$, we conclude that $\mathsf{Det}^r(\mathcal{N}) = \mathsf{Min}^r(L)$.

3.5 A Congruence-Based Perspective on Known Algorithms

There exist several well-known independent techniques for the construction of minimal DFAs in the literature. Some of these methods are based on refining a state partition of an input DFA, such as Moore's algorithm [73], while others directly manipulate an input NFA, such as the double-reversal method [16]. Now, we establish a connection between these algorithms through Theorem 3.4.2.

We start with the double-reversal algorithm.

3.5.1 Double-Reversal Method

We give a simple proof of the well-known double-reversal minimization algorithm of Brzozowski [16] using Theorem 3.4.2. Note that, since $\mathsf{Det}^r(\mathcal{N})$ is isomorphic to $\mathcal{N}^D$ by Theorem 3.3.7 (3), the following result coincides with that of Brzozowski.

Theorem 3.5.1 (from [16]). *Let $\mathcal{N}$ be an NFA. Then $\mathsf{Det}^r((\mathsf{Det}^r(\mathcal{N}^R))^R)$ is isomorphic to the minimal DFA for $\mathcal{L}(\mathcal{N})$.*

> ***Proof.*** Let $L = \mathcal{L}(\mathcal{N})$. By definition, $\mathcal{N}' = (\mathsf{Det}^r(\mathcal{N}^R))^R$ is a co-DFA with no empty states, and therefore it satisfies the second condition of Corollary 3.3.5. Note that in, the latter condition, the right-to-left implication always holds since $\sim^r_{\mathcal{N}'} \subseteq \sim^r_{\mathcal{L}(\mathcal{N}')}$. Now, let us look at the left-to-right implication. Note that $\mathcal{N}'$ has no empty states, and thus if $u^{-1}L = v^{-1}L = \varnothing$ then $\mathrm{post}^{\mathcal{N}'}_u(I) = \mathrm{post}^{\mathcal{N}'}_v(I) = \varnothing$. On the other hand, assume that there exists $u, v \in \Sigma^*$ with $u^{-1}L = v^{-1}L \neq \varnothing$ and $\mathrm{post}^{\mathcal{N}'}_u(I) \neq \mathrm{post}^{\mathcal{N}'}_v(I)$. W.l.o.g. assume that there exist $q_u \in (\mathrm{post}^{\mathcal{N}'}_u(I) \cap (\mathrm{post}^{\mathcal{N}'}_v(I))^c)$ and $q_v \in (\mathrm{post}^{\mathcal{N}'}_v(I) \cap (\mathrm{post}^{\mathcal{N}'}_u(I))^c)$. Let q_f be the unique final state of $\mathcal{N}'$. Since $u^{-1}L = v^{-1}L \neq \varnothing$, there is $w \in u^{-1}L$ such that $q_u, q_v \in \mathrm{pre}_w(\{q_f\})$, which contradicts the fact that $\mathcal{N}'$ is a co-DFA. Therefore, $\sim^r_{\mathcal{L}(\mathcal{N}')} \subseteq \sim^r_{\mathcal{N}'}$.
>
> It follows that $\sim^r_{\mathcal{L}(\mathcal{N}')} = \sim^r_{\mathcal{L}(\mathcal{N}')}$ which, by Lemma 3.4.1 and Theorem 3.4.2, implies that $\mathsf{Det}^r(\mathcal{N}')$ is minimal.

Observe that Theorem 3.5.1 can be inferred from Figure 3.1 by following the path starting at $\mathcal{N}$, labeled with $R - \mathsf{Det}^r - R - \mathsf{Det}^r$ and ending in $\mathsf{Min}^r(\mathcal{L}(\mathcal{N}))$.

3.5.2 Simulation-Based Double-Reversal Method

Relying on the fact that determinizing a co-DFA (with no empty states) yields to the minimal DFA, the double-reversal method determinizes the automaton $(\mathsf{Det}^r(\mathcal{N}^R))^R$ to obtain the minimal DFA for $\mathcal{L}(\mathcal{N})$. Next theorem extends this method to any left congruence that precisely represents $\mathcal{L}(\mathcal{N})$.

Theorem 3.5.2. *Let $\mathcal{N}$ be an NFA with $L = \mathcal{L}(\mathcal{N})$, and let $\sim^\ell$ be a left congruence such that $P_{\sim^\ell}(L) = L$. Then $\mathsf{Det}^r(\mathsf{H}^\ell(\sim^\ell, \mathcal{L}(\mathcal{N})))$ is isomorphic to the minimal DFA for $\mathcal{L}(\mathcal{N})$.*

> ***Proof.*** By construction, $\mathsf{H}^\ell(\sim^\ell, L)$ is a co-DFA with no empty states for L. The remainder of the proof follows that of Theorem 3.5.1.

In view of the latter result, we will propose a congruence which is coarser than (or equal to) the automata-based congruence, and thus yields

to a double-reversal method that produces an intermediate co-DFA with fewer (or equal number of) states than Brzozowski's method. To this aim, let us first define the notion of *forward* and *backward* simulation. Recall that a *quasiorder* on a set X is a reflexive and transitive binary relation (not necessarily symmetric) over X.

Definition 3.5.3 (Forward simulation relation)**.** Given an NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$, a *forward simulation* on $\mathcal{N}$ is a quasiorder $\rightarrow \subseteq Q \times Q$ such that if $q \rightarrow q'$ then:

1. $q \in F$ implies $q' \in F$;
2. for every $p \in \delta(q, a)$, there exists $p' \in \delta(q', a)$ such that $p \rightarrow p'$.

Definition 3.5.4 (Backward simulation relation)**.** Given an NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$, a *backward simulation* on $\mathcal{N}$ is a quasiorder $\leftarrow \subseteq Q \times Q$ such that if $q \leftarrow q'$ then:

1. $q \in I$ implies $q' \in I$;
2. for every $q \in \delta(p, a)$, there exists $q' \in \delta(p', a)$ such that $p \leftarrow p'$.

We extend the definition of forward simulation from Q to $\wp(Q)$ as follows. Let $S, T \subseteq Q$ then $S \rightarrow T$ iff $\forall q \in S\ \exists q' \in T$ s.t. $q \rightarrow q'$. Similarly, the definition of backward simulation can be extended from Q to $\wp(Q)$ as follows. Let $S, T \subseteq Q$ then $S \leftarrow T$ iff $\forall q \in S\ \exists q' \in T$ s.t. $q \leftarrow q'$.

We will define our right (resp. left) congruence as the one induced by the intersection of a right (resp. left) quasiorder and its inverse. This way, using the notion of simulation, we define the *right simulation-based quasiorder* $\preccurlyeq^r_{\rightarrow}$ on Σ^* and its left counterpart as follows.

Definition 3.5.5 (Simulation-based quasiorders)**.** Let $u, v \in \Sigma^*$, $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA and $\rightarrow$ and $\leftarrow$ be a forward and a backward simulation on $\mathcal{N}$, respectively. Define:

$$u \preccurlyeq^r_{\rightarrow} v \stackrel{\text{def}}{\Leftrightarrow} \text{post}^{\mathcal{N}}_u(I) \rightarrow \text{post}^{\mathcal{N}}_v(I) \qquad \textit{Right} \text{ simulation-based quasiorder}$$

$$u \preccurlyeq^{\ell}_{\leftarrow} v \stackrel{\text{def}}{\Leftrightarrow} \text{pre}^{\mathcal{N}}_u(F) \leftarrow \text{pre}^{\mathcal{N}}_v(F) \qquad \textit{Left} \text{ simulation-based quasiorder}$$

Note that a quasiorder $\preccurlyeq$ and its inverse $(\preccurlyeq)^{-1}$ induces the congruence $\sim \stackrel{\text{def}}{=} \preccurlyeq\ \cap\ (\preccurlyeq)^{-1}$. In this way, we define our simulation-based congruences in terms of the simulation-based quasiorders.

Definition 3.5.6 (Simulation-based congruences)**.** Let $u, v \in \Sigma^*$, $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA and $\to$ and $\leftarrow$ be a forward and a backward simulation on $\mathcal{N}$. Define:

$$u \sim^r_{\to} v \overset{\text{def}}{\Leftrightarrow} \text{post}^{\mathcal{N}}_u(I) \to \text{post}^{\mathcal{N}}_v(I) \wedge \quad \textit{Right} \text{ simulation-based congruences}$$
$$\text{post}^{\mathcal{N}}_v(I) \to \text{post}^{\mathcal{N}}_u(I)$$
$$u \sim^{\ell}_{\leftarrow} v \overset{\text{def}}{\Leftrightarrow} \text{pre}^{\mathcal{N}}_u(F) \leftarrow \text{pre}^{\mathcal{N}}_v(F) \wedge \quad \textit{Left} \text{ simulation-based congruences}$$
$$\text{pre}^{\mathcal{N}}_v(F) \leftarrow \text{pre}^{\mathcal{N}}_u(F)$$

We give a proof of the fact that the equivalences defined above are a right and a left congruence, respectively, that precisely represent the language $\mathcal{L}(\mathcal{N})$ in Lemma 3.8.3.

The following result shows the relations between the simulation-based congruence and the congruences we defined in the previous section.

Lemma 3.5.7. *Let $\mathcal{N}$ be an automaton with $L = \mathcal{L}(\mathcal{N})$, and let $\to$ and $\leftarrow$ be a forward and backward simulation on $\mathcal{N}$, respectively. Then,*

1. $\sim^r_{\mathcal{N}} \subseteq \sim^r_{\to} \subseteq \sim^r_L$, *and*
2. $\sim^{\ell}_{\mathcal{N}} \subseteq \sim^{\ell}_{\leftarrow} \subseteq \sim^{\ell}_L$.

Proof. We will prove statement 1. (statement 2. goes similarly). First, we prove that $\sim^r_{\mathcal{N}} \subseteq \sim^r_{\to}$. By definition, $u \sim^r_{\mathcal{N}} v$ iff $\text{post}^{\mathcal{N}}_u(I) = \text{post}^{\mathcal{N}}_v(I)$, with $u, v \in \Sigma^*$. Therefore, we have that $\text{post}^{\mathcal{N}}_u(I) \to \text{post}^{\mathcal{N}}_v(I)$ and $\text{post}^{\mathcal{N}}_u(I) \leftarrow \text{post}^{\mathcal{N}}_v(I)$ when $\to$ and $\leftarrow$ are both defined by the equality $=$.

Now, we prove $\sim^r_{\to} \subseteq \sim^r_L$. First, we prove that if $q \to q'$ then $W^{\mathcal{N}}_{q,F} \subseteq W^{\mathcal{N}}_{q',F}$. Recall that $q \to q'$ implies that: (i) $q \in F$ then $q' \in F$; (ii) for every $p \in \delta(q, a)$, there exists $p' \in \delta(q', a)$ such that $p \to p'$. If $w \in W^{\mathcal{N}}_{q,F}$ then there exists $p \in F$ such that $p \in \text{post}^{\mathcal{N}}_w(q)$. Since $q \to q'$, using condition (i), there exists $p' \in \text{post}^{\mathcal{N}}_w(q')$ with $p' \to p$, and thus $p' \in F$. Therefore, $w \in W^{\mathcal{N}}_{q',F}$.

By the previous result, we have that the condition $\text{post}^{\mathcal{N}}_u(I) \to \text{post}^{\mathcal{N}}_v(I)$ and $\text{post}^{\mathcal{N}}_v(I) \to \text{post}^{\mathcal{N}}_u(I)$ implies: $W^{\mathcal{N}}_{\text{post}^{\mathcal{N}}_u(I),F} = W^{\mathcal{N}}_{\text{post}^{\mathcal{N}}_v(I),F}$. Therefore, since $W^{\mathcal{N}}_{\text{post}^{\mathcal{N}}_u(I)} = W^{\mathcal{N}}_{\text{post}^{\mathcal{N}}_v(I)} \Leftrightarrow u^{-1}L = v^{-1}L$, we have that $\text{post}^{\mathcal{N}}_u(I) \to \text{post}^{\mathcal{N}}_v(I)$ and $\text{post}^{\mathcal{N}}_v(I) \to \text{post}^{\mathcal{N}}_u(I)$ implies that $u^{-1}L = v^{-1}L$. We thus conclude that $\sim^r_{\to} \subseteq \sim^r_L$.

Note that $\sim^r_{\rightarrow}$ is a finite-index congruence since $\sim^r_{\mathcal{N}} \subseteq \sim^r_{\rightarrow}$ and $\sim^r_{\mathcal{N}}$ is of finite index. The same holds for its left counterpart. Let us define the constructions H^r and H^ℓ when applied to a right and left simulation-based congruence, respectively, as follows.

Definition 3.5.8. Let $\mathcal{N}$ be an NFA with $L = \mathcal{L}(\mathcal{N})$ and let $\rightarrow$ and $\leftarrow$ be a forward and backward simulation on $\mathcal{N}$, respectively. Define:

$$\mathsf{Sim}^r(\mathcal{N}, \rightarrow) \stackrel{\text{def}}{=} \mathsf{H}^r(\sim^r_{\rightarrow}, L) \qquad \mathsf{Sim}^\ell(\mathcal{N}, \leftarrow) \stackrel{\text{def}}{=} \mathsf{H}^\ell(\sim^\ell_{\leftarrow}, L) \ .$$

The following result is a consequence of Theorem 3.5.2.

Corollary 3.5.9. *Let $\mathcal{N}$ be an NFA with $L = \mathcal{L}(\mathcal{N})$, and let $\rightarrow$ be a forward simulation on $\mathcal{N}^R$. Then, $\mathsf{Det}^r(\mathsf{Sim}^r(\mathcal{N}^R, \rightarrow)^R)$ is isomorphic to the minimal DFA for $\mathcal{L}(\mathcal{N})$.*

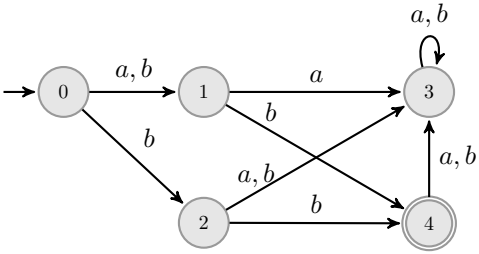
Note that, relying on Lemma 3.2.6, $\mathsf{Det}^r(\mathsf{Sim}^r(\mathcal{N}^R, \rightarrow)^R)$ is isomorphic to $\mathsf{Det}^r(\mathsf{Sim}^\ell(\mathcal{N}, \leftarrow))$. Below these lines we show an example of an NFA $\mathcal{N}$ for which the co-DFA $\mathsf{Sim}^r(\rightarrow, \mathcal{N}^R)^R$, for a given forward simulation $\rightarrow$, has fewer states than the co-DFA $\mathsf{Det}^r(\mathcal{N}^R)^R$.

Example 3.5.10. For the sake of clarity, we give directly the NFA $\mathcal{N}^R$ (instead of $\mathcal{N}$) in Figure 3.2a. Let us define the forward simulation $\rightarrow$ on $\mathcal{N}^R$ as $\rightarrow \stackrel{\text{def}}{=} \{(1,2),(2,1),(3,4)\}$, where we have omitted the pairs (i,i) for each $i \in \{0,\ldots,4\}$, for simplicity. Figure 3.2b shows the DFA $\mathsf{Det}^r(\mathcal{N}^R)$. Intuitively, its states correspond to all subsets of reachable states of $\mathcal{N}^R$ of the form $\mathrm{post}^{\mathcal{N}}_u(I)$, for all $u \in \Sigma^*$. On the other hand, Figure 3.2c shows the DFA $\mathsf{Sim}^r(\mathcal{N}^R, \rightarrow)$. In this case, its states are all subsets of reachable states of $\mathcal{N}^R$ where we have merged those subsets $\mathrm{post}^{\mathcal{N}}_u(I)$ and $\mathrm{post}^{\mathcal{N}}_v(I)$ such that $\mathrm{post}^{\mathcal{N}}_u(I) \rightarrow \mathrm{post}^{\mathcal{N}}_v(I)$ and $\mathrm{post}^{\mathcal{N}}_v(I) \rightarrow \mathrm{post}^{\mathcal{N}}_u(I)$, for each $u, v \in \Sigma^*$. Note that $\mathsf{Sim}^r(\mathcal{N}^R, \rightarrow)$ has 2 fewer states than $\mathsf{Det}^r(\mathcal{N}^R)$. When reversing both DFAs and applying Det^r construction, the resulting (minimal) DFAs will be isomorphic.

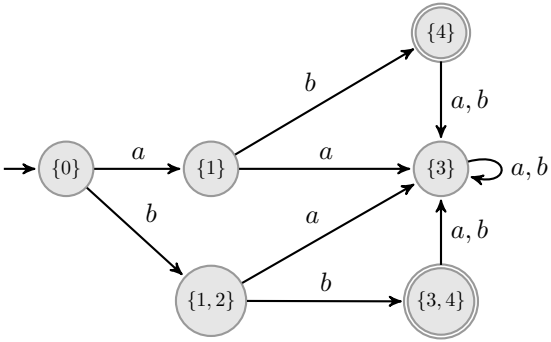
◀

3.5.3 Generalization of the Double-Reversal Method

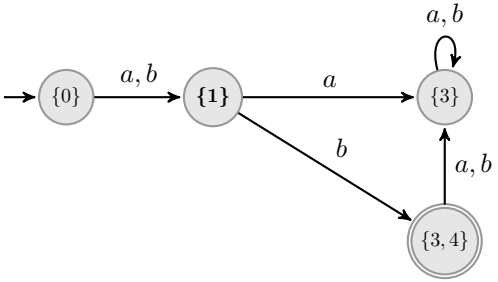
Brzozowski and Tamm [18] generalized the double-reversal algorithm by defining a necessary and sufficient condition on an NFA which guarantees that the determinized automaton is minimal. They introduced the notion of *atoms* of the language and the so-called *atomic* NFAs, and showed



(a) NFA $\mathcal{N}^R$.



(b) DFA $\mathsf{Det}^r(\mathcal{N}^R)$.



(c) DFA $\mathsf{Sim}^r(\mathcal{N}^R, \to)$. Since the pairs $(1,1)$ and $(2,1) \in \to$, then $\{1\} = \mathrm{post}_a^{\mathcal{N}^R}(I) \to \mathrm{post}_b^{\mathcal{N}^R} = \{1,2\}$ and $\{1,2\} = \mathrm{post}_b^{\mathcal{N}^R}(I) \to \mathrm{post}_a^{\mathcal{N}^R} = \{1\}$, and thus $a \sim_{\to}^r b$. As a result, states $\{1\}$ and $\{1,2\}$ of $\mathsf{Det}^r(\mathcal{N}^R)$ are merged into the state $\mathbf{\{1\}}$ in $\mathsf{Sim}^r(\mathcal{N}^R, \to)$. Note that, since $(4,3) \notin \to$ we cannot merge states $\{3\}, \{4\}$ and $\{3,4\}$.

Figure 3.2: NFA $\mathcal{N}^R$ and DFAs $\mathsf{Det}^r(\mathcal{N}^R)$ and $\mathsf{Sim}^r(\mathcal{N}^R, \to)$.

that $\mathcal{N}^D$ is minimal iff $\mathcal{N}^R$ is atomic.[1] We will show that this result is equivalent to Theorem 3.4.2 due to the left-right duality between the language-based equivalences (Lemma 3.2.6). We recall here the definition of *atom* and *atomic* NFA.

Definition 3.5.11 (Atom [18])**.** Let L be a regular language L. Let $\{K_i \mid 0 \leq i \leq n-1\}$ be the set of left quotients of L. An *atom* is any non-empty intersection of the form $\widetilde{K_0} \cap \widetilde{K_1} \cap \ldots \cap \widetilde{K_{n-1}}$, where each $\widetilde{K_i}$ is either K_i or K_i^c.

The notion of atom coincides with that of equivalence class for the left language-based congruence $\sim_L^\ell$. This was first noticed by Iván [57].

Lemma 3.5.12. *Let L be a regular language. Then for every $u \in \Sigma^*$,*

$$P_{\sim_L^\ell}(u) = \bigcap_{\substack{u \in w^{-1}L \\ w \in \Sigma^*}} w^{-1}L \cap \bigcap_{\substack{u \notin w^{-1}L \\ w \in \Sigma^*}} (w^{-1}L)^c \ .$$

Proof. For each $u \in \Sigma^*$, define $L_u = \bigcap_{\substack{u \in w^{-1}L \\ w \in \Sigma^*}} w^{-1}L \bigcap_{\substack{u \notin w^{-1}L \\ w \in \Sigma^*}} (w^{-1}L)^c$.
First, we show that $P_{\sim_L^\ell}(u) \subseteq L_u$, for each $u \in \Sigma^*$. Let $v \in P_{\sim_L^\ell}(u)$, i.e., $Lu^{-1} = Lv^{-1}$. Then, for each $w \in \Sigma^*$, $u \in w^{-1}L \Leftrightarrow wu \in L \Leftrightarrow w \in Lu^{-1} \Leftrightarrow w \in Lv^{-1} \Leftrightarrow v \in w^{-1}L$. Therefore, $\forall v \in P_{\sim_L^\ell}(u)$, $v \in L_u$, and thus $P_{\sim_L^\ell}(u) \subseteq L_u$.

Next, we show that $L_u \subseteq P_{\sim_L^\ell}(u)$. Let $v \in L_u$. Then, $\forall w \in \Sigma^*$, $u \in w^{-1}L \Leftrightarrow v \in w^{-1}L$. It follows that $w \in Lu^{-1} \Leftrightarrow w \in Lv^{-1}$, and therefore $v \in P_{\sim_L^\ell}(u)$.

Definition 3.5.13 (Atomic NFA [18])**.** An NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ is *atomic* iff for every state $q \in Q$, the right language $W_{q,F}^{\mathcal{N}}$ is a union of atoms of $\mathcal{L}(\mathcal{N})$.

Remark 3.5.14. Observe that a set $S \subseteq \Sigma^*$ is a union of atoms iff $P_{\sim_L^\ell}(S) = S$. Then, an NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ with $L = \mathcal{L}(\mathcal{N})$ is *atomic* iff for every state $q \in Q$, $P_{\sim_L^\ell}(W_{q,F}^{\mathcal{N}}) = W_{q,F}^{\mathcal{N}}$.

Similarly, we characterize the equivalence classes of the right language-based congruence $\sim_L^r$ as nonempty intersections of complement or uncomplemented *right* quotients.

[1] A shorter version of these results was previously presented in [17].

Lemma 3.5.15. *Let L be a regular language. Then, for every $u \in \Sigma^*$,*

$$P_{\sim_L^r}(u) = \bigcap_{\substack{u \in Lw^{-1} \\ w \in \Sigma^*}} Lw^{-1} \cap \bigcap_{\substack{u \notin Lw^{-1} \\ w \in \Sigma^*}} (Lw^{-1})^c \ .$$

Proof. For each $u \in \Sigma^*$, define $L_u = \bigcap_{\substack{u \in Lw^{-1} \\ w \in \Sigma^*}} Lw^{-1} \bigcap_{\substack{u \notin Lw^{-1} \\ w \in \Sigma^*}} (Lw^{-1})^c$. First, we show that $P_{\sim_L^r}(u) \subseteq L_u$, for each $u \in \Sigma^*$. Let $v \in P_{\sim_L^r}(u)$, i.e., $u^{-1}L = v^{-1}L$. Then, for each $w \in \Sigma^*$, $u \in Lw^{-1} \Leftrightarrow uw \in L \Leftrightarrow w \in u^{-1}L \Leftrightarrow w \in v^{-1}L \Leftrightarrow v \in Lw^{-1}$. Therefore, $\forall v \in P_{\sim_L^r}(u),\ v \in L_u$, and thus $P_{\sim_L^r}(u) \subseteq L_u$.

Next, we show that $L_u \subseteq P_{\sim_L^r}(u)$. Let $v \in L_u$. Then, $\forall w \in \Sigma^*$, $u \in Lw^{-1} \Leftrightarrow v \in Lw^{-1}$. It follows that $w \in u^{-1}L \Leftrightarrow w \in v^{-1}L$, and therefore $v \in P_{\sim_L^r}(u)$.

Along the lines of the notion of atom, we coin the equivalence classes of $\sim_L^r$ *co-atoms*. Thus, we define a *co-atomic* NFA as follows.

Definition 3.5.16 (Co-atomic NFA)**.** An NFA $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ is *co-atomic* iff for every state $q \in Q$, the left language $W_{I,q}^{\mathcal{N}}$ is a union of co-atoms of $\mathcal{L}(\mathcal{N})$.

We are now in condition to give an alternative proof of the generalization of Brzozowski and Tamm [18] relying on Theorem 3.4.2.

Lemma 3.5.17. *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA with $L = \mathcal{L}(\mathcal{N})$. Then $\mathcal{N}^R$ is atomic iff $\mathsf{Det}^r(\mathcal{N})$ is the minimal DFA for L.*

Proof. Let $\mathcal{N}^R = (Q, \Sigma, \delta_r, F, I)$ and $L^R = \mathcal{L}(\mathcal{N}^R)$. By Remark 3.5.14,

$$\begin{aligned}
&\forall q \in Q,\ P_{\sim_{L^R}^{\ell}}(W_{q,I}^{\mathcal{N}^R}) = W_{q,I}^{\mathcal{N}^R} \Leftrightarrow && [\text{By } A = B \Leftrightarrow A^R = B^R] \\
&\forall q \in Q,\ \left(P_{\sim_{L^R}^{\ell}}(W_{q,I}^{\mathcal{N}^R})\right)^R = \left(W_{q,I}^{\mathcal{N}^R}\right)^R \Leftrightarrow && [\text{By } u \sim_L^{\ell} v \Leftrightarrow u^R \sim_{L^R}^{r} v^R] \\
&\forall q \in Q,\ P_{\sim_L^r}\left(\left(W_{q,I}^{\mathcal{N}^R}\right)^R\right) = \left(W_{q,I}^{\mathcal{N}^R}\right)^R \Leftrightarrow && [\text{By } \left(W_{q,I}^{\mathcal{N}^R}\right)^R = W_{I,q}^{\mathcal{N}}] \\
&\forall q \in Q,\ P_{\sim_L^r}(W_{I,q}^{\mathcal{N}}) = W_{I,q}^{\mathcal{N}} \ .
\end{aligned}$$

It follows from Theorem 3.4.2 that $\mathsf{Det}^r(\mathcal{N})$ is minimal.

In view of the notion of co-atomic NFA and the above result, the sufficient and necessary condition by Brzozowski and Tamm [18] can be formulated as follows.

Corollary 3.5.18. *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA with $L = \mathcal{L}(\mathcal{N})$. Then $\mathcal{N}$ is co-atomic iff $\mathsf{Det}^r(\mathcal{N})$ is the minimal DFA for L.*

We conclude this section by compiling all the conditions described so far that guarantee that determinizing an automaton yields the minimal DFA.

Corollary 3.5.19. *Let $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ be an NFA with $L = \mathcal{L}(\mathcal{N})$. The following conditions are equivalent:*

1. $\mathsf{Det}^r(\mathcal{N})$ *is minimal.*
2. $\sim^r_{\mathcal{N}} = \sim^r_L$.
3. $\forall u, v \in \Sigma^*,\ W^{\mathcal{N}}_{\mathrm{post}^{\mathcal{N}}_u(I),F} = W^{\mathcal{N}}_{\mathrm{post}^{\mathcal{N}}_v(I),F} \Leftrightarrow \mathrm{post}^{\mathcal{N}}_u(I) = \mathrm{post}^{\mathcal{N}}_v(I)$.
4. $\forall q \in Q,\ P_{\sim^r_L}(W^{\mathcal{N}}_{I,q}) = W^{\mathcal{N}}_{I,q}$.
5. $\mathcal{N}^R$ *is atomic.*
6. $\mathcal{N}$ *is co-atomic.*

3.5.4 Moore's Algorithm

Given a DFA $\mathcal{D}$, Moore [73] builds the minimal DFA for the language $L = \mathcal{L}(\mathcal{D})$ by removing unreachable states from $\mathcal{D}$ and then performing a stepwise refinement of an initial partition of the set of reachable states. Since we are interested in the refinement step, in what follows we assume that all DFAs have no unreachable states.

In this section, we will describe Moore's state-partition $\mathcal{Q}^{\mathcal{D}}$ and the right language-based partition $P_{\sim^r_L}$ as greatest fixpoint computations and show that there exists an isomorphism between the two at each step of the fixpoint computation. In fact, this isomorphism shows that the output DFA M of Moore's algorithm satisfies $P_{\sim^r_L}(W^M_{I,q}) = W^M_{I,q}$, for every state q. Thus, by Theorem 3.4.2, M is isomorphic to the minimal DFA for L.

First, we give Moore's algorithm which computes the state-partition that is later used to define Moore's DFA.

Moore's Algorithm: Algorithm that builds Moore's partition.

Data: DFA $\mathcal{D} = (Q, \Sigma, \delta, I, F)$ with $L = \mathcal{L}(\mathcal{D})$.
Result: $\mathcal{Q}^{\mathcal{D}} \in Part(Q)$.

```
Q^D := {F, F^c}, Q' := ∅;
while Q^D ≠ Q' do
    Q' := Q^D;
    forall a ∈ Σ do
        Q_a := ⨅_{p∈Q^D} {pre_a^D(p), (pre_a^D(p))^c};
    Q^D := Q^D ⩘ ⨅_{a∈Σ} Q_a;
return Q^D;
```

Definition 3.5.20 (Moore's DFA)**.** Let $\mathcal{D} = (Q, \Sigma, \delta, I, F)$ be a DFA, and let $\mathcal{Q}^{\mathcal{D}}$ be the output state-partition of Moore's algorithm. *Moore's DFA* for $\mathcal{L}(\mathcal{D})$ is $M = (Q^M, \Sigma, \delta^M, I^M, F^M)$ where $Q^M = \mathcal{Q}^{\mathcal{D}}$, $I^M = \{\mathcal{Q}^{\mathcal{D}}(q) \mid q \in I\}$, $F^M = \{\mathcal{Q}^{\mathcal{D}}(q) \mid q \in F\}$ and, for each $S, S' \in Q^M$ and $a \in \Sigma$, we have that $\delta^M(S, a) = S'$ iff $\exists q \in S, q' \in S'$ with $\delta(q, a) = q'$.

Next, we describe Moore's state-partition $\mathcal{Q}^{\mathcal{D}}$ and the right language-based partition $P_{\sim_L^r}$ as greatest fixpoint computations and show that there exists an isomorphism between the two at each step of the fixpoint computation.

Definition 3.5.21 (Moore's state-partition)**.** Let $\mathcal{D} = (Q, \Sigma, \delta, I, F)$ be a DFA. Define *Moore's state-partition w.r.t.* $\mathcal{D}$, denoted by $\mathcal{Q}^{\mathcal{D}}$, as follows.

$$\mathcal{Q}^{\mathcal{D}} \stackrel{\text{def}}{=} \text{gfp}(\lambda X. \bigwedge_{a \in \Sigma, S \in X} \{\text{pre}_a(S), (\text{pre}_a(S))^c\} \curlywedge \{F, F^c\}) \ .$$

By Theorem 3.3.7 (2), each state of the minimal DFA for L corresponds to an equivalence class of $\sim_L^r$. Relying on Lemma 3.5.15, these equivalence classes can be defined in terms of non-empty intersections of complemented or uncomplemented right quotients of L. In other words,

$$P_{\sim_L^r} = \bigwedge_{w \in \Sigma^*} \{Lw^{-1}, (Lw^{-1})^c\} \ ,$$

for every regular language L . Thus, $P_{\sim_L^r}$ can also be obtained as a greatest fixpoint computation as follows. We defer a proof of this result to Section 3.8.

Lemma 3.5.22. *Let L be a regular language. Then*

$$P_{\sim_L^r} = \mathrm{gfp}(\lambda X. \bigwedge_{a\in\Sigma, B\in X} \{Ba^{-1}, (Ba^{-1})^c\} \curlywedge \{L, L^c\}) \ . \tag{3.9}$$

Now we show that, given a DFA $\mathcal{D}$ with $L = \mathcal{L}(\mathcal{D})$, there exists a partition isomorphism between $\mathcal{Q}^{\mathcal{D}}$ and $P_{\sim_L^r}$ at each step of the fixpoint computations given in Definition 3.5.21 and Lemma 3.5.22 respectively.

Theorem 3.5.23. *Let $\mathcal{D} = (Q, \Sigma, \delta, I, F)$ be a DFA with $L = \mathcal{L}(\mathcal{D})$ and let $\varphi : \wp(Q) \to \wp(\Sigma^*)$ be a function defined by $\varphi(S) \stackrel{\text{def}}{=} W^{\mathcal{D}}_{I,S}$. Let $\mathcal{Q}^{\mathcal{D}(n)}$ and $P^{(n)}_{\sim_L^r}$ be the n-th step of the fixpoint computation of $\mathcal{Q}^{\mathcal{D}}$ (Definition 3.5.21) and $P_{\sim_L^r}$ (Lemma 3.5.22), respectively. Then, φ is an isomorphism between $\mathcal{Q}^{\mathcal{D}(n)}$ and $P^{(n)}_{\sim_L^r}$ for each $n \geq 0$.*

Proof. In order to show that φ is a partition isomorphism, it suffices to prove that φ is a bijective mapping between the partitions. We first show that $\varphi(\mathcal{Q}^{\mathcal{D}(n)}) = P^{(n)}_{\sim_L^r}$, for every $n \geq 0$. Thus, the mapping φ is surjective. Secondly, we show that φ is an injective mapping from $\mathcal{Q}^{\mathcal{D}(n)}$ to $P^{(n)}_{\sim_L^r}$. Therefore, we conclude that φ is a bijection.

To show that $\varphi(\mathcal{Q}^{\mathcal{D}(n)}) = P^{(n)}_{\sim_L^r}$, for each $n \geq 0$, we proceed by induction.

- *Base case:* By definition, $\mathcal{Q}^{\mathcal{D}(0)} = \{F, F^c\}$ and $P^{(0)}_{\sim_L^r} = \{L, L^c\}$. Since $\mathcal{D}$ is deterministic (and complete), it follows that $\varphi(F) = W^{\mathcal{D}}_{I,F} = L$ and $\varphi(F^c) = W^{\mathcal{D}}_{I,F^c} = L^c$.
- *Inductive step:* Before proceeding with the inductive step, we show that the following equations hold for each $a, b \in \Sigma$ and $S, S_i, S_j \in \mathcal{Q}^{\mathcal{D}(n)}$ with $n \geq 0$:

$$\varphi(\mathrm{pre}_a(S)^c) = ((W^{\mathcal{D}}_{I,S})a^{-1})^c \tag{3.10}$$

$$\varphi(\mathrm{pre}_a(S_i) \cap \mathrm{pre}_b(S_j)) = (W^{\mathcal{D}}_{I,S_i})a^{-1} \cap (W^{\mathcal{D}}_{I,S_j})b^{-1} \ . \tag{3.11}$$

  For each $S \in \mathcal{Q}^{\mathcal{D}(n)}$ and $a \in \Sigma$ we have the following. Note that, in cases where the equality between sets holds by definition, we simply omit the reasoning (between brackets).

$$
\begin{aligned}
\varphi(\mathrm{pre}_a(S)^c) &= \\
W^{\mathcal{D}}_{I,\mathrm{pre}_a(S)^c} &= \\
\{w \in \Sigma^* \mid \exists q \in \mathrm{pre}_a(S)^c,\ q = \hat{\delta}(q_0, w)\} &=^{\dagger} \\
\{w \in \Sigma^* \mid \exists q \in \mathrm{pre}_a(S),\ q = \hat{\delta}(q_0, w)\}^c &= \\
\{w \in \Sigma^* \mid \exists q \in S,\ q = \hat{\delta}(q_0, wa)\}^c &= \\
((W^{\mathcal{D}}_{I,S})a^{-1})^c\ , &
\end{aligned}
$$

where the equality marked with † holds since $\mathcal{D}$ is a (complete) DFA.

Therefore, Equation (3.10) holds at each step of the fixpoint computation. Consider now Equation (3.11). Let $S_i, S_j \in \mathcal{Q}^{\mathcal{D}(n)}$. Then,

$$
\begin{aligned}
\varphi(\mathrm{pre}_a(S_i) \cap \mathrm{pre}_b(S_j)) &= \\
W^{\mathcal{D}}_{I,(\mathrm{pre}_a(S_i) \cap \mathrm{pre}_b(S_j))} &= \\
\{w \in \Sigma^* \mid \exists q \in \mathrm{pre}_a(S_i) \cap \mathrm{pre}_b(S_j), q = \hat{\delta}(q_0, w)\} &= \\
\{w \in \Sigma^* \mid \exists q \in \mathrm{pre}_a(S_i),\ q \in \mathrm{pre}_b(S_j), q = \hat{\delta}(q_0, w)\} &=^{\dagger} \\
W^{\mathcal{D}}_{I,\mathrm{pre}_a(S_i)} \cap W^{\mathcal{D}}_{I,\mathrm{pre}_b(S_j)} &= \\
(W^{\mathcal{D}}_{I,S_i})a^{-1} \cap (W_{I,S_j})b^{-1}\ , &
\end{aligned}
$$

where equality † holds since $\mathcal{D}$ is a DFA.

Therefore, Equation (3.11) holds at each step of the fixpoint computation.

Assume that $\varphi\left(\mathcal{Q}^{\mathcal{D}(n)}\right) = P^{(n)}_{\sim^r_L}$ for every $n \leq k$ $(k > 0)$. Then,

$$
\begin{aligned}
\varphi\big(\mathcal{Q}^{\mathcal{D}(k+1)}\big) &= \quad \text{[Def. 3.5.21]} \\
\varphi\Big(\bigwedge_{\substack{a \in \Sigma \\ S \in X}} \{\mathrm{pre}_a(S), \mathrm{pre}_a(S)^c\} \curlywedge \{F, F^c\}\Big) &= \quad \text{[Eqs. (3.10) and (3.11)]}
\end{aligned}
$$

$$\bigwedge_{\substack{a\in\Sigma\\ \varphi(S)\in\varphi(X)}} \{(W^{\mathcal{D}}_{I,S})a^{-1}, ((W^{\mathcal{D}}_{I,S})a^{-1})^c\} \curlywedge \{L, L^c\} = \quad \text{[I.H., } \varphi(X) = P^{(k)}_{\sim^r_L}\text{]}$$

$$\bigwedge_{a\in\Sigma, B\in X'} \{Ba^{-1}, (Ba^{-1})^c\} \curlywedge \{L, L^c\} = \quad \text{[Lemma 3.5.22]}$$

$$P^{(k+1)}_{\sim^r_L} \ .$$

Note that in the first equality $X \stackrel{\text{def}}{=} \mathcal{Q}^{\mathcal{D}(k)}$, and in the last equality $X' \stackrel{\text{def}}{=} P^{(k)}_{\sim^r_L}$. Finally, since $\mathcal{D}$ is a DFA then, for each $S_i, S_j \in \mathcal{Q}^{\mathcal{D}(n)} (n \geq 0)$ with $S_i \neq S_j$ we have that $W^{\mathcal{D}}_{I,S_i} \neq W^{\mathcal{D}}_{I,S_j}$, i.e., $\varphi(S_i) \neq \varphi(S_j)$. Therefore, φ is an injective mapping.

Corollary 3.5.24. *Let $\mathcal{D}$ be a DFA with $L = \mathcal{L}(\mathcal{D})$. Let $\mathcal{Q}^{\mathcal{D}(n)}$ and $P^{(n)}_{\sim^r_L}$ be the n-th step of the fixpoint computation of $\mathcal{Q}^{\mathcal{D}}$ and $P_{\sim^r_L}$, respectively. Then, for each $n \geq 0$,*

$$P^{(n)}_{\sim^r_L}(W^{\mathcal{D}}_{I,S}) = W^{\mathcal{D}}_{I,S} \ , \text{ for each } S \in \mathcal{Q}^{\mathcal{D}(n)} \ .$$

It follows that Moore's DFA M, whose set of states corresponds to the state-partition at the end of the execution of Moore's algorithm, satisfies that, for each $q \in Q^M$, $P_{\sim^r_L}(W^M_{I,q}) = W^M_{I,q}$ with $L = \mathcal{L}(M)$. Finally, by Theorem 3.4.2, we have that $\mathsf{Det}^r(M) (= M$, since M is a DFA) is minimal.

For the sake of completeness, we prove that indeed M is isomorphic to $\mathsf{Min}^r(\mathcal{L}(\mathcal{D}))$. We defer the proof to Section 3.8.

Theorem 3.5.25. *Let $\mathcal{D}$ be a DFA and M be Moore's DFA for $\mathcal{L}(\mathcal{D})$ as in Definition 3.5.20. Then, M is isomorphic to $\mathsf{Min}^r(\mathcal{L}(\mathcal{D}))$.*

Finally, Hopcroft [51] devised a DFA minimization algorithm which offers better performance than Moore's. The ideas used by Hopcroft can be adapted to our framework to devise a new algorithm from computing $P_{\sim^r_L}$. However, by doing so, we could not derive a better explanation than the one provided by Berstel et al. [9].

3.6 Related Work

Brzozowski and Tamm [18] showed that every regular language defines a unique NFA, which they call *átomaton*. The átomaton is built upon

the minimal DFA $\mathcal{N}^{DM}$ for the language, defining its states as non-empty intersections of complemented or uncomplemented right languages of $\mathcal{N}^{DM}$, i.e., the atoms of the language, which are thus intersections of complemented or uncomplemented left quotients of the language. Then they proved that the átomaton is isomorphic to the reverse automaton of the minimal deterministic DFA for the reverse language.

Intuitively, the construction of the átomaton based on the right languages of the minimal DFA corresponds to $\mathsf{Det}^{\ell}(\mathcal{N}^{DM})$, while its construction based on left quotients of the language corresponds to $\mathsf{Min}^{\ell}(\mathcal{L}(\mathcal{N}))$.

Corollary 3.6.1. *Let $\mathcal{N}^{DM}$ be the minimal DFA for a regular language L. Then,*

1. $\mathsf{Det}^{\ell}(\mathcal{N}^{DM})$ *is isomorphic to the átomaton of L.*
2. $\mathsf{Min}^{\ell}(L)$ *is isomorphic to the átomaton of L.*

In the same paper, they also defined the notion of *partial átomaton* which is built upon an NFA $\mathcal{N}$. Each state of the partial átomaton is a non-empty intersection of complemented or uncomplemented right languages of $\mathcal{N}$, i.e., union of atoms of the language. Intuitively, the construction of the partial átomaton corresponds to $\mathsf{Det}^{\ell}(\mathcal{N})$.

Corollary 3.6.2. *Let $\mathcal{N}$ be an NFA. Then, $\mathsf{Det}^{\ell}(\mathcal{N})$ is isomorphic to the partial átomaton of $\mathcal{N}$.*

Finally, they also presented a number of results [18, Theorem 3] related to the átomaton $\mathcal{A}$ of a minimal DFA $\mathcal{D}$ with $L = \mathcal{L}(\mathcal{D})$:

1. $\mathcal{A}$ is isomorphic to $\mathcal{D}^{RDR}$.
2. $\mathcal{A}^R$ is the minimal DFA for L^R
3. $\mathcal{A}^D$ is the minimal DFA for L.
4. $\mathcal{A}$ is isomorphic to $\mathcal{N}^{RDMR}$ for every NFA $\mathcal{N}$ accepting L.

All these relations can be inferred from Figure 3.3 which connects all the automata constructions described in this paper together with those introduced by Brzozowski and Tamm. For instance, property 1 corresponds to the path starting at $\mathcal{N}^{DM}$ (the minimal DFA for $\mathcal{L}(\mathcal{N})$), labeled with $R - \mathsf{Det}^{r} - R$, and ending in the átomaton of $\mathcal{L}(\mathcal{N})$. On the other hand, property 4 corresponds to the path starting at $\mathcal{N}$, labeled with $R - \mathsf{Min}^{r} - R$ and ending in the átomaton of $\mathcal{L}(\mathcal{N})$. Finally, the path starting at $\mathcal{N}$,

labeled with $R - \mathsf{Det}^r - R$ and ending in the partial átomaton of $\mathcal{N}$ shows that the latter is isomorphic to $\mathcal{N}^{RDR}$.

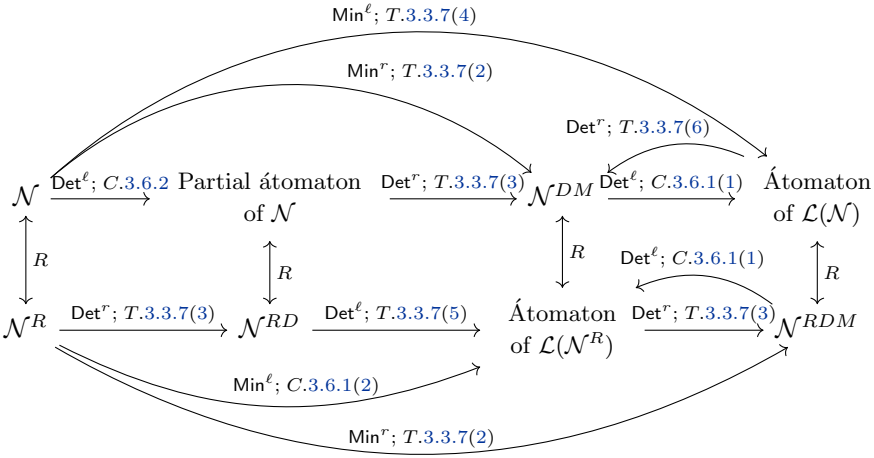


Figure 3.3: Extension of the diagram of Figure 3.1 including the átomaton and the partial átomaton. Recall that $\mathcal{N}^{DM}$ is the minimal DFA for $\mathcal{L}(\mathcal{N})$. The results referenced in the labels are those justifying the output of the operation.

3.7 Concluding Remarks

In this work, we establish a connection between well-known independent minimization methods through Theorem 3.4.2. Given a DFA, the left languages of its states form a partition on words, P, and thus each left language is identified by a state. Intuitively, Moore's algorithm merges states to enforce the condition of Theorem 3.4.2, which results in merging blocks of P that belong to the same Nerode's equivalence class. Note that Hopcroft's partition refinement method [51] achieves the same goal at the end of its execution though, stepwise, the partition computed may differ from Moore's.

On the other hand, for co-deterministic NFAs (with no empty states) the right language-based and the right automata-based congruences coin-

cide. As a result, determinizing a co-deterministic NFA yields directly the minimal DFA for the language. This is the key idea behind the double-reversal method. However, there exists a more general class of NFAs enjoying this condition, namely, the class of NFAs whose reverse is an atomic NFA [18]. We show that, relying on the left-right duality of the language-based equivalences, this generalization is equivalent to asking whether the left languages of the NFA are represented precisely by the abstraction given by right Nerode's congruence (see Theorem 3.4.2).

Some of these connections have already been studied in order to offer a better understanding of Brzozowski's double-reversal method. In particular, Courcelle et al. [25] offer an alternative view of minimization and determinization of finite-state automata using notions on rectangular decompositions of relations over words. When these relations are left and right congruences we obtain the same notion of left-right duality through the reverse operation. In our work, we further exploit these congruences to define automata constructions that allows us to devise more efficient versions of the double-reversal method in a systematic way. Furthermore, we use the notion of duality to reformulate the later generalization of the double-reversal method, and thus shed light on the connection between this method and Moore's algorithm. Other uniform views to this method are offered by Adámek et al. [1] and Bonchi et al. [12] from a category-theoretic perspective. Interestingly, Bonchi et al. [12] devise a notion of duality, in this case, between the concepts of reachability and observability of transition systems, that allows them to explain the double-reversal method. Regarding these alternative approaches, our work revisits minimization techniques relying on simple language-theoretic notions.

3.8 Supplementary Proofs

In this section we defer supplementary and technical proofs of results presented in this chapter.

Lemma 3.8.1. *Let $L \subseteq \Sigma^*$ be a regular language. Then, the following holds:*

1. $\sim_L^r$ *is a right congruence;*
2. $\sim_L^\ell$ *is a left congruence; and*
3. $P_{\sim_L^r}(L) = L = P_{\sim_L^\ell}(L)$.

Proof. Let us prove that $\sim_L^r$ is a right congruence. Assume $u \sim_L^r v$, i.e., $u^{-1}L = v^{-1}L$. Given $x \in \Sigma^*$, we have that,

$$(ux)^{-1}L = x^{-1}(u^{-1}L) = x^{-1}(v^{-1}L) = (vx)^{-1}L \ .$$

Therefore, $ux \sim_L^r vx$.

Now, let us prove that $\sim_L^\ell$ is a left congruence. Assume $u \sim_L^\ell v$, i.e., $Lu^{-1} = Lv^{-1}$. Given $x \in \Sigma^*$, we have that,

$$L(xu)^{-1} = (Lu^{-1})x^{-1} = (Lv^{-1})x^{-1} = L(xv)^{-1} \ .$$

Therefore, $xu \sim_L^r xv$.

Finally, let $P_{\sim_L^r}$ be the finite partition induced by $\sim_L^r$. We show that $P_{\sim_L^r}(L) = L$. First note that $L \subseteq P_{\sim_L^r}(L)$ by the reflexivity of the equivalence relation $\sim_L^r$. On the other hand, we prove that for every $u \in \Sigma^*$, if $u \in P_{\sim_L^r}(L)$ then $u \in L$. By hypothesis, there exists $v \in L$ such that $u \sim_L^r v$, i.e., $u^{-1}L = v^{-1}L$. Since $v \in L$ then $\varepsilon \in v^{-1}L$. Therefore, $\varepsilon \in u^{-1}L$ and we conclude that $u \in L$.

The proof of $P_{\sim_L^\ell}(L) = L$ goes similarly.

Lemma 3.8.2. *Let $\mathcal{N}$ be an NFA. Then, the following holds:*

1. *$\sim_{\mathcal{N}}^r$ is a right congruence;*
2. *$\sim_{\mathcal{N}}^\ell$ is a left congruence; and*
3. *$P_{\sim_{\mathcal{N}}^r}(\mathcal{L}(\mathcal{N})) = \mathcal{L}(\mathcal{N}) = P_{\sim_{\mathcal{N}}^\ell}(\mathcal{L}(\mathcal{N}))$.*

Proof. Let us prove that $\sim_{\mathcal{N}}^r$ is a right congruence. Assume $u \sim_{\mathcal{N}}^r v$, i.e., $\mathrm{post}_u^{\mathcal{N}}(I) = \mathrm{post}_v^{\mathcal{N}}(I)$. Given $x \in \Sigma^*$, we have that,

$$\mathrm{post}_{ux}^{\mathcal{N}}(I) = \mathrm{post}_x^{\mathcal{N}}(\mathrm{post}_u^{\mathcal{N}}(I)) = \mathrm{post}_x^{\mathcal{N}}(\mathrm{post}_v^{\mathcal{N}}(I)) = \mathrm{post}_{vx}^{\mathcal{N}}(I) \ .$$

Therefore, $ux \sim_{\mathcal{N}}^r vx$.

Now, let us prove that $\sim_{\mathcal{N}}^\ell$ is a left congruence. Assume $u \sim_{\mathcal{N}}^\ell v$, i.e., $\mathrm{pre}_u^{\mathcal{N}}(F) = \mathrm{pre}_v^{\mathcal{N}}(F)$. Given $x \in \Sigma^*$, we have that,

$$\mathrm{pre}_{xu}^{\mathcal{N}}(F) = \mathrm{pre}_u^{\mathcal{N}}(\mathrm{pre}_x^{\mathcal{N}}(F)) = \mathrm{pre}_v^{\mathcal{N}}(\mathrm{pre}_x^{\mathcal{N}}(F)) = \mathrm{pre}_{xv}^{\mathcal{N}}(F) \ .$$

Therefore, $xu \sim_{\mathcal{N}}^r xv$.

Finally, $P_{\sim^r_{\mathcal{N}}}$, the finite partition induced by $\sim^r_{\mathcal{N}}$. We show that $P_{\sim^r_{\mathcal{N}}}(\mathcal{L}(\mathcal{N})) = \mathcal{L}(\mathcal{N})$. First note that $\mathcal{L}(\mathcal{N}) \subseteq P_{\sim^r_{\mathcal{N}}}(\mathcal{L}(\mathcal{N}))$ by the reflexivity of the equivalence relation $\sim^r_{\mathcal{N}}$. On the other hand, we prove that for every $u \in \Sigma^*$, if $u \in P_{\sim^r_{\mathcal{N}}}(\mathcal{L}(\mathcal{N}))$ then $u \in \mathcal{L}(\mathcal{N})$. By hypothesis, there exists $v \in \mathcal{L}(\mathcal{N})$ such that $u \sim^r_{\mathcal{N}} v$, i.e., $\text{post}^{\mathcal{N}}_u(I) = \text{post}^{\mathcal{N}}_v(I)$. Since $v \in \mathcal{L}(\mathcal{N})$ then $\text{post}^{\mathcal{N}}_v \cap F \neq \varnothing$. Thus, $\text{post}^{\mathcal{N}}_u \cap F \neq \varnothing$ and we conclude that $u \in \mathcal{L}(\mathcal{N})$.

The proof of $P_{\sim^\ell_{\mathcal{N}}}(L) = L$ goes similarly.

Lemma 3.8.3. *Let $\mathcal{N}$ be an NFA and let $\rightarrow$ and $\leftarrow$ be a foward and backward simulation on $\mathcal{N}$, respectively. Then, the following holds:*

1. *$\sim^r_{\rightarrow}$ is a right congruence;*
2. *$\sim^\ell_{\leftarrow}$ is a left congruence; and*
3. *$P_{\sim^r_{\rightarrow}}(\mathcal{L}(\mathcal{N})) = \mathcal{L}(\mathcal{N}) = P_{\sim^\ell_{\leftarrow}}(\mathcal{L}(\mathcal{N}))$.*

Proof. Let us prove that $u \sim^r_{\rightarrow} v$ is a right congruence. Assume $u \sim^r_{\rightarrow} v$, i.e., (i) $\forall q \in \text{post}_u(I)\ \exists q' \in \text{post}_v(I) : q \rightarrow q'$; and (ii) $\forall q' \in \text{post}_v(I)\ \exists q \in \text{post}_u(I) : q' \rightarrow q$.

Given $x \in \Sigma^*$, we will prove that $ux \sim^r_{\rightarrow} vx$, i.e., (i) $\forall p \in \text{post}_{ux}(I)\ \exists p' \in \text{post}_{vx}(I) : p \rightarrow p'$; and (ii) $\forall p' \in \text{post}_{vx}(I)\ \exists p \in \text{post}_{ux}(I) : p' \rightarrow p$.

We start with (i). Let $p \in \text{post}_{ux}(I)$. Then $p \in \text{post}_x(q)$ for some $q \in \text{post}_u(I)$. By hypothesis, there exists $q' \in \text{post}_v(I)$ with $q \rightarrow q'$. Since $p \in \text{post}_x(q)$, by definition of $q \rightarrow q'$, there must exist $p' \in \text{post}_x(q')$ with $p \rightarrow p'$. Therefore, there exists $p' \in \text{post}_{vx}(I)$ with $p \rightarrow p'$. The proof of (ii) goes similarly.

Now, let us prove that $u \sim^\ell_{\leftarrow} v$ is a left congruence. Assume $u \sim^\ell_{\leftarrow} v$, i.e., (i) $\forall q \in \text{pre}_u(F)\ \exists q' \in \text{pre}_v(F) : q \leftarrow q'$; and (ii) $\forall q' \in \text{pre}_v(F)\ \exists q \in \text{pre}_u(F) : q' \leftarrow q$.

Given $x \in \Sigma^*$, we will prove that $xu \sim^\ell_{\leftarrow} xv$, i.e., (i) $\forall p \in \text{pre}_{xu}(F)\ \exists p' \in \text{pre}_{xv}(F) : p \leftarrow p'$; and (ii) $\forall p' \in \text{pre}_{xv}(F)\ \exists p \in \text{pre}_{xu}(F) : p' \leftarrow p$.

We start with (i). Let $p \in \text{pre}_{xu}(F)$. Then $p \in \text{pre}_x(q)$ for some $q \in \text{pre}_u(F)$. By hypothesis, there exists $q' \in \text{pre}_v(F)$ with $q \leftarrow q'$. Since $p \in \text{pre}_x(q)$, by definition of $q \leftarrow q'$, there must exist $p' \in \text{pre}_x(q')$ with $p \leftarrow p'$. Therefore, there exists $p' \in \text{pre}_{xv}(F)$ with

$p \leftarrow p'$. The proof of (*ii*) goes similarly.

Now we prove that $P_{\sim^r_{\rightarrow}}(\mathcal{L}(\mathcal{N})) = \mathcal{L}(\mathcal{N})$. On one hand, $\mathcal{L}(\mathcal{N}) \subseteq P_{\sim^r_{\rightarrow}}(\mathcal{L}(\mathcal{N}))$ by reflexivity of $\sim^r_{\rightarrow}$. On the other hand, we prove that $P_{\sim^r_{\rightarrow}}(\mathcal{L}(\mathcal{N})) \subseteq \mathcal{L}(\mathcal{N})$. If $u \in P_{\sim^r_{\rightarrow}}(\mathcal{L}(\mathcal{N}))$ then there exists $v \in \mathcal{L}(\mathcal{N})$ s.t. $u \sim^r_{\rightarrow} v$, and, in particular, for each $q' \in \mathrm{post}_v(I)$ there exists $q \in \mathrm{post}_u(I)$ with $q' \rightarrow q$. Since $v \in \mathcal{L}(\mathcal{N})$, then there must be at least one $q' \in F$. Since $q' \rightarrow q$, necessarily $q \in F$. Therefore, $u \in \mathcal{L}(\mathcal{N})$.

Finally, we prove that $P_{\sim^\ell_{\leftarrow}}(\mathcal{L}(\mathcal{N})) = \mathcal{L}(\mathcal{N})$. On one hand, $\mathcal{L}(\mathcal{N}) \subseteq P_{\sim^\ell_{\leftarrow}}(\mathcal{L}(\mathcal{N}))$ by reflexivity of $\sim^\ell_{\leftarrow}$. On the other hand, we prove that $P_{\sim^\ell_{\leftarrow}}(\mathcal{L}(\mathcal{N})) \subseteq \mathcal{L}(\mathcal{N})$. If $u \in P_{\sim^\ell_{\leftarrow}}(\mathcal{L}(\mathcal{N}))$ then there exists $v \in \mathcal{L}(\mathcal{N})$ s.t. $u \sim^\ell_{\leftarrow} v$, and, in particular, for each $q' \in \mathrm{pre}_v(F)$ there exists $q \in \mathrm{pre}_u(F)$ with $q' \leftarrow q$. Since $v \in \mathcal{L}(\mathcal{N})$, then there must be at least one $q' \in I$. Since $q' \leftarrow q$, necessarily $q \in I$. Therefore, $u \in \mathcal{L}(\mathcal{N})$.

Lemma 3.5.22. *Let L be a regular language. Then*

$$P_{\sim^r_L} = \mathrm{gfp}(\lambda X. \bigwedge_{a\in\Sigma, B\in X} \{Ba^{-1}, (Ba^{-1})^c\} \curlywedge \{L, L^c\}) \ . \tag{3.9}$$

Proof. Let $\Sigma^{\leq n}$ (resp. Σ^n) denote the set of words with length up to n (resp. exactly n), i.e., $\Sigma^{\leq n} \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid |w| \leq n\}$ (resp. $\Sigma^n \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid |w| = n\}$). Let us denote $X^{\leq n}$, the n-th iteration of the greatest fixpoint computation of Equation (3.9). We will prove by induction on n that the following equation holds, for each $n \geq 0$:

$$X^{\leq n+1} = \bigwedge_{a\in\Sigma, B\in X^{\leq n}} \{Ba^{-1}, (Ba^{-1})^c\} \curlywedge \{L, L^c\} = \bigwedge_{w\in\Sigma^{\leq n}} \{Lw^{-1}, (Lw^{-1})^c\} \ . \tag{3.12}$$

- *Base case:* Let $n = 0$. It is easy to see that equality (3.12) holds since $\{L, L^c\} = \{L\varepsilon^{-1}, (L\varepsilon^{-1})^c\}$. Now, let $n = 1$.

Then,

$$\bigwedge_{a\in\Sigma, B\in X^{\leq 1}} \{Ba^{-1}, (Ba^{-1})^c\} \curlywedge \{L, L^c\} =$$

$$\bigwedge_{a\in\Sigma} \left(\{La^{-1}, (La^{-1})^c\} \curlywedge \{(L^c)a^{-1}, ((L^c)a^{-1})^c\}\right) \curlywedge \{L, L^c\} =$$

$$\bigwedge_{a\in\Sigma} \{La^{-1}, (La^{-1})^c\} \curlywedge \{L, L^c\} =$$

$$\bigwedge_{a\in\Sigma, w\in\Sigma^{\leq 1}} \{Lw^{-1}, (Lw^{-1})^c\} \ .$$

The first equality holds using $X^{\leq 1} = \{L, L^c\}$, the second equality holds using $(La^{-1})^c = L^c a^{-1}$, and the last equality holds using $\Sigma^{\leq 1} \stackrel{\text{def}}{=} \{\varepsilon\} \cup \Sigma$.

- *Inductive Step:* Let us assume that equality 3.12 holds for each $n \leq k$. We will prove that it holds for $n = k + 1$. Note that, using the inductive hypothesis twice, we have that:

$$X^{\leq k+1} = \bigwedge_{w\in\Sigma^{\leq k}} \{Lw^{-1}, (Lw^{-1})^c\} =$$

$$\bigwedge_{w\in\Sigma^{\leq k-1}} \{Lw^{-1}, (Lw^{-1})^c\} \curlywedge \bigwedge_{\substack{a\in\Sigma \\ w\in\Sigma^{k-1}}} \{Lw^{-1}a^{-1}, (Lw^{-1}a^{-1})^c\} =$$

$$X^{\leq k} \curlywedge \bigwedge_{w\in\Sigma^{k}} \{Lw^{-1}, (Lw^{-1})^c\} \ .$$

Using the above result, the induction hypothesis and $Ba^{-1} \cap \widetilde{B}a^{-1} = (B \cap \widetilde{B})a^{-1}$, it follows that:

$$X^{\leq k+2} = \bigwedge_{\substack{a\in\Sigma \\ B\in X^{\leq k+1}}} \{Ba^{-1}, (Ba^{-1})^c\} \curlywedge \{L, L^c\} =$$

$$\bigwedge_{\substack{a\in\Sigma \\ B\in X^{\leq k}}} \{Ba^{-1}, (Ba^{-1})^c\} \curlywedge \bigwedge_{\substack{a\in\Sigma \\ B\in X^{k}}} \{Ba^{-1}, (Ba^{-1})^c\} \curlywedge \{L, L^c\} =^{\dagger}$$

$$\bigwedge_{w\in\Sigma^{\leq k}} \{Lw^{-1}, (Lw^{-1})^c\} \curlywedge \bigwedge_{w\in\Sigma^{k+1}} \{Lw^{-1}, (Lw^{-1})^c\} \curlywedge \{L, L^c\} = \bigwedge_{w\in\Sigma^{\leq k+1}} \{Lw^{-1}, (Lw^{-1})^c\} \ ,$$

where equality † holds with $X^k \overset{\text{def}}{=} \bigwedge_{w\in\Sigma^k}\{Lw^{-1}, (Lw^{-1})^c\}$. We conclude that $P_{\sim_L^r} = \text{gfp}(\lambda X. \bigwedge_{a\in\Sigma, B\in X} \{Ba^{-1}, (Ba^{-1})^c\} \curlywedge \{L, L^c\})$.

Theorem 3.5.25. *Let $\mathcal{D}$ be a DFA and M be Moore's DFA for $\mathcal{L}(\mathcal{D})$ as in Definition 3.5.20. Then, M is isomorphic to $\mathsf{Min}^r(\mathcal{L}(\mathcal{D}))$.*

Proof. Let $\mathcal{D} = (Q', \Sigma, \delta', I', F')$. Recall that Moore's minimal DFA is defined as $M = (Q, \Sigma, \delta, I, F)$ where the set of states corresponds to Moore's state-partition w.r.t. $\mathcal{D}$, i.e., $Q = \mathcal{Q}^{\mathcal{D}}$; $I = \{\mathcal{Q}^{\mathcal{D}}(q) \mid q \in I'\}$; $F = \{\mathcal{Q}^{\mathcal{D}}(q) \mid q \in F'\}$ and $S' = \delta(S, a)$ iff $\exists q \in S, q' \in S' : q' = \delta'(q, a)$, for each $S, S' \in Q$ and $a \in \Sigma$. Let $\mathsf{Min}^r(\mathcal{L}(\mathcal{D})) = (\widetilde{Q}, \Sigma, \widetilde{\delta}, \widetilde{I}, \widetilde{F})$ be described as in Definition 3.3.6. Finally, let L denote $\mathcal{L}(\mathcal{D})$, for simplicity. By Theorem 3.5.23, the mapping $\varphi : \wp(Q') \to \wp(\Sigma^*)$ defined as $\varphi(S) = W^{\mathcal{D}}_{I',S}$, for each $S \in \mathcal{Q}^{\mathcal{D}}$, is a partition isomorphism between $\mathcal{Q}^{\mathcal{D}}$ and $P_{\sim_L^r}$. Note that, by construction of M, $W^M_{I,S} = W^{\mathcal{D}}_{I',S}$, for each $S \in \mathcal{Q}^{\mathcal{D}}$. Thus, the mapping $\psi : Q \to \widetilde{Q}$ defined as $\psi(S) = W^M_{I,S}$, for each $S \in Q$, is also a partition isomorphism between $\mathcal{Q}^{\mathcal{D}}$ and $P_{\sim_L^r}$.

We show that ψ is a DFA morphism between M and $\mathsf{Min}^r(L)$. Let us denote $P_{\sim_L^r}$ simply as P. The initial state I of M is mapped to $\psi(I) = W^M_{I,I} = P(\varepsilon)$, since $\varepsilon \in W^M_{I,I}$. Therefore, ψ maps the initial state of M with the initial state of $\mathsf{Min}^r(L)$. Note that each final state S in F is such that $S \subseteq F'$. Therefore, $\psi(S) = W^M_{I,S} = P(u)$ with $u \in L$, i.e., ψ maps each final state of M to a final state of $\mathsf{Min}^r(L)$.

We also have to show that $S' = \delta(S, a)$ iff $\psi(S') = \widetilde{\delta}(\psi(S), a)$, for all $S, S' \in Q$ and $a \in \Sigma$. Assume that $S' = \delta(S, a)$, for some $S, S' \in Q$ and $a \in \Sigma$. Therefore, there exists $q, q' \in Q'$ such that $q \in S, q' \in S'$ and $q' = \delta'(q, a)$. Then, $\psi(S) = W^M_{I,S}$ and $\psi(S') = W^M_{I,S'}$ and there exists $u \in W_{I,S}(M)$ such that $ua \in W^M_{I,S'}$ (recall that M is a DFA, and therefore complete). Then, $\psi(S) = P(u)$ and $\psi(S') =$

$P(ua)$. Since P is a partition induced by a right congruence then, using Lemma 2.5.2, $P(u)a \subseteq P(ua)$. Therefore, $\psi(S') = \widetilde{\delta}(\psi(S), a)$. Assume now that, $P(ua) = \widetilde{\delta}(P(u), a)$ for some $u \in \Sigma^*$ and $a \in \Sigma$. Consider $S \in Q$ such that $\psi(S) = P(u)$, then u belongs to the left language of S, i.e., $u \in W^M_{I,S}$. Likewise, consider $S' \in Q$ such that $\psi(S') = P(ua)$, then $ua \in W^M_{I,S'}$. Therefore, there exists $q, q' \in Q'$ such that $q \in S, q' \in S'$ and $q' = \delta'(q, a)$. Thus, $S' = \delta(S, a)$.

4 Parikh Image of Pushdown Automata

In this chapter, we compare pushdown automata against context-free grammars and finite-state automata for the Parikh image of context-free languages.

4.1 Introduction

Now we turn our attention to Parikh equivalence over words. Unlike the congruences we proposed in the previous chapter, Parikh equivalence is an infinite-index congruence. Recall that two words u, v are Parikh-equivalent iff they have the same number of alphabet symbols regardless of the positions where the symbols appear in the words, i.e., u and v have the same Parikh image, or formally, $\lbag u \rbag = \lbag v \rbag$. Thus, it is easy to see that, given an alphabet Σ, the number of Parikh equivalence classes corresponds to the number of points in the space $\mathbb{N}^{|\Sigma|}$. On the other hand, Parikh equivalence is both, a right and a left congruence. Namely, given two Parikh-equivalent words u and v then $\lbag au \rbag = \lbag av \rbag$ and $\lbag ua \rbag = \lbag va \rbag$, for all $a \in \Sigma$.

Despite of not being an appropriate congruence for the construction of finite-state automata, Parikh equivalence offers an interesting angle to compare the conciseness of finite representations of the Parikh image of context-free languages in combination with the celebrated Parikh's Theorem [76].

The question about the conciseness of pushdown automata against context-free grammars for the representation of context-free languages was first addressed by Goldstine et al. [45] in a paper where they introduced an infinite family of context-free languages whose representation by a

pushdown automaton is more concise than by context-free grammars. In particular, they showed that each language of the family can be accepted by a pushdown automaton with $n \geq 1$ states and $p \geq 1$ stack symbols, but every context-free grammar needs at least $n^2p + 1$ variables if $n > 1$ (p if $n = 1$). Incidentally, the family shows that the translation procedure of a pushdown automaton into an equivalent context-free grammar that appears in textbooks [52], which uses the same large number of $n^2p + 1$ variables if $n > 1$ (p if $n = 1$), is optimal in the sense that there is no other algorithm that always produces fewer grammar variables.

We revisit this question but this time we turn our attention to Parikh equivalence. We define an infinite family of context-free languages as Goldstine et al. did but our family differs significantly from theirs. Given $n \geq 1$ and $k \geq 1$, each member of our family is given by a PDA with n states, $p = k + 2n + 4$ stack symbols and a single input symbol.[1] We show that, for each PDA of the family, every equivalent CFG has $\Omega(n^2(p - 2n - 4))$ variables.

First, we conclude that the textbook translation of a PDA into a language-equivalent context-free grammar is *optimal*[2] when the alphabet is unary. Note that if the alphabet is a singleton, there is no notion of ordering of the symbols in the words and the usual equality over words coincides with Parikh equivalence. Thus, we also conclude that the conversion algorithm is optimal for Parikh equivalence.

Furthermore, we investigate the special case of deterministic PDAs over a singleton alphabet for which equivalent context-free grammar representations of small size had been defined [23, 78]. We give a new construction for an equivalent context-free grammar given a unary DPDA that achieves the best known bounds [23] by combining two existing procedures.

Finally, Parikh's theorem [76] allows us to compare PDAs against finite state automata for Parikh-equivalent languages. First, we use the same family of PDAs to derive a lower bound on the number of states of every Parikh-equivalent NFA. The comparison becomes simple as its alphabet is unary and it accepts one single word. Second, using this lower bound we show that the 2-step procedure chaining existing constructions: (*i*) translate the PDA into a language-equivalent CFG [52]; and (*ii*) translate the CFG into a Parikh-equivalent NFA [32] yields *optimal*[3] results in the number of states of the resulting NFA.

[1] Their family has an alphabet of non-constant size.
[2] In a sense that we will precise in Section 4.3 (Remark 4.3.5).
[3] In a sense that we will precise in Section 4.4 (Remark 4.4.3).

As an independent contribution, we introduce a semantics of PDA runs as trees that we call *actrees*. The richer tree structure (compared to a sequence) makes simpler to compare each PDA of the family with its smallest grammar representation.

4.1.1 Notation

In this chapter, we will use the letter b to denote a symbol of a given alphabet Σ, instead of the letter a, as we did in Chapter 2. In this way, we avoid confusion with the notation used to denote the actions of an actree (see Section 4.2).

As defined in Chapter 2, an alphabet is *unary* iff it is a singleton. Thus, a language, or an automaton, is *unary* iff it is defined over a unary alphabet.

4.1.2 Disassembly and Assembly of Quasi-runs

We complement the definition of quasi-run of a pushdown automaton given in Chapter 2 with the notion of *disassembly* and *assembly* of quasi-runs. Broadly speaking, we will show that every quasi-run with more than one move can be *disassembled* into its first move and subsequent quasi-runs. Analogously, we will assemble a quasi-run from a given action and a list of quasi-runs. Apart from seeking completeness, these notions will be useful for the proof of Theorem 4.2.3.

To this end, we need to introduce a few auxiliary definitions. Given a word $w \in \Sigma^*$ and an integer i, define $w_{sh(i)} = (w)_{i+1} \cdots (w)_{i+|w|}$. Intuitively, w is shifted i positions to the left if $i \geq 0$ and to the right otherwise. So given $i \geq 0$, we will conveniently write $w_{\ll_i}$ for $w_{sh(i)}$ and $w_{\gg_i}$ for $w_{sh(-i)}$. Moreover, set $w_{\ll} = w_{\ll_1}$. For example, $a_{\ll_1} = a_{\gg_1} = \varepsilon$, $abcde_{\ll_3} = de$, $abcde_{\gg_3} = ab$, $w = (w)_1 \cdots (w)_i \, w_{\ll_i}$ and $w = w_{\gg_i} \, (w)_{|w|-i+1} \cdots (w)_{|w|}$ for $i > 0$.

Given an ID I and $i > 0$ define $I_{\gg_i} = (\mathrm{state}(I), \mathrm{stack}(I)_{\gg_i})$ which, intuitively, removes from I the i bottom stack symbols. The next results how to *disassemble* quasi-runs.

Lemma 4.1.1 (from [42])**.** *Let $r = I_0 \vdash \cdots \vdash I_n$, be a quasi-run. Then we can* disassemble *r into its first move $I_0 \vdash I_1$ and $d = |stack(I_1)|$ quasi-runs $r_1, \ldots, r_d$ each of which is such that*

$$r_i = (I_{p_{i-1}})_{\gg_{s_i}} \vdash \cdots \vdash (I_{p_i})_{\gg_{s_i}} \quad .$$

where $p_0 \leq p_1 \leq \cdots \leq p_d$ are defined to be the least positions such that $p_0 = 1$ and $stack(I_{p_i}) = stack(I_{p_{i-1}})_{\ll}$, for all i. Also $s_i = |stack(I_{p_i})|$, for all i, i.e., r_i is a quasi-run obtained by removing from the move sequence $I_{p_{i-1}} \vdash \cdots \vdash I_{p_i}$ the s_i bottom stack symbols leaving the stack of I_{p_i} empty and that of $I_{p_{i-1}}$ with one symbol only. Necessarily, $p_d = n$ and each quasi-run r_i starts with $(stack(I_1))_i$ as its initial content.

Example 4.1.2. Consider a PDA $\mathcal{P}$ with actions a_1 to a_5 respectively given by $(q_0, X_1) \hookrightarrow_\varepsilon (q_0, X_0\, X_0)$, $(q_0, X_0) \hookrightarrow_\varepsilon (q_1, X_1 \star)$, $(q_1, X_1) \hookrightarrow_\varepsilon (q_1, X_0\, X_0)$,
$(q_1, X_0) \hookrightarrow_b (q_1, \varepsilon)$ and $(q_1, \star) \hookrightarrow_\varepsilon (q_0, \varepsilon)$. Consider the following quasi-run:

$$r = (q_0, X_1) \vdash (q_0, X_0\, X_0) \vdash (q_1, X_1 \star X_0) \vdash (q_1, X_0\, X_0 \star X_0) \vdash (q_1, X_0 \star X_0) \vdash$$
$$(q_1, \star\, X_0) \vdash (q_0, X_0) \vdash (q_1, X_1 \star) \vdash (q_1, X_0\, X_0 \star) \vdash (q_1, X_0 \star) \vdash (q_1, \star) \vdash (q_0, \varepsilon) \ .$$

We can dissasemble r into its first move $I_0 \vdash I_1 = (q_0, X_1) \vdash (q_0, X_0\, X_0)$ and $d = 2$ quasi-runs r_1, r_2 such that:

$$\begin{aligned} r_1 &= (I_{p_0})_{\gg_{s_1}} \vdash^* (I_{p_1})_{\gg_{s_1}} \\ &= (I_1)_{\gg_1} \vdash^* (I_6)_{\gg_1} \\ &= (q_0, X_0) \vdash (q_1, X_1 \star) \vdash (q_1, X_0\, X_0 \star) \vdash (q_1, X_0 \star) \vdash (q_1, \star) \vdash (q_0, \varepsilon) \ , \end{aligned}$$

with $p_0 = 1, p_1 = 6, s_1 = |\mathrm{stack}(I_6)| = 1$.

$$\begin{aligned} r_2 &= (I_{p_1})_{\gg_{s_2}} \vdash^* (I_{p_2})_{\gg_{s_2}} \\ &= (I_6)_{\gg_0} \vdash (I_{11})_{\gg_0} \\ &= (q_0, X_0) \vdash (q_1, X_1 \star) \vdash (q_1, X_0\, X_0 \star) \vdash (q_1, X_0 \star) \vdash (q_1, \star) \vdash (q_0, \varepsilon) \ , \end{aligned}$$

with $p_1 = 6, p_2 = 11, s_2 = |\mathrm{stack}(I_{11})| = 0$.

Note that for each quasi-run r_i $(i = 1, 2)$, the stack of $(I_{p_i})_{\gg_{s_i}}$ is empty and that of $(I_{p_{i-1}})_{\gg_{s_i}}$ contains one symbol only. Also, $p_d = p_2 = n = 11$ and each r_i starts with $(\mathrm{stack}(I_1))_i$ as its initial content. ◀

Now we show how to *assemble* a quasi-run from a given action and a list of quasi-runs. We need the following notation: given I and $w \in \Gamma^*$, define $I \bullet w = (\mathrm{state}(I), \mathrm{stack}(I)\, w)$. We skip the proof of this lemma, which follows from a simple induction on $d \geq 1$, the length of the word pushed by the given action.

Lemma 4.1.3. *Let $a = (q, X) \hookrightarrow (q', \beta_1 \ldots \beta_d)$ be an action and $r_1, \ldots, r_d$ be $d \geq 1$ quasi-runs with $r_i = I_0^i \vdash I_1^i \vdash \cdots \vdash I_{n_i}^i$, for all i, such that:*

1. *the first action of* r_i *pops* β_i *for every* i*, and*
2. *the target state of last action of* r_i *(a when* $i = 0$*) is the source state of first action of* r_{i+1} *for all* $i \in \{1, \dots, d-1\}$ *(when* $d > 1$*).*

Then there exists a quasi-run r *given by:*

$$\begin{aligned}(q, X) \vdash (q', \beta_1 \dots \beta_d) &\vdash (I_1^1 \bullet \beta_2 \dots \beta_d) \vdash \cdots \vdash (I_{n_1}^1 \bullet \beta_2 \dots \beta_d) \vdash \cdots \\ &\vdash (I_1^\ell \bullet \beta_{\ell+1} \dots \beta_d) \vdash \cdots \vdash (I_{n_\ell}^\ell \bullet \beta_{\ell+1} \dots \beta_d) \vdash \cdots \\ &\vdash (I_1^d \bullet \varepsilon) \vdash \cdots \vdash (I_{n_d}^d \bullet \varepsilon) \ , \end{aligned} \tag{4.1}$$

with $1 < \ell < d$.

4.2 A Tree-Based Semantics for Pushdown Automata

In this section we introduce a tree-based semantics for PDA. Using trees instead of sequences sheds the light on key properties needed to present our main results.

Definition 4.2.1. Given a PDA $\mathcal{P}$, an action-tree (or *actree* for short) is a labeled tree $a(a_1(\dots), \dots, a_d(\dots))$ where a is an action of $\mathcal{P}$ pushing β with $|\beta| = d$ and each children $a_i(\dots)$ is an actree such that a_i pops $(\beta)_i$ for all i. Furthermore, an actree τ must satisfy that the source state of $(\bar{t})_{i+1}$ and the target state of $(\bar{t})_i$ coincide for every i.

An actree τ consumes an input resulting from replacing each action in the sequence $\bar{t}$ by the symbol it consumes (or ε, if the action does not consume any). An actree $a(\dots)$ is *accepting* if the initial ID enables a.

Example 4.2.2. Recall the PDA $\mathcal{P}$ described in Example 4.1.2. The reader can check that the actree $\tau = a_1(a_2(a_3(a_4, a_4), a_5), a_2(a_3(a_4, a_4), a_5))$, depicted in Figure 4.1, satisfies the conditions of Definition 4.2.1 where the sequence $\overline{\tau}$ is $a_1\, a_2\, a_3\, a_4\, a_4\, a_5\, a_2\, a_3\, a_4\, a_4\, a_5$, $|\tau| = 11$ and the input consumed is b^4.

Recall the Definition 2.4.4 of dimension of a labeled tree. The annotation $\overset{d(\tau)}{\tau}(\dots)$ shows that the actree of Example 4.2.2 has dimension 2:

$$\overset{2}{a_1}\,(\overset{1}{a_2}\,(\overset{1}{a_3}\,(\overset{0}{a_4}, \overset{0}{a_4}), \overset{0}{a_5}), \overset{1}{a_2}\,(\overset{1}{a_3}\,(\overset{0}{a_4}, \overset{0}{a_4}), \overset{0}{a_5}))\ .$$

◀

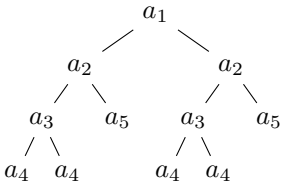


Figure 4.1: Depiction of the tree $a_1(a_2(a_3(a_4, a_4), a_5), a_2(a_3(a_4, a_4), a_5))$.

The proof of the following result is deferred to Section 4.5

Theorem 4.2.3. *Given a PDA, its actrees and quasi-runs are in a one-to-one correspondence.*

4.3 Parikh-Equivalent Context-Free Grammars

In this section we compare PDAs against CFGs when they describe Parikh-equivalent languages. We first study the general class of (nondeterministic) PDAs and, in Section 4.3.2, we look into the special case of unary deterministic PDAs.

We prove that, for every $n \geq 1$ and $p \geq 2n + 4$, there exists a PDA with n states and p stack symbols for which every Parikh-equivalent CFG has $\Omega(n^2(p - 2n - 4))$ variables. To this aim, we present a family of PDAs $\mathcal{P}(n, k)$ where $n \geq 1$ and $k \geq 1$. Each member has n states and $k + 2n + 4$ stack symbols, and accepts one single word over a unary alphabet.

4.3.1 The Family $\mathcal{P}(n, k)$ of PDAs

Definition 4.3.1. Given natural values $n \geq 1$ and $k \geq 1$, define the PDA $\mathcal{P}(n, k)$ with states $Q = \{q_i \mid 0 \leq i \leq n - 1\}$, input alphabet $\Sigma = \{b\}$, stack alphabet $\Gamma = \{S, \star, \$\} \cup \{X_i \mid 0 \leq i \leq k\} \cup \{s_i \mid 0 \leq i \leq n-1\} \cup \{r_i \mid$

$0 \leq i \leq n-1\}$, initial state q_0, initial stack symbol S and actions δ:

$$
\begin{array}{rll}
(q_0, S) \hookrightarrow_b (q_0, X_k\, r_0) & & \\
(q_i, X_j) \hookrightarrow_b (q_i, X_{j-1}\, r_m\, s_i\, X_{j-1}\, r_m) & \forall\, i, m \in \{0, \ldots, n-1\}, & \\
& \forall\, j \in \{1, \ldots, k\}, & \\
(q_j, s_i) \hookrightarrow_b (q_i, \varepsilon) & \forall i, j \in \{0, \ldots, n-1\}, & \\
(q_i, r_i) \hookrightarrow_b (q_i, \varepsilon) & \forall i \in \{0, \ldots, n-1\}, & \\
(q_i, X_0) \hookrightarrow_b (q_i, X_k \star) & \forall i \in \{0, \ldots, n-1\}, & \\
(q_i, X_0) \hookrightarrow_b (q_{i+1}, X_k\, \$) & \forall i \in \{0, \ldots, n-2\}, & \\
(q_i, \star) \hookrightarrow_b (q_{i-1}, \varepsilon) & \forall i \in \{1, \ldots, n-1\}, & \\
(q_0, \$) \hookrightarrow_b (q_{n-1}, \varepsilon) & & \\
(q_{n-1}, X_0) \hookrightarrow_b (q_{n-1}, \varepsilon)\ . & &
\end{array}
$$

Lemma 4.3.2. *Given $n \geq 1$ and $k \geq 1$, $\mathcal{P}(n,k)$ has a single accepting actree consuming input b^N where $N \geq 2^{n^2\, k}$.*

Proof. Fix values n and k and refer to the member of the family $\mathcal{P}(n,k)$ as $\mathcal{P}$. We show that $\mathcal{P}$ has exactly one accepting actree. We define a witness labeled tree τ inductively on the structure of the tree. Later we will prove that the induction is finite. First, we show how to construct the root and its children subtrees. This corresponds to case 1 below. Then, each non-leaf subtree is defined inductively in cases 2 to 5. Note that each non-leaf subtree of τ falls into one (and only one) of the cases. In fact, all cases are disjoint, in particular 2, 4 and 5. The reverse is also true: all cases describe a non-leaf subtree that does occur in τ. Finally, we show that each case describes *uniquely* how to build the next layer of children subtrees of a given non-leaf subtree.

1. $\tau = a(a_1(\ldots), a_2)$ where $a = (q_0, S) \hookrightarrow_b (q_0, X_k\, r_0)$ and $a_1(\ldots)$ and a_2 are of the form:

   $$
   \begin{array}{ll}
   a_2 = (q_0, r_0) \hookrightarrow_b (q_0, \varepsilon) & \text{only action popping } r_0 \\
   a_1 = (q_0, X_k) \hookrightarrow_b (q_0, X_{k-1}\, r_0\, s_0\, X_{k-1}\, r_0) & \text{only way to enable } a_2.
   \end{array}
   $$

   Note that the initial ID (q_0, S) enables a which is the only action of $\mathcal{P}$ with this property. Note also that $\overset{d}{a}\ (\overset{d}{a_1}\ (\ldots), \overset{0}{a_2})$ holds, where $d > 0$.

2. Each subtree whose root is labeled:

$$a = (q_i, X_j) \hookrightarrow_b (q_i, X_{j-1}\, r_m\, s_i\, X_{j-1}\, r_m)$$

with $i, m \in \{0, \ldots, n-1\}$ and $j \in \{2, \ldots, k\}$ is of the form $a(a_1(\ldots), a_2, a_3, a_1(\ldots), a_2)$ where:

$a_2 = (q_m, r_m) \hookrightarrow_b (q_m, \varepsilon)$ only action popping r_m

$a_3 = (q_m, s_i) \hookrightarrow_b (q_i, \varepsilon)$ only action popping s_i from q_m

$a_1 = (q_i, X_{j-1}) \hookrightarrow_b (q_i, X_{j-2}\, r_m\, s_i\, X_{j-2}\, r_m)$ only way to enable a_2.

Assume for now that τ is unique. Therefore, as the 1^{st} and 4^{th} child of a share the same label a_1, they also root the same subtree. Thus, it holds $(d > 0)$

$$\overset{d+1}{a}\ (\overset{d}{a_1}\ (\ldots), \overset{0}{a_2}, \overset{0}{a_3}, \overset{d}{a_1}\ (\ldots), \overset{0}{a_2})\ .$$

3. Each subtree whose root is labeled $a = (q_i, X_0) \hookrightarrow_b (q_{i+1}, X_k\, \$)$ with $i \in \{0, \ldots, n-2\}$ has the form $a(a_1(\ldots), a_2)$ where

$a_2 = (q_0, \$) \hookrightarrow_b (q_{n-1}, \varepsilon)$ only action popping \$

$a_1 = (q_{i+1}, X_k) \hookrightarrow_b (q_{i+1}, X_{k-1}\, r_0\, s_{i+1}\, X_{k-1}\, r_0)$ only way to enable a_2.

Note that $\overset{d}{a}\ (\overset{d}{a_1}\ (\ldots), \overset{0}{a_2})$ holds, where $d > 0$.

4. Each subtree whose root is labeled $a = (q_i, X_1) \hookrightarrow_b (q_i, X_0\, r_m\, s_i\, X_0\, r_m)$ with $i \in \{0, \ldots, n-1\}$ and $m \in \{0, \ldots, n-2\}$ has the form

$$a(a_1(a_{11}(\ldots), a_{12}), a_2, a_3, a_1(a_{11}(\ldots), a_{12}), a_2)\ .$$

where

$a_2 = (q_m, r_m) \hookrightarrow_b (q_m, \varepsilon)$ only action popping r_m

$a_3 = (q_m, s_i) \hookrightarrow_b (q_i, \varepsilon)$ only action popping s_i from q_m

$a_1 = (q_i, X_0) \hookrightarrow_b (q_i, X_k \star)$ assume it for now

$a_{12} = (q_{m+1}, \star) \hookrightarrow_b (q_m, \varepsilon)$ only way to enable a_2

$a_{11} = (q_i, X_k) \hookrightarrow_b (q_i, X_{k-1}\, r_{m+1}\, s_i\, X_{k-1}\, r_{m+1})$ only way to enable a_{12}.

Assume a_1 is given by the action $(q_i, X_0) \hookrightarrow_b (q_{i+1}, X_k\,\$)$ instead. Then following the action popping \$, we would end up in the state q_{n-1}, not enabling a_2 since $m < n-1$.
Again, assume for now that τ is unique. Hence, as the 1st and 4th child of a are both labeled by a_1, they root the same subtree. Thus, it holds ($d > 0$)

$$\overset{d+1}{a}\ (\overset{d}{a_1}\ (\overset{d}{a_{11}}\ (\ldots), \overset{0}{a_{12}}), \overset{0}{a_2}, \overset{0}{a_3}, \overset{d}{a_1}\ (\overset{d}{a_{11}}\ (\ldots), \overset{0}{a_{12}}), \overset{0}{a_2})\ .$$

5. Each subtree whose root is labeled

$$a = (q_i, X_1) \hookrightarrow_b (q_i, X_0\, r_{n-1}\, s_i\, X_0\, r_{n-1})$$

with $i \in \{0, \ldots, n-1\}$ has the form $a(a_1(\ldots), a_2, a_3, a_1(\ldots), a_2)$ where

$$a_2 = (q_{n-1}, r_{n-1}) \hookrightarrow_b (q_{n-1}, \varepsilon) \qquad \text{only action popping } r_{n-1}$$

$$a_3 = (q_{n-1}, s_i) \hookrightarrow_b (q_i, \varepsilon) \qquad \text{only action popping } s_i \text{ from } q_i$$

$$a_1 = \begin{cases} (q_i, X_0) \hookrightarrow_b (q_{i+1}, X_k\,\$) & \text{if } i < n-1 \\ (q_{n-1}, X_0) \hookrightarrow_b (q_{n-1}, \varepsilon) & \text{otherwise} \end{cases} \qquad \text{assume it for now.}$$

For both cases ($i < n-1$ and $i = n-1$), assume a_1 is given by $(q_i, X_0) \hookrightarrow_b (q_i, X_k\,\star)$ instead. Then, the action popping $\star$ must end up in the state q_{n-1} in order to enable a_2, i.e., it must be of the form $(q_n, \star) \hookrightarrow_b (q_{n-1}, \varepsilon)$. Hence the action popping X_k must be of the form $(q_i, X_k) \hookrightarrow_b (q_i, X_{k-1}\, r_m\, s_i\, X_{k-1}\, r_m)$ where necessarily $m = n$, a contradiction (the stack symbol r_n is not defined in $\mathcal{P}$).
Assume for now that τ is unique. Then, as the 1st and 4th child of a are labeled by a_1, they root the same subtree (possibly a leaf). Thus, it holds ($d \geq 0$)

$$\overset{d+1}{a}\ (\overset{d}{a_1}\ (\ldots), \overset{0}{a_2}, \overset{0}{a_3}, \overset{d}{a_1}\ (\ldots), \overset{0}{a_2})\ .$$

We now prove that τ is finite by contradiction. Suppose τ is an infinite tree. König's Lemma shows that τ has thus at least one infinite path, say p, from the root. As the set of labels of τ is finite

then some label must repeat infinitely often along p. Let us define a strict partial order between the labels of the non-leaf subtrees of τ. We restrict to the non-leaf subtrees because no infinite path contains a leaf subtree. Let $a_1(\ldots)$ and $a_2(\ldots)$ be two non-leaf subtrees of τ. Let q_{i_1} be the source state of a_1 and q_{f_1} be the target state of the last action in the sequence $\overline{a_1(\ldots)}$. Define q_{i_2}, q_{f_2} similarly for $a_2(\ldots)$. Let X_{j_1} be the symbol that a_1 pops and X_{j_2} be the symbol that a_2 pops. Define $a_1 \prec a_2$ iff (a) either $i_1 < i_2$, (b) or $i_1 = i_2$ and $f_1 < f_2$, (c) or $i_1 = i_2, f_1 = f_2$ and $j_1 > j_2$. First, note that the label a of the root of τ (case 1) only occurs in the root as there is no action of $\mathcal{P}$ pushing S. Second, relying on cases 2 to 5, we observe that every pair of non-leaf subtrees $a_1(\ldots)$ and $a_2(\ldots)$ (excluding the root) such that $a_1(\ldots)$ is the parent node of $a_2(\ldots)$ verifies $a_1(\ldots) \prec a_2(\ldots)$. Using the transitive property of the strict partial order $\prec$, we conclude that everypair of subtrees $a_1(\ldots)$ and $a_2(\ldots)$ in p such that $a_1(\ldots a_2(\ldots)\ldots)$ verifies $a_1(\ldots) \prec a_2(\ldots)$. Therefore, no repeated variable can occur in p (contradiction). We conclude that τ is finite.

The reader can observe that $\tau = a(\ldots)$ verifies all conditions of the definition of actree (Definition 4.2.1) and the initial ID enables a, thus it is an accepting actree of $\mathcal{P}$. Since we also showed that no other tree can be defined using the actions of $\mathcal{P}$, τ is unique.

Finally, we give a lower bound on the length of the word consumed by τ. To this aim, we prove that $d(t) = n^2\,k$. Then since all actions consume input symbol b, Lemma 2.4.6 shows that the word b^N consumed is such that $N \geq 2^{n^2\,k}$.

Note that, if a subtree of τ verifies case 1 or 3, its dimension remains the same w.r.t. its children subtrees. Otherwise, the dimension always grows. Recall that all cases from 1 to 5 describe a set of labels that does occur in τ. Also, as τ is unique, no path from the root to a leaf repeats a label. Thus, to compute the dimension of τ is enough to count the number of distinct labels of τ that are included in cases 2, 4 and 5, which is equivalent to compute the size of the set

$$D = \{(q_i, X_j) \hookrightarrow (q_i, X_{j-1}\, r_m\, s_i\, X_{j-1}\, r_m) \mid 1 \leq j \leq k,\, 0 \leq i, m \leq n{-}1\} \ .$$

Clearly $|D| = n^2\,k$ from which we conclude that $d(t) = n^2\,k$. Hence,

$|t| \geq 2^{n^2\,k}$, and therefore τ consumes a word b^N where $N \geq 2^{n^2\,k}$ since each action of τ consumes a b.

Let us give an example of a member of the family $\mathcal{P}(n,k)$.

Example 4.3.3. We give a graphical depiction of the accepting actree τ of $\mathcal{P}(2,1)$ in Figure 4.2 at the end of the chapter. Recall that $\mathcal{P}(2,1)$ corresponds to the member of the family $\mathcal{P}(n,k)$ that has 2 states q_0 and q_1, and 9 stack symbols $S, X_0, X_1, s_0, s_1, r_0, r_1, \star$ and \$. ◀

Theorem 4.3.4. *For each $n \geq 1$ and $p > 2n+4$, there is a PDA with n states and p stack symbols for which every Parikh-equivalent CFG has $\Omega(n^2(p-2n-4))$ variables.*

Proof. Consider the family of PDAs $\mathcal{P}(n,k)$ with $n \geq 1$ and $k \geq 1$ described in Definition 4.3.1. Fix n and k and refer to the corresponding member of the family as $\mathcal{P}$.

First, Lemma 4.3.2 shows that $L(\mathcal{P})$ consists of a single word b^N with $N \geq 2^{n^2\,k}$. It follows that a language L is Parikh-equivalent to $L(\mathcal{P})$ iff L is language-equivalent to $L(\mathcal{P})$.

Let $\mathcal{G}$ be a CFG such that $L(\mathcal{G}) = L(\mathcal{P})$. The smallest CFG that generates exactly one word of length ℓ has size $\Omega(log(\ell))$ [21, Lemma 1], where the size of a grammar is the sum of the length of all the rules. It follows that $\mathcal{G}$ is of size $\Omega(log(2^{n^2k})) = \Omega(n^2k)$. As $k = p-2n-4$, then $\mathcal{G}$ has size $\Omega(n^2(p-2n-4))$. We conclude that $\mathcal{G}$ has $\Omega(n^2\,(p-2n-4))$ variables.

Remark 4.3.5. According to the classical conversion algorithm, every CFG that is equivalent to $\mathcal{P}(n,k)$ needs at most $n^2(k+2n+4) \in \mathcal{O}(n^2k+n^3)$ variables. On the other hand, Theorem 4.3.4 shows that a lower bound for the number of variables is $\Omega(n^2k)$. We observe that, as long as $n \leq Ck$ for some positive constant C, the family $\mathcal{P}(n,k)$ shows that the conversion algorithm is optimal[4] in the number of variables when assuming both language and Parikh equivalence. Otherwise, the algorithm is not optimal as there exists a gap between the lower bound and the upper bound. For instance, if $n = k^2$ then the upper bound is $\mathcal{O}(k^5+k^6) = \mathcal{O}(k^6)$ while the lower bound is $\Omega(k^5)$.

[4]Note that if $n \leq Ck$ for some $C > 0$ then the n^3 addend in $\mathcal{O}(n^2k+n^3)$ becomes negligible compared to n^2k, and the lower and upper bound coincide.

4.3.2 The Case of Unary Deterministic Pushdown Automata

We have seen that the classical translation from PDA to CFG is optimal in the number of grammar variables for the family of unary nondeterministic PDA $\mathcal{P}(n,k)$ when n is in linear relation with respect to k. However, for unary *deterministic* PDA (UDPDA, for short) the situation is different. Pighizzini [78, Theorem 4] shows that for every UDPDA accepting by final states with n states and p stack symbols, there exists an equivalent CFG with at most $2np$ variables. Although he gives a definition of such a grammar, we were not able to extract an algorithm from it. On the other hand, Chistikov and Majumdar [23, Lemma 4] give a polynomial time algorithm that transforms a UDPDA accepting by final states into an equivalent CFG going through the construction of a pair of straight-line programs[5]. The size of the resulting CFG is linear in that of the UDPDA.[6]

We propose a new polynomial time algorithm that converts a UDPDA with n states and p stack symbols into an equivalent CFG with $\mathcal{O}(np)$ variables. Our algorithm is based on the observation that the conversion algorithm from PDAs to CFGs need not consider all the triples in (2.1). We discard unnecessary triples using the *saturation procedure* [13, 35] that computes the set of reachable IDs.

For a given PDA $\mathcal{P}$ with $q \in Q$ and $X \in \Gamma$, define the set of *reachable IDs* $R_{\mathcal{P}}(q,X)$ as follows:

$$R_{\mathcal{P}}(q,X) \stackrel{\text{def}}{=} \{(q',\beta) \mid \exists (q,X) \vdash \cdots \vdash (q',\beta)\} \ .$$

Lemma 4.3.6. *If $\mathcal{P}$ is a UDPDA then the set $\{I \in R_{\mathcal{P}}(q,X) \mid stack(I) = \varepsilon\}$ has at most one element for every state q and stack symbol X.*

> ***Proof.*** Let $\mathcal{P}$ be a UDPDA with $\Sigma = \{a\}$. Since $\mathcal{P}$ is *deterministic* we have that (i) for every $q \in Q, X \in \Gamma$ and $b \in \Sigma \cup \{\varepsilon\}$, $|\delta(q,b,X)| \leq 1$ and, (ii) for every $q \in Q$ and $X \in \Gamma$, if $\delta(q,\varepsilon,X) \neq \varnothing$ then

[5] An *straight-line program* over an alphabet Σ is a context-free grammar generating one single word over Σ. The combination of the two straight-line programs constructed by Chistikov and Majumdar [23] represent the unary language of the input UDPDA.

[6] Chistikov and Majumdar [23] define the size of a CFG as its number of variables when the grammar is given in Chomsky normal form. They define the size of a unary pushdown automaton as the product of number of states and the number of stack symbols.

$\delta(q, b, X) = \varnothing$ for every $b \in \Sigma$.

The proof goes by contradiction. Assume that for some state q and stack symbol X, there are two IDs I_1 and I_2 in $R_{\mathcal{P}}(q, X)$ such that $\text{stack}(I_1) = \text{stack}(I_2) = \varepsilon$ and $\text{state}(I_1) \neq \text{state}(I_2)$.

Necessarily, there exist three IDs J, J_1 and J_2 with $J_1 \neq J_2$ such that the following holds:

$$(q, X) \vdash \cdots \vdash J \vdash_a J_1 \vdash \cdots \vdash I_1$$
$$(q, X) \vdash \cdots \vdash J \vdash_b J_2 \vdash \cdots \vdash I_2 \ .$$

It is routine to check that if $a = b$ then $\mathcal{P}$ is not deterministic, a contradiction. Next, we consider the case $a \neq b$. When a and b are symbols, because $\mathcal{P}$ is a unary DPDA, then they are the same, a contradiction. Else, if either a or b is ε, then $\mathcal{P}$ is not deterministic, a contradiction. We conclude from the previous that when $\text{stack}(I_1) = \text{stack}(I_2) = \varepsilon$, then necessarily $\text{state}(I_1) = \text{state}(I_2)$, and therefore the set $\{I \in R_{\mathcal{P}}(q, X) \mid \text{stack}(I) = \varepsilon\}$ has at most one element.

Intuitively, Lemma 4.3.6 shows that, when fixing q and X, there is at most one q' such that the triple $[qXq']$ generates a string of terminals. We use this fact to prove the following theorem.

Theorem 4.3.7. *For every UDPDA with n states and p stack symbols, there is a polynomial time algorithm that computes an equivalent CFG with at most np variables.*

Proof. The conversion algorithm translating a PDA $\mathcal{P}$ to a CFG G computes the set of grammar variables $\{[qXq'] \mid q, q' \in Q, X \in \Gamma\}$. By Lemma 4.3.6, for each q and X there is at most one variable $[qXq']$ in the previous set generating a string of terminals. The consequence of the lemma is two-fold: (*i*) For the triples it suffices to compute the subset of the aforementioned generating variables, which we denote by T in the sequel. Clearly, $|T| \leq np$. (*ii*) Each action of $\mathcal{P}$ now yields a single rule in G. This is because, according to the definition of the set of productions in (2.2) (see Definition 2.4.9), there is at most one choice for r_1 to r_d, hence we avoid the exponential blowup of the runtime in the conversion algorithm. To compute T given $\mathcal{P}$, we use the polynomial time saturation procedure [13, 35] which given (q, X) computes a NFA for the set $R_{\mathcal{P}}(q, X)$. Then we compute from this

set the unique state q' (if any) such that $(q', \varepsilon) \in R_{\mathcal{P}}(q, X)$, hence T. From the above we find that, given $\mathcal{P}$, we compute G in polynomial time.

Up to this point, we have assumed the empty stack as the acceptance condition. For general PDA, assuming final states or empty stack as acceptance condition induces no loss of generality. The situation is different for deterministic PDAs where accepting by final states is more general than empty stack (see Remark 2.3.9). For this reason, we contemplate the case where the UDPDA accepts by final states. Theorem 4.3.8 shows how our previous construction can be modified to accommodate the acceptance condition by final states.

Theorem 4.3.8. *For every UDPDA with n states and p stack symbols that accepts by final states, there is a polynomial time algorithm that computes an equivalent CFG with $\mathcal{O}(np)$ variables.*

Proof. Let $\mathcal{P}$ be a UDPDA with n states and p stack symbols that accepts by final states. We first translate $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$[a] into a (possibly nondeterministic) unary pushdown automaton $\mathcal{P}' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0')$ with an empty stack acceptance condition [52]. In particular, $Q' = Q \cup \{q_0', \mathit{sink}\}$; $\Gamma' = \Gamma \cup \{Z_0'\}$; and δ' is given by:

$$\begin{aligned}
&\delta \cup \{(q_0', Z_0') \hookrightarrow_\varepsilon (q_0, Z_0\, Z_0')\} \\
&\quad \cup \{(q, X) \hookrightarrow_\varepsilon (\mathit{sink}, X) \mid X \in \Gamma', q \in F\} \\
&\quad \cup \{(\mathit{sink}, X) \hookrightarrow_\varepsilon (\mathit{sink}, \varepsilon) \mid X \in \Gamma'\} \ .
\end{aligned}$$

The new stack symbol Z_0' is to prevent $\mathcal{P}'$ from incorrectly accepting when $\mathcal{P}$ is in a nonfinal state with an empty stack. The state *sink* is to empty the stack upon $\mathcal{P}$ entering a final state. Observe that $\mathcal{P}'$ need not be deterministic. Also, it is routine to check that $L(P') = L(P)$ (for a proof, see Theorem 6.11 in [52]) and $\mathcal{P}'$ is computable in time linear in the size of $\mathcal{P}$. Now let us turn to $R_{P'}(q, X)$. For $\mathcal{P}'$ a weaker version of Lemma 4.3.6 holds: the set $H = \{I \in R_{P'}(q, X) \mid \mathrm{stack}(I) = \varepsilon\}$ has at most two elements for every state $q \in Q'$ and stack symbols $X \in \Gamma'$. This is because if H contains two IDs then necessarily one of them has *sink* for state.

Based on this result, we construct T as in Theorem 4.3.7, but this time we have that $|T|$ is $\mathcal{O}(np)$.

Now we turn to the set of production rules as defined in (2.2) (see Definition 2.4.9). We show that each action $(q, X) \hookrightarrow_b (q', \beta)$ of $\mathcal{P}'$ yields at most d production rules in G where $d = |\beta|$. For each state r_i in (2.2) we have two choices, one of which is *sink*. We also know that once a move sequence enters *sink* it cannot leave it. Therefore, we have that if $r_i = sink$ then $r_{i+1} = \cdots = r_d = sink$. Given an action, it thus yields d production rules one where $r_1 = \cdots = r_d = sink$, another where $r_2 = \cdots = r_d = sink$, ..., etc. Hence, we avoid the exponential blowup of the runtime in the conversion algorithm.

The remainder of the proof follows that of Theorem 4.3.7.

[a] Recall that the set of final states is given by a subset F of Q (see Remark 2.3.7).

4.4 Parikh-Equivalent Finite-State Automata

Parikh's theorem [76] shows that every context-free language is Parikh-equivalent to a regular language. Using this result, we can compare PDAs against NFAs under Parikh equivalence. We start by deriving a lower bound using the family $\mathcal{P}(n, k)$. Because its alphabet is unary and it accepts a single long word, the comparison becomes straightforward.

Theorem 4.4.1. *For each $n \geq 1$ and $p > 2n + 4$, there is a PDA with n states and p stack symbols for which every Parikh-equivalent NFA has at least $2^{n^2(p-2n-4)} + 1$ states.*

Proof. Consider the family of PDAs $\mathcal{P}(n, k)$ with $n \geq 1$ and $k \geq 1$ described in Definition 4.3.1. Fix n and k and refer to the corresponding member of the family as $\mathcal{P}$. By Lemma 4.3.2, $L(\mathcal{P}) = \{b^N\}$ with $N \geq 2^{n^2 k}$. Then, the smallest NFA that is Parikh-equivalent to $L(\mathcal{P})$ needs $N + 1$ states. As $k = p - 2n - 4$, we conclude that the smallest Parikh-equivalent NFA has at least $2^{n^2(p-2n-4)} + 1$ states.

Let us now turn to upper bounds. We give a 2-step procedure that, given a PDA, computes a Parikh-equivalent NFA. The steps are: (*i*) translate the PDA into a language-equivalent context-free grammar [52]; and (*ii*) translate the context-free grammar into a Parikh-equivalent finite state automaton [32].

First, let us introduce the following definition. A grammar is in *2-1 normal form* (2-1-NF, for short) if each rule $(X, \alpha) \in R$ is such that α

consists of at most one terminal and at most two variables. It is worth pointing that, when the grammar is in 2-1-NF, the resulting Parikh-equivalent NFA from step (*ii*) has $\mathcal{O}(4^n)$ states where n is the number of grammar variables [32]. For the sake of simplicity, we will assume that grammars are in 2-1-NF which holds when PDAs are in *reduced form*: every move is of the form $(q, X) \hookrightarrow_b (q', \beta)$ with $|\beta| \leq 2$ and $b \in \Sigma \cup \{\varepsilon\}$.

Theorem 4.4.2. *Given a PDA in reduced form with $n \geq 1$ states and $p \geq 1$ stack symbols, there is a Parikh-equivalent NFA with $\mathcal{O}(4^{n^2 p})$ states.*

> ***Proof.*** The algorithm to convert a PDA with $n \geq 1$ states and $p \geq 1$ stack symbols into a CFG that generates the same language [52] uses at most $n^2 p + 1$ variables if $n > 1$ (or p if $n = 1$). Given a CFG of n variables in 2-1-NF, one can construct a Parikh-equivalent NFA with $\mathcal{O}(4^n)$ states [32].
>
> Given a PDA $\mathcal{P}$ with $n \geq 1$ states and $p \geq 1$ stack symbols the conversion algorithm returns a language-equivalent CFG G. Note that if $\mathcal{P}$ is in reduced form, then the conversion algorithm returns a CFG in 2-1-NF. Then, apply to G the known construction that builds a Parikh-equivalent NFA [32]. The resulting NFA has $\mathcal{O}(4^{n^2 p})$ states.

Remark 4.4.3. Theorem 4.4.1 shows that every NFA that is Parikh-equivalent to $\mathcal{P}(n, k)$ needs $\Omega(2^{n^2 k})$ states. On the other hand, Theorem 4.4.2 shows that the number of states of every Parikh-equivalent NFA is $O(4^{n^2(k+2n+4)})$. Thus, our construction is *close to optimal*[7] when n is in linear relation w.r.t. k.

Finally, let us discuss the reduced form assumption. Its role is to simplify the exposition and, indeed, it is not needed to prove correctness of the 2-step procedure. The assumption can be relaxed and bounds can be inferred. They will contain an additional parameter related to the length of the longest sequence of symbols pushed on the stack.

4.5 Supplementary Proofs

We defer to this section the proof of Theorem 4.2.3.

[7] As the blow up of our construction is $O(4^{n^2(k+2n+4)})$ for a lower bound of $2^{n^2 k}$, we say that it is *close to optimal* in the sense that $2n^2(k + 2n + 4) \in \Theta(n^2 k)$, which holds when n is in linear relation with respect to k (see Remark 4.3.5).

Theorem 4.2.3. *Given a PDA, its actrees and quasi-runs are in a one-to-one correspondence.*

Proof. To prove the existence of a one-to-one correspondence we show that:

1. Each quasi-run must be paired with at least one actree, and viceversa.
2. No quasi-run may be paired with more than one actree, and viceversa.

1. First, given a quasi-run r of $\mathcal{P}$, define a tree τ inductively on the length of r. We prove at the same time that (4.2) holds for τ, which we show is an actree.

$$\overline{t} \text{ and the sequence of actions of } r \text{ coincide.} \tag{4.2}$$

- *Base case:* In this case, necessarily $r = I_0 \vdash I_1$ and we define τ as the leaf labeled by the action $I_0 \hookrightarrow I_1$. Clearly, τ satisfies (4.2), hence τ is an actree (τ trivially verifies Definition 4.2.1).

- *Inductive step:* Now consider the case $r = I_0 \vdash I_1 \vdash \cdots \vdash I_n$ where $n > 1$, we define τ as follows. Lemma 4.1.1 shows r disassembles into its first action a and $d = |\text{stack}(I_1)| \geq 1$ quasi-runs $r_1, \ldots, r_d$. The action a labels the root of τ which has d children τ_1 to τ_d. The subtrees τ_1 to τ_d are defined applying the induction hypothesis on the quasi-runs r_1 to r_d, respectively. From the induction hypothesis, τ_1 to τ_d are actrees and each sequence $\overline{t_i}$ coincide with the sequence of actions of the quasi-run r_i. Moreover, Lemma 4.1.1 shows that the state of the last ID of r_i coincides with the state of the first ID of r_{i+1} for all $i \in \{1, \ldots, d-1\}$. Also, the state of the first ID of r_1 coincides with the state of I_1. We conclude from above that τ satisfies (4.2) and that τ is an actree since it verifies Definition 4.2.1.

Second, given an actree τ of $\mathcal{P}$, we define a move sequence r inductively on the height of τ. We prove at the same time that (4.2) holds for r which we show is a quasi-run.

- *Base case:* In this case, we assume $h(t) = 0$. Then, the root

of τ is a leaf labeled by an action $a = I_0 \hookrightarrow I_1$ and we define $r = I_0 \vdash I_1$. Clearly, r satisfies (4.2) and is a quasi-run.

- *Base case:* Now, assume that τ has d children τ_1 to τ_d, we define r as follows. By the induction hypothesis, each subtree τ_i for all $i \in \{1, \ldots, d\}$ defines a quasi-run r_i verifying (4.2). The definition of actree shows that the root of τ pushes β_1 to β_d which are popped by its d children. By induction hypothesis each r_i for all $i \in \{1, \ldots, d\}$ thus starts by popping β_i. Next it follows from the induction hypothesis and the definition of actree that the target state of the action given by the last move of r_i coincides with the source state of the action given by the first move of r_{i+1} for all $i \in \{1, \ldots, d-1\}$. Moreover, the target state of a coincides with the source state of the action given by the first move of r_1. Thus, applying Lemma 4.1.3 to the action given by the root of τ and $r_1, \ldots, r_d$ yields the quasi-run r that satisfies (4.2) following our previous remarks.

2. First, we prove that no quasi-run may be paired with more than one actree. The proof goes by contradiction. Given a move sequence $I_0 \vdash \ldots \vdash I_n$, define its sequence of actions $a_1 \ldots a_n$ such that the move $I_i \vdash I_{i+1}$ is given by the action a_{i+1}, for all i. Note that two quasi-runs $r = I_0 \vdash \ldots \vdash I_n$ and $r' = I'_0 \vdash \ldots \vdash I'_m$ are *equal* iff their sequences of actions coincide.

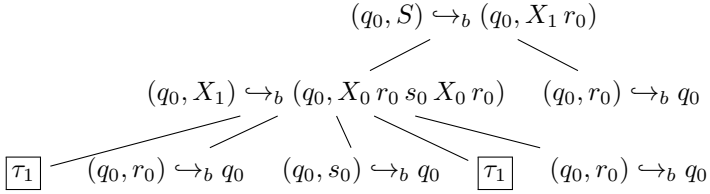
Suppose that given the actrees τ and τ' with $\tau \neq t'$, there exist two quasi-runs r and r' such that r is paired with τ and r' is paired with τ', under the relation we described in part 1. of this proof, and $r = r'$. Let $\overline{t} = a_1, \ldots, a_n$ and $\overline{t'} = a'_1, \ldots, a'_m$. Let $p \in \{1, \ldots, min(n, m)\}$ be the least position in both sequences such that $a_p \neq a'_p$. By (4.2), the sequences of actions of r and r' also differ at position p (at least). Thus, $r \neq r'$ (contradiction).

Second, we prove that no actree may be paired with more than one quasi-run. Again, we give a proof by contradiction.

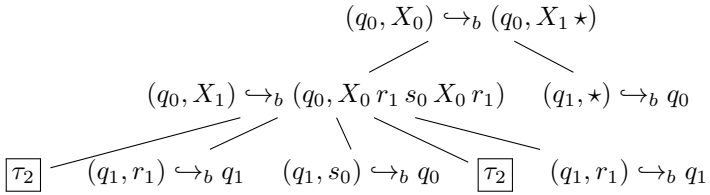
Suppose that given the quasi-runs r and r' with $r \neq r'$, there exist two actrees τ and τ' such that τ is paired with r and τ' is paired with r', under the relation we described in part 1. of the proof, and $\tau = t'$. We rely on the standard definition of equality between labeled trees.

Suppose $a_1 \ldots a_n$ is the sequence of actions of r and $a'_1 \ldots a'_m$ is

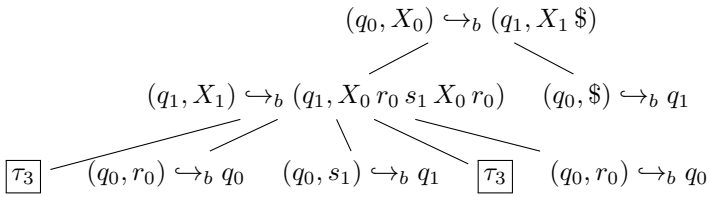
> the sequence of actions of r'. Let $p \in \{1, \ldots, min(n, m)\}$ be the least position such that $a_p \neq a'_p$. By (4.2), $\bar{t}$ and $\overline{t'}$ also differ at position p (at least). Then, $\tau \neq t'$ (contradiction).



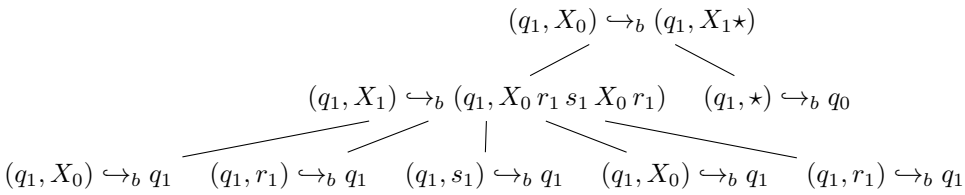
(a) Top of the tree τ



(b) Subtree τ_1



(c) Subtree τ_2



(d) Subtree τ_3

Figure 4.2: Accepting actree τ of $\mathcal{P}(2, 1)$. We have split the tree into 4 subtrees, and replaced actions of the form $(q_0, X) \hookrightarrow_b (q_1, \varepsilon)$ simply by $(q_0, X) \hookrightarrow_b q_1$.

5 Parikh Image of Weighted Context-Free Grammars

In this chapter, we address the problem of extending Parikh's Theorem to the weighted case. Since in this setting the result does not hold in general, we consider the so-called Parikh property and we study: (*i*) under which conditions the Parikh property holds, and (*ii*) whether the property is decidable or not.

5.1 Introduction

We continue working under the Parikh equivalence assumption, but this time we turn our attention to the weighted case.

Extending Parikh's Theorem to this setting consists in asking whether, for a given weighted pushdown automaton $\mathcal{P}$, there is a weighted finite-state automaton $\mathcal{A}$ that accepts a Parikh-equivalent language and such that for every word w, the sum of the weights of all words Parikh-equivalent to w in $\mathcal{P}$ coincides with that of all Parikh-equivalent words to w in $\mathcal{A}$.

This generalization has the potential of expanding the applications of this result on the analysis of multi-threaded asynchronous programs with procedures to systems where transitions are augmented with a weight that may represent the cost of performing the transition or the probability of an event associated to it. Finding a weighted finite-state automaton that is Parikh-equivalent to the original program and preserves the costs enables quantitative [22, 29, 83] and probabilistic analysis [5] of programs following this paradigm.

In this chapter, we will present our results using the grammar model (as

opposed to the automata model). Using weighted context-free grammars allows us to exploit their connection with algebraic systems of equations to give more simple and convincing proofs of our results.

In a WCFG, a weight is assigned to each rule of the grammar. The notion of weight is extended from rules to parse trees by multiplying the weights of the rules used along a tree, and from parse trees to words by adding the weights of all the possible parse trees that yield to a word. We say that two WCFGs $\mathcal{G}_1$ and $\mathcal{G}_2$ are Parikh-equivalent iff for each class of Parikh-equivalence, the sum of the weights of all its words coincide for $\mathcal{G}_1$ and $\mathcal{G}_2$.

We consider the following problem: given a WCFG $\mathcal{G}_1$, does there exists a Parikh-equivalent WCFG $\mathcal{G}_2$ that is regular? If the answer is positive we say that $\mathcal{G}_1$ satisfies the Parikh property.

It follows from a known counterexample by Petre [77] that the property is not true in general. Recently, Bhattiprolu et al. [10] further investigated this question. They show a class of WCFGs over the unary alphabet that always satisfy the Parikh property.

Now, we show that every *nonexpansive* WCFG (over an arbitrary alphabet and arbitrary semiring) satisfies the Parikh property. As we will see, a WCFG is nonexpansive if no grammar derivation is of the form $X \Rightarrow^* \alpha_0 X \alpha_1 X \alpha_2$, where X is a grammar variable and each α_i is a word of terminals and variables. Note that nonexpansiveness is decidable as it reduces to computing predecessors of a regular set [34]. We will show that in the unary case the class of nonexpansive grammars strictly contains the class defined by Bhattiprolu et al. [10]. However, nonexpansiveness is a sufficient condition for the Parikh property, but not necessary. In particular, we give an example of an expansive WCFG for which there exists a Parikh-equivalent regular WCFG. This shows that a conjecture formulated by Baron and Kuich [8] in 1981 is false.[1] Furthermore, we will show that nonexpansiveness is not necessary for the property even when the alphabet is unary by means of a similar example.

In the second part of our work, we study the question of whether the Parikh property is decidable. To the best of our knowledge, this question is open. However, it implicitly follows from a result by Kuich et al. [67] that, when we equivalently formulate the property in terms of formal power series, it is decidable over the semiring of rational numbers. Their proof relies on a cumbersome elimination procedure which is hard

[1] Essentially, they conjectured that every unambiguous WCFG $\mathcal{G}$ is nonexpansive iff $\mathcal{G}$ has the Parikh property [8, Conjecture C].

to perform even on small examples. We sidestep this issue using a more standard technique: *Groebner bases*. This way, we illustrate the algorithm on examples with the support of mainstream open-source computer algebra systems.

5.1.1 Notation and Definitions

First, recall that $\mathbb{S}$ will always denote a *commutative* semiring.

As usual, we will denote by Σ^* the set of all words over the alphabet Σ and we will use w, w' and w_i $(i \in \mathbb{N})$ to denote its elements. On the other hand, we will use $\Sigma^{\oplus}$ to denote all the *monomials* in the variables Σ which we will denote by v, v' and v_i. Thus, a monomial v in the variables $\Sigma = \{a_1, \ldots, a_n\}$ is an expression of the form $a_1^{x_1}, \ldots a_n^{x_n}$ where each $x_i \geq 1$ denotes the number of occurrences of a_i in v. Note that, if $x_i = 0$ we do not write a_i. The length of v is defined as $|v| \stackrel{\text{def}}{=} x_1 + \ldots + x_n$. For instance, if $\Sigma = \{a, b\}$ then all the elements in Σ^* of length 3 containing 2 a's and 1 b are the words aab, aba and baa, while the only element with that property in $\Sigma^{\oplus}$ is the monomial a^2b.

Next we give a formal definition of the notion of *nonexpansive* context-free grammar.

Definition 5.1.1 (Nonexpansive CFG)**.** A CFG $\mathcal{G} = (V, \Sigma, S, R)$ is *nonexpansive* if no derivation is of the form $X \Rightarrow^* \alpha_0 X \alpha_1 X \alpha_2$ with $X \in V$ and $\alpha_i \in (\Sigma \cup V)^*$, for each $i \in \{0, \ldots, 2\}$. Otherwise, the $\mathcal{G}$ is *expansive*.

Let us illustrate this definition with an example.

Example 5.1.2. Let $\mathcal{G}_1 = (\{X, Y\}, \{a, b\}, X,\ R = \{X \to a\,X\,Y,\ X \to a\,X, X \to a, Y \to b\,Y, Y \to b\})$ be a CFG. Since no derivation is of the form $X \Rightarrow^* \alpha_0 X \alpha_1 X \alpha_2$ with $X \in V$ and $\alpha_i \in (\Sigma \cup V)^*$, $\mathcal{G}_1$ is nonexpansive. Observe that there exists a value $k \geq 1$ such that it is possible to generate every word of $\mathcal{G}_1$ using a derivation sequence where the derivation sentence with the maximum number of variables has at most k variables. For $\mathcal{G}_1$ the corresponding value of k is 2.

Now consider the CFG $\mathcal{G}_2 = (\{X\}, \{a, b\}, X, R = \{X \to a\,X\,b\,X, X \to \varepsilon\})$. Clearly, this CFG is expansive. In this particular case, there exists always a word for which its unique parse tree is of dimension n, for every $n \geq 0$. In other words, the dimension of the set of parse trees of this grammar is unbounded. However, this is not necessarily true for every expansive CFG, namely, there might be a natural value $k \geq 1$ such that every word of an expansive CFG might be generated using a derivation

sequence where the sentence with the maximum number of variables has at most k variables. ◀

We also extend the notion of *Parikh image* from words to weighted context-free grammars. To do so recall that, given a WCFG $(\mathcal{G}, W)$, $[\![\mathcal{G}]\!]_W$ denotes its *semantics* which is defined as the function $[\![\mathcal{G}]\!]_W : \Sigma^* \to \mathbb{S}$ such that $[\![\mathcal{G}]\!]_W(w) \stackrel{\text{def}}{=} W(w)$.

Definition 5.1.3 (Parikh image of a WCFG)**.** Given a WCFG $(\mathcal{G}, W)$, the *Parikh image* of $(\mathcal{G}, W)$, denoted by $\wr\mathcal{G}\int_W$, is the mapping $\wr\mathcal{G}\int_W : \Sigma^{\oplus} \to \mathbb{S}$ such that:

$$\wr\mathcal{G}\int_W(v) \stackrel{\text{def}}{=} \sum_{\substack{v=\wr w\int \\ w\in\Sigma^*}} [\![\mathcal{G}]\!]_W(w) \ .$$

We denote the formal sums $\sum_{w\in\Sigma^*} [\![\mathcal{G}]\!]_W(w)\, w$ and $\sum_{v\in\Sigma^{\oplus}} \wr\mathcal{G}\int_W(v)\, v$ by $[\![\mathcal{G}]\!]_W$ and $\wr\mathcal{G}\int_W$, respectively.

Definition 5.1.4 (Language and Parikh equivalence of WCFGs)**.** Given two WCFGs $(\mathcal{G}_1, W_1)$ and $(\mathcal{G}_2, W_2)$, $(\mathcal{G}_1, W_1)$ is *language-equivalent* to $(\mathcal{G}_2, W_2)$ iff

$$[\![\mathcal{G}_1]\!]_{W_1} = [\![\mathcal{G}_2]\!]_{W_2} \ ,$$

and $(\mathcal{G}_1, W_1)$ is *Parikh-equivalent* to $(\mathcal{G}_2, W_2)$ iff

$$\wr\mathcal{G}_1\int_{W_1} = \wr\mathcal{G}_2\int_{W_2} \ .$$

We say that a WCFG $(\mathcal{G}, W)$ is *regular* (*nonexpansive* or *cycle-free*) iff $\mathcal{G}$ is regular (nonexpansive or cycle-free, respectively).

Now we are ready to define the *Parikh property*.

Definition 5.1.5 (Parikh property)**.** A WCFG $(\mathcal{G}, W)$ satisfies the *Parikh property* iff there exists a WCFG $(\mathcal{G}_\ell, W_\ell)$ such that:

1. $(\mathcal{G}_\ell, W_\ell)$ is regular, and
2. $\wr\mathcal{G}\int_W = \wr\mathcal{G}_\ell\int_{W_\ell}$.

5.1.2 WCFGs and Algebraic Systems

Now we will establish the connection between WCFGs and algebraic systems in commuting variables. First, let us introduce some preliminary definitions.

Given $\mathbb{S}$ and an alphabet Σ, a *formal power series in commuting variables* is a mapping of $\Sigma^{\oplus}$ into $\mathbb{S}$. $\mathbb{S}\langle\langle\Sigma^{\oplus}\rangle\rangle$ denotes the set of all formal power

series in commuting variables Σ and coefficients in $\mathbb{S}$. The values of a formal power series r are denoted by (r, v) where $v \in \Sigma^{\oplus}$. As r is a mapping of $\Sigma^{\oplus}$ into $\mathbb{S}$, it can be written as a formal sum as $r = \sum_{v \in \Sigma^{\oplus}} (r, v)\, v$. When $v = \varepsilon$, we will write the term $(r, \varepsilon)\varepsilon$ of r simply as (r, ε). We define the *support* of a formal power series as the set of monomials whose coefficient is different from $0_{\mathbb{S}}$. The subset of $\mathbb{S}\langle\langle\Sigma^{\oplus}\rangle\rangle$ consisting of all series with a finite support is denoted by $\mathbb{S}\langle\Sigma^{\oplus}\rangle$ and its elements are called *polynomials*. Finally, define, the operator R_k ($k \geq 0$) applied on $r \in \mathbb{S}\langle\langle\Sigma^{\oplus}\rangle\rangle$ as $R_k(r) \stackrel{\text{def}}{=} \sum_{|v| \leq k} (r, v)v$.

Consider a WCFG $(\mathcal{G}, W)$ with $\mathcal{G} = (V, \Sigma, X_1, R)$, $V = \{X_1, \ldots, X_n\}$, and W defined over the semiring $\mathbb{S}$. We associate to $(\mathcal{G}, W)$ the algebraic system in commuting variables defined as follows. For each $X_i \in V$:

$$X_i = \sum_{\substack{\pi \in R \\ \pi = (X_i \to \alpha)}} W(\pi) \wr \alpha \int \quad . \tag{5.1}$$

We refer to this system as the *algebraic system (in commuting variables) corresponding to* $(\mathcal{G}, W)$. Sometimes, we write $\mathbb{S}\langle\Sigma^{\oplus}\rangle$-algebraic system to indicate that the coefficients of the system lie in $\mathbb{S}\langle\Sigma^{\oplus}\rangle$. Note that (5.1) can be written as follows. For each $X_i \in V$:

$$X_i = p_i \quad , \text{ with } p_i \in \mathbb{S}\langle(\Sigma \cup V)^{\oplus}\rangle \quad . \tag{5.2}$$

A *solution* to (5.2) is defined as an n-tuple $r = (r_1, \ldots, r_n)$ of elements of $\mathbb{S}\langle\langle\Sigma^{\oplus}\rangle\rangle$ such that $r_i = r(p_i)$, for $i = 1, \ldots, n$, where $r(p_i)$ denotes the series obtained from p_i by replacing, for $j = 1, \ldots, n$, simultaneously each occurrence of X_j by r_j. Note that, r_1, the first component of r, always corresponds to the solution for X_1, the initial variable of G. The *approximation sequence* $\sigma^0, \sigma^1, \ldots, \sigma^j, \ldots$ where each σ^j is an n-tuple of elements of $\mathbb{S}\langle\Sigma^{\oplus}\rangle$ associated to an algebraic system as (5.2) is defined as $\sigma^0 = (0_{\mathbb{S}}, \ldots, 0_{\mathbb{S}})$ and $\sigma^{j+1} = (\sigma^j(p_1), \ldots, \sigma^j(p_n))$ for all $j \geq 0$. We have that $\lim_{j \to \infty} \sigma^j = \sigma$ iff, for all $k \geq 0$, there exists a natural $m(k)$ such that $R_k(\sigma^{m(k)+j}) = R_k(\sigma^{m(k)}) = R_k(\sigma)$, for all $j \geq 0$. If $\lim_{j \to \infty} \sigma^j = \sigma$, then σ is a solution of (5.2) and is referred to as the *strong solution* [67, Theorem 14.1]. Note that, by definition, the strong solution is unique whenever it exits.

Finally, if $(\mathcal{G}, W)$ is a regular WCFG then each p_i in its corresponding algebraic system written as in (5.2) is a polynomial in $\mathbb{S}\langle\mathcal{M}\rangle$, where $\mathcal{M}$ denotes the set of monomials of the form $a_1^{x_1} \ldots a_m^{x_m} X_1^{y_1} \ldots X_n^{y_n}$ with $a_i \in$

Σ, $x_i, y_j \in \mathbb{N}$, for all i and j, and $\sum_{i=1}^{n} y_i \leq 1$. We call a system of this form a *regular algebraic system*.

Conversely, we associate to each $\mathbb{S}\langle\Sigma^{\oplus}\rangle$-algebraic system S in commuting variables of the form (5.2) a WCFG $(\mathcal{G}, W)$ over the semiring $\mathbb{S}$ as follows. Define $\mathcal{G} = (\{X_1, \ldots, X_n\}, \Sigma, X_1, R)$ and such that $\pi = (X_i \to \alpha) \in R$ iff $(p_i, \alpha) \neq 0_{\mathbb{S}}$. If $\pi \in R$, then $W(\pi) = (p_i, \alpha)$. We will refer to $(\mathcal{G}, W)$ as the *WCFG corresponding to the algebraic system* S. Note that if we begin with an algebraic system in commuting variables, then go to the corresponding WCFG and back again to an algebraic system, then the latter coincides with the original. However, if we begin with the WCFG, form the corresponding algebraic system and then again the corresponding WCFG, then the latter grammar may differ from the original.

5.2 Sufficient Condition for the Parikh Property

Petre [77] shows that the Parikh property is not true in general. In the following example we show a well-known WCFG (for instance, see [10, 77]) for which no regular Parikh-equivalent WCFG exists.

Example 5.2.1. Consider the WCFG $(\mathcal{G}, W)$ with $\mathcal{G} = (\{X\}, \{a\}, X, \{X \to aXX,\ X \to a\})$ and the weight function W over $(\mathbb{N}, +, \cdot, 0, 1)$ that assigns 1 to each production in the grammar. Note that, because the alphabet is unary, we have that $[\![\mathcal{G}]\!]_W = \wr\mathcal{G}\int_W$. As W assigns 1 to each grammar rule, the weight of each word can be interpreted as its ambiguity w.r.t. $\mathcal{G}$. Then, the reader can check that:

$$[\![\mathcal{G}]\!]_W = \sum_{n \geq 0} C_n\, a^{2n+1} = 1a + 1a^3 + 2a^5 + 5a^7 + 14a^9 + 42a^{11} + 132a^{13} + \ldots$$

where $C_n \stackrel{\text{def}}{=} \frac{1}{n+1}\binom{2n}{n}$, the n-th Catalan number. We will see in Example 5.3.6 that this formal power series cannot be generated by a regular WCFG. ◀

Now we show that every nonexpansive WCFG over an arbitrary commutative semiring satisfies the Parikh property. The proof is rather technical and therefore here we give a sketch of it. For a complete proof, see Section 5.6.

Theorem 5.2.2. *Let $(\mathcal{G}, W)$ be an arbitrary WCFG. If $\mathcal{G}$ is nonexpansive then $(\mathcal{G}, W)$ satisfies the Parikh property.*

Proof. The proof is constructive. Here we give the main intuition. For every nonexpansive WCFG $(\mathcal{G}, W)$, we give a 2-step construction that results in a Parikh-equivalent regular WCFG $(\mathcal{G}_\ell, W_\ell)$.

The steps are:

1. construct a new WCFG $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$, where $k \in \mathbb{N}$, language-equivalent to $(\mathcal{G}, W)$; and
2. construct a regular WCFG $(\mathcal{G}_\ell, W_\ell)$ Parikh-equivalent to $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$.

The idea behind the first step is to build a WCFG $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ that contains all the information needed to define a "strategic" derivation policy. This derivation policy is strategic in the sense that the total number of grammar variables in all derivation sentences produced along a derivation sequence is bounded by an affine function of k. To build $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ we rely on the grammar construction given by Luttenberger et al. [68].

In the second step of the construction, we use $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ to build a regular WCFG $(\mathcal{G}_\ell, W_\ell)$ that is Parikh-equivalent. Each grammar variable of $(\mathcal{G}_\ell, W_\ell)$ represents each possible sentence (without the terminals) along a derivation sequence of $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$, and each rule simulates a derivation step of $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$. Because the number of variables in the sentences is bounded by a function of k, the number of variables and rules of $(\mathcal{G}_\ell, W_\ell)$ is necessarily finite. This grammar relies on previous constructions proposed by Bhattiprolu et al. [10] and Esparza et al. [32].

The converse of Theorem 5.2.2 is not true. The next counterexample illustrates this fact by defining an expansive WCFG for which a Parikh-equivalent regular WCFG exists. Thus, nonexpansiveness does not provide an exact characterization of the Parikh property.

Example 5.2.3. Consider the WCFG $(\mathcal{G}_1, W_1)$ where $\mathcal{G}_1 = (\{X_1\}, \{a, \overline{a}\}, X_1, R_1 = \{X_1 \to aX_1,\ X_1 \to \overline{a}X_1,\ X_1 \to \varepsilon\})$ and W_1 is defined over $(\mathbb{N}, +, \cdot, 0, 1)$ and assigns 1 to each rule in R_1. First, note that $(\mathcal{G}_1, W_1)$ is regular and the weight of each word can be interpreted as its ambiguity w.r.t. $\mathcal{G}_1$. Because $\mathcal{G}_1$ is unambiguous, the weight of each word in the language of $\mathcal{G}_1$

is 1. It is easy to see that:

$$[\![\mathcal{G}_1]\!]_{W_1} = (a+\overline{a})^* = \sum_{n\geq 0}(a+\overline{a})^n = 1\varepsilon+1a+1\overline{a}+1a\overline{a}+1\overline{a}a+1aaa+1aa\overline{a}+1a\overline{a}a+\dots$$

Now consider the expansive WCFG $(\mathcal{G}_D, W_D)$ where $\mathcal{G}_D = (\{D\}, \{a, \overline{a}\}, D, R_D = \{D \to aD\overline{a}D,\ D \to \varepsilon\})$ and W_D is defined over $\mathbb{N}$ and assigns 1 to each rule in R_D. The grammar $\mathcal{G}_D$ generates the Dyck language[2] L_D over the alphabet $\{a, \overline{a}\}$ and it is also unambiguous. It is well-known that L_D is a deterministic context-free language (DCFL). Then the complement[3] of L_D, namely $\{a, \overline{a}\}^* \backslash L_D$, is also a DCFL, and thus admits an unambiguous[4] CFG. Define the WCFG $(\mathcal{G}_{\overline{D}}, W_{\overline{D}})$, where $\mathcal{G}_{\overline{D}} = (\{\overline{D}, D, Y, Z\}, \{a, \overline{a}\}, \overline{D}, R_{\overline{D}})$, $R_{\overline{D}}$ is given by:

$$\begin{aligned} \overline{D} &\to D\,\overline{a}\,Y \mid D\,a\,Z & Y &\to a\,Y \mid \overline{a}\,Y \mid \varepsilon \\ D &\to a\,D\,\overline{a}\,D \mid \varepsilon & Z &\to D\,a\,Z \mid D\ . \end{aligned}$$

and the weight function $W_{\overline{D}}$ is defined over $(\mathbb{N}, +, \cdot, 0, 1)$ and assigns 1 to each production in the grammar.

Finally, define $(\mathcal{G}_2, W_2)$ where $\mathcal{G}_2 = (\{X_2, D, \overline{D}, Y, Z\}, \{a, \overline{a}\}, X_2, R_2)$, R_2 is defined as $R_2 = \{X_2 \to D,\ X_2 \to \overline{D}\} \cup R_{\overline{D}}$ where W_2 is defined over $\mathbb{N}$ and assigns 1 to each rule in R_2.

First, $\mathcal{G}_2$ is expansive because $\mathcal{G}_D$ is expansive. Furthermore, D and $\overline{D}$ generate unambiguous context-free languages that are complementary over $\{a, \overline{a}\}$. As the weight of each word in $(\mathcal{G}_2, W_2)$ corresponds to its ambiguity, we have that $[\![\mathcal{G}_2]\!]_{W_2} = (a + \overline{a})^*$. Hence $[\![\mathcal{G}_1]\!]_{W_1} = [\![\mathcal{G}_2]\!]_{W_2}$, and thus $\wr\mathcal{G}_1\wr_{W_1} = \wr\mathcal{G}_2\wr_{W_2}$. Since $(\mathcal{G}_1, W_1)$ is regular, we conclude that $(\mathcal{G}_2, W_2)$ is expansive and satisfies the Parikh property. ◀

We can give a similar counterexample over a *unary* alphabet. This shows that nonexpansiveness is not necessary for the Parikh property even in the unary case.

Example 5.2.4. The idea behind this example is to use the definition of $(\mathcal{G}_2, W_2)$ from Example 5.2.3 and replace each occurrence of the symbol

[2] The *Dyck language* is defined as the set of well-paranthesized words over the alphabet {"(",")"}. For clarity, we will instead use the alphabet $\{a, \overline{a}\}$.

[3] The class of deterministic context-free languages is closed under complementation.

[4] Note that every deterministic context-free languages is unambiguous (the reverse is not true, in general).

$\overline{a}$ in the rules of $(\mathcal{G}_2, W_2)$ by a. Thus, define the WCFG $(\mathcal{G}, W)$ where $\mathcal{G} = (\{X, \overline{D}, D, Y, Z\}, \{a\}, X, R)$, R is given by:

$$X \to D \mid \overline{D} \qquad \overline{D} \to D\,a\,Y \mid D\,a\,Z \qquad Z \to D\,a\,Z \mid D \ ,$$
$$D \to a\,D\,a\,D \mid \varepsilon \qquad Y \to a\,Y \mid \varepsilon$$

and the weight function W is defined over $(\mathbb{N}, +, \cdot, 0, 1)$ and assigns 1 to each production in the grammar except from the rule $Y \to a\,Y$ which is assigned weight 2. Recall that $\wr\mathcal{G}_2\int_{W_2} = (a + \overline{a})^*$. Now, relying on our construction of $(\mathcal{G}, W)$, we have that $\wr\mathcal{G}\int_W$ is the formal power series that results from replacing each $\overline{a}$ by a in the series $\wr\mathcal{G}_2\int_{W_2}$. Thus, we obtain that $\wr\mathcal{G}\int_W = (a + a)^* = (2a)^*$. The reader can check that the formal power series $(2a)^*$ corresponds to the Parikh image of the regular WCFG $(\mathcal{G}_\ell, W_\ell)$ where $\mathcal{G}_\ell$ is defined as $\mathcal{G}_\ell = (\{X\}, \{a\}, X, \{X \to aX,\ X \to \varepsilon\})$ and the weight function W_ℓ is defined over $(\mathbb{N}, +, \cdot, 0, 1)$ and assigns weight 2 to the rule $X \to aX$ and weight 1 to the rule $X \to \varepsilon$. ◀

5.3 A Decision Procedure Over the Rationals

In this section we give a decision procedure that tells whether or not a given WCFG with weights over the rational semiring satisfies the Parikh property. Our procedure relies on a decidability result by Kuich and Salomaa [67, Theorem 16.13]. It implicitly follows from this result that the Parikh property is decidable over the rational semiring. However, their decision procedure is hard to follow as it relies on algebraic methods beyond the scope of this dissertation. This makes its implementation rather involved even for small instances. We propose an alternative method to sidestep this problem using Groebner bases.

We start by showing that the Parikh image of a cycle-free WCFG corresponds to the solution for the initial variable in its corresponding algebraic system.

Theorem 5.3.1. *Let $(\mathcal{G}, W)$ be a cycle-free WCFG and let S be the algebraic system in commuting variables corresponding to $(\mathcal{G}, W)$. Then, the strong solution r of S exists and the first component of r corresponds to $\wr\mathcal{G}\int_W$.*

Proof. First, we prove that if a WCFG $(\mathcal{G}, W)$ defined over a commutative semiring $\mathbb{S}$ is cycle-free, then the strong solution of the algebraic system in commuting variables corresponding to $(\mathcal{G}, W)$ exists. Second, we show that the first component of the strong solution corresponds to $\wr\mathcal{G}\int_W$.

We give a proof by contradiction of the first statement. Thus, we prove the following:

Let S be the algebraic system corresponding to a WCFG $(\mathcal{G}, W)$ and let $\sigma^0, \sigma^1, \ldots, \sigma^j, \ldots$ be the approximation sequence associated to S. If $\lim_{j\to\infty} \sigma^j$ does not exist, then $(\mathcal{G}, W)$ is not cycle-free.

Note that $\lim_{j\to\infty} \sigma^j$ does not exist iff either the approximation sequence oscillates between a finite number of states, or there exists a length $k \geq 0$ such that the coefficient of some monomial $v \in \Sigma^{\oplus}$, with $|v| \leq k$, increases (w.r.t. the partial ordering of $\mathbb{S}$) unboundedly at every step in the approximation sequence. Formally, there exists $k \geq 0$ such that, for every $m \geq 0$, $R_k(\sigma^m) \leq R_k(\sigma^{m+j})$, for some $j > 0$, where $\leq$ is the partial ordering of $\mathbb{S}$. The first case cannot hold because every approximation sequence is monotonic [67, Lemma 14.4]. Next we see that, if the second case holds, then necessarily the corresponding WCFG $(\mathcal{G}, W)$ is not cycle-free.

To give some intuition, let us consider the following simple scenario. Consider that the set of variables V of $\mathcal{G}$ contains only 1 variable, say X, and assume that the limit of the approximation sequence of its corresponding algebraic system does not exist. Intuitively, for each $j \geq 0$, the monomials occurring in the finite series σ^j in the approximation sequence of the corresponding system S correspond to the monomials that can be produced by $\mathcal{G}$ in at most j derivation steps, and the coefficient of each monomial corresponds to its weight if only derivations of at most j steps are considered. By hypothesis, there exists a length $k \geq 0$ such that the coefficient of some monomial $v \in \Sigma^{\oplus}$, with $|v| \leq k$, increases unboundedly at every step in the approximation sequence. It means that, for each derivation sequence of $m (m \geq 1)$ steps generating $w \in \Sigma^*$ with $\wr w \int = v$, there is another derivation sequence of $l > m$ steps generating $w' \in \Sigma^*$ with $\wr w' \int = v$. In other words, there exist derivation sequences in $\mathcal{G}$ of arbitrary

length. Because the number of rules of $\mathcal{G}$ is finite and so is the number of words $w \in \Sigma^*$ such that $\lceil w \rfloor = v$, then either $\mathcal{G}$ contains a rule of the form $X \to X$, or $\mathcal{G}$ contains a rule of the form $X \to \alpha$ with $\alpha \in \{X\}^+$ and $|\alpha| > 1$, and a rule of the form $X \to \varepsilon$. It follows that there exists a derivation sequence in $\mathcal{G}$ of the form $X \Rightarrow^+ X$, and thus $\mathcal{G}$ is not cycle-free.

The proof of the statement for every WCFG $(\mathcal{G}, W)$ with an arbitrary number of variables goes in a similar fashion. By hypothesis, there exists a length $k \geq 0$ such that for some monomial $v \in \Sigma^{\oplus}$, with $|v| \leq k$, for every derivation sequence of $m (m \geq 1)$ steps generating $w \in \Sigma^*$ with $\lceil w \rfloor = v$, there is another derivation sequence of l with $l > m$ generating $w' \in \Sigma^*$ with $\lceil w' \rfloor = v$. That is, there are arbitrarily large derivation sequences in $\mathcal{G}$ using rules that do not add alphabet symbols. Since the number of grammar rules of $\mathcal{G}$ is finite and so is the number of words $w \in \Sigma^*$ such that $\lceil w \rfloor = v$, there must exist a cycle in $\mathcal{G}$, i.e., a derivation sequence of the form $X_i \Rightarrow^+ X_i$, where X_i is a variable of $\mathcal{G}$. Then, we conclude that $\mathcal{G}$ is not cycle-free.

We have shown that the strong solution r of S exists. Now we prove that the first component of r corresponds to $\lceil \mathcal{G} \rfloor_W$. First, consider $\widetilde{S}$ as the algebraic system in *noncommuting* variables corresponding to a cycle-free $(\mathcal{G}, W)$ that is built as follows:

$$X_i = \sum_{\substack{\pi \in R \\ \pi = (X_i \to \alpha)}} W(\pi)\, \alpha \ . \tag{5.3}$$

Note that (5.3) now is of the form:

$$X_i = p_i \ \ , \text{ with } p_i \in \mathbb{S}\langle (\Sigma \cup V)^* \rangle \ .$$

Salomaa et al. prove that, if $\widetilde{r_1}$ is the first component of the strong solution of $\widetilde{S}$, then $\widetilde{r_1}(w) = [\![\mathcal{G}]\!]_W(w)$ for every $w \in \Sigma^*$ when the weight function W is defined over $(\mathbb{N}, +, \cdot, 0, 1)$ and assigns 1 to each rule in $\mathcal{G}$ [82, Theorem 1.5]. In the proof they denote $[\![\mathcal{G}]\!]_W(w)$ by $\text{amb}(\mathcal{G}, w)$, as it corresponds to the ambiguity of w according to $\mathcal{G}$. The proof for the more general case where W is any arbitrary weight function defined over a commutative semiring reduces to replacing $[\![\mathcal{G}]\!]_W(w)$ by $\text{amb}(\mathcal{G}, w)$ and using the corresponding semiring operations.

Now consider S as the algebraic system in commuting variables corresponding to $(\mathcal{G}, W)$ and built as in (5.1) (page 99). Let r_1 be the first component of its solution. It is known that r_1 and $\widetilde{r_1}$ verify the following equality [67]. For each $v \in \Sigma^{\oplus}$:

$$r_1(v) = \sum_{\substack{v=\lbag w \rbag \\ w \in \Sigma^*}} \widetilde{r_1}(w) \ .$$

Then for each $v \in \Sigma^{\oplus}$:

$$\lbag \mathcal{G} \rbag_W (v) = \sum_{\substack{v=\lbag w \rbag \\ w \in \Sigma^*}} [\![\mathcal{G}]\!]_W(w) = \sum_{\substack{v=\lbag w \rbag \\ w \in \Sigma^*}} \widetilde{r_1}(w) = r_1(v) \ ,$$

where the first equality holds by definition of $\lbag \mathcal{G} \rbag_W$.

Now we introduce the class of *rational* power series in commuting variables Σ with coefficients in the semiring $\mathbb{S}$, denoted by $\mathbb{S}^{rat}\langle\langle \Sigma^{\oplus} \rangle\rangle$.

Definition 5.3.2 (Rational Power Series in Commuting Variables)**.** A formal power series $r \in \mathbb{S}^{rat}\langle\langle \Sigma^{\oplus} \rangle\rangle$ iff r is the first component of the solution of a regular algebraic system in commuting variables.

From the previous definition and Theorem 5.3.1 we can characterize the WCFGs that satisfy the Parikh property as follows.

Lemma 5.3.3. *Let $(\mathcal{G}, W)$ be a cycle-free WCFG. Then $(\mathcal{G}, W)$ satisfies the Parikh property iff $\lbag \mathcal{G} \rbag_W \in \mathbb{S}^{rat}\langle\langle \Sigma^{\oplus} \rangle\rangle$.*

Proof. Let $(\mathcal{G}, W)$ be a WCFG with $\lbag \mathcal{G} \rbag_W \in \mathbb{S}^{rat}\langle\langle \Sigma^{\oplus} \rangle\rangle$. Then $\lbag \mathcal{G} \rbag_W$ is the first component of the solution of a regular algebraic system S. Hence, the WCFG $(\mathcal{G}_\ell, W_\ell)$ corresponding to S is regular and $\lbag \mathcal{G} \rbag_W = \lbag \mathcal{G}_\ell \rbag_{W_\ell}$.

Now, let $(\mathcal{G}, W)$ be a WCFG with the Parikh property. Then there exists a regular WCFG $(\mathcal{G}_\ell, W_\ell)$ such that $\lbag \mathcal{G} \rbag_W = \lbag \mathcal{G}_\ell \rbag_{W_\ell}$. Let S be the regular algebraic system corresponding to (G_ℓ, W_ℓ). The first component of its solution vector is $\lbag \mathcal{G}_\ell \rbag_{W_\ell}$, and thus it is in $\mathbb{S}^{rat}\langle\langle \Sigma^{\oplus} \rangle\rangle$. As $\lbag \mathcal{G} \rbag_W = \lbag \mathcal{G}_\ell \rbag_{W_\ell}$ then $\lbag \mathcal{G} \rbag_W$ is also in $\mathbb{S}^{rat}\langle\langle \Sigma^{\oplus} \rangle\rangle$.

Next we observe that every WCFG $(\mathcal{G}, W)$ defined over a commutative *ring* with the Parikh property satisfies a linear equation of a special kind. This result directly follows from Theorem 16.4 in [67].

Theorem 5.3.4. *Let $(\mathcal{G}, W)$ be a cycle-free WCFG with W defined over a commutative ring $\mathbb{S}$. Then $(\mathcal{G}, W)$ satisfies the Parikh property iff $\wr\mathcal{G}\int_W$ satisfies a linear equation of the form: $X = sX + t$, for some $s, t \in \mathbb{S}\langle\Sigma^{\oplus}\rangle$ with $(s, \varepsilon) = 0$.*

> ***Proof.*** The result is a consequence of Lemma 5.3.3 and Theorem 16.4 in [67]. To be precise, the latter theorem states that, if $\mathbb{S}$ is a ring, then $r \in \mathbb{S}^{rat}\langle\langle\Sigma^{\oplus}\rangle\rangle$ iff there exist polynomials $s, t \in \mathbb{S}\langle\Sigma^{\oplus}\rangle$ such that r is the solution of $X = sX + t$, with $(s, \varepsilon) = 0$.

It follows from the previous theorem that, given a WCFG $(\mathcal{G}, W)$ with W defined over a commutative ring, if such a linear equation exists then $(\mathcal{G}, W)$ satisfies the Parikh property; otherwise it does not. Now we will use a result by Kuich et al. [67] to conclude that, if $(\mathcal{G}, W)$ is defined over $\mathbb{Q}$ then there exists an irreducible polynomial $q(X)$ such that q evaluates to 0 when $X = \wr\mathcal{G}\int_W$, denoted by $q(\wr\mathcal{G}\int_W) \equiv 0$. Intuitively, this polynomial contains all the information needed to decide whether or not $(\mathcal{G}, W)$ has the Parikh property.

Theorem 5.3.5 (from Theorem 16.9 in [67])**.** *Let S be the $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$-algebraic system in commuting variables corresponding to a cycle-free WCFG. Let r_1 be the first component of its strong solution. Then there exists an irreducible polynomial $q(X_1)$ with coefficients in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, and unique up to a factor in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, such that $q(r_1) \equiv 0$.*

Kuich et al. [67] show that the polynomial q is effectively computable by means of a procedure based on the classical elimination theory. Now we develop an alternative method using Groebner bases. Before introducing this technique, we give some intuition on the ideas presented above by revisiting the examples of the previous section.

Example 5.3.6. Consider the cycle-free WCFG $(\mathcal{G}, W)$ defined in Example 5.2.1 where the weight function W is now defined over $(\mathbb{Q}, +, \cdot, 0, 1)$ and assigns 1 to each production in the grammar. The algebraic system S corresponding to $(\mathcal{G}, W)$ is given by the equation $X = a\,X^2 + a$. Let r_1 be its strong solution. Assume for now that the irreducible polynomial

$q(X) \in \mathbb{Q}\langle\{a\}^{\oplus}\rangle\langle X\rangle$ from Theorem 5.3.5 is $q(X) = aX^2 - X + a$ (later we will give its construction using Groebner bases). We will see later that the fact that $q(X)$ is not linear is enough to conclude that $(\mathcal{G}, W)$ does not satisfy the Parikh property (as we expected). Note that the solution of S is $r_1 = \frac{1-\sqrt{1-4a^2}}{2a}$, which written as a series corresponds to $r_1 = \sum_{n\geq 0} C_n\, a^{2n+1}$, with $C_n = \frac{1}{n+1}\binom{2n}{n}$ the n-th Catalan number. It is known that this formal power series cannot be written as the solution of a linear equation with coefficients in $\mathbb{Q}\langle\{a\}^{\oplus}\rangle$ [10]. ◀

Example 5.3.7. Now we will consider the same WCFG $(\mathcal{G}_2, W_2)$ given in Example 5.2.3 except that W_2 is now defined over $(\mathbb{Q}, +, \cdot, 0, 1)$ (still, W_2 assigns 1 to each production in the grammar). Note that $(\mathcal{G}_2, W_2)$ is cycle-free. The grammar variable D generates all the words in the Dyck language L_D over the alphabet $\{a, \overline{a}\}$, while the variable $\overline{D}$ generates $\{a, \overline{a}\}^* \setminus L_D$. The system S corresponding to $(\mathcal{G}_2, W_2)$ consists of the following equations:

$$\begin{aligned} X_2 &= D + \overline{D} & \overline{D} &= D\,\overline{a}\,Y + D\,a\,Z & Z &= D\,a\,Z + D \ . \\ D &= a\,D\,\overline{a}\,D + 1 & Y &= a\,Y + \overline{a}\,Y + 1 \end{aligned}$$

Let $\sigma = (r_1, r_2, r_3, r_4, r_5)$ be its strong solution where r_1 corresponds to the solution for the start variable X_2. Assume for now that the irreducible polynomial $q(X_2) \in \mathbb{Q}\langle\{a, \overline{a}\}^{\oplus}\rangle\langle X_2\rangle$ described by Theorem 5.3.5 is:

$$q(X_2) = (1 - (a + \overline{a}))X_2 - 1 \ .$$

We observe that q is linear in X_2 and can be written as:

$$q(X_2) = (1 - s)X_2 - t = (1 - (a + \overline{a}))X_2 - 1 \ ,$$

with $(s, \varepsilon) = 0$. Thus, by Theorem 5.3.4, we conclude that $(\mathcal{G}_2, W_2)$ satisfies the Parikh property, as we expected. ◀

5.3.1 Groebner Bases

Now we develop the technique we will use to construct the irreducible polynomial of Theorem 5.3.5: Groebner bases. A Groebner basis is a set of polynomials in one or more variables enjoying certain properties. Given a set of polynomials F with coefficients in a field, one can compute a Groebner basis G of F with the property that G has the same solutions as

F when interpreted as a polynomial system of equations. Then, problems such as finding the solutions for the system induced by F, or looking for alternative representations of polynomials in terms of other polynomials become easier using G instead of F. One of the main insights for using Groebner bases is that they are *effectively constructable* using standard computer algebra systems, for any set of polynomials with coefficients in a field.

We are interested in computing Groebner bases of algebraic systems in commuting variables corresponding to weighted context-free grammars. Given a WCFG and its corresponding algebraic system, our goal is to obtain a system with the same solution as the original, and such that one equation in the new system depends only on the initial grammar variable X_1. This equation will contain all the information needed to decide whether or not the given WCFG satisfies the Parikh property. We will not enter into the technical details of how Groebner bases are constructed and their properties as these lie beyond the scope of this dissertation (however, an explicit reference will be given in connection with each result applied). Instead, we will give a result that encapsulates all the preconditions and postconditions we need for our purpose (Theorem 5.3.11). We first introduce the definitions that will appear in the theorem.

In what follows, K will always denote a field. First we need to introduce the notion of *ideal*.

Definition 5.3.8 (Ideal)**.** Let $K\langle V^{\oplus}\rangle$ denote the ring of polynomials in variables V and with coefficients in K. A subset $I \subset K\langle V^{\oplus}\rangle$ is an *ideal* iff:

1. $0_K \in I$,
2. if $f, g \in I$ then $f + g \in I$, and
3. if $f \in I$ and $h \in K\langle V^{\oplus}\rangle$ then $h \cdot f \in I$.

Given a set of polynomials $F = \{f_1, \ldots, f_n\}$, we define $\langle F\rangle$ as $\langle F\rangle \stackrel{\text{def}}{=} \{\sum_{i=1}^{n} h_i \cdot f_i \mid h_i \in K\langle V^{\oplus}\rangle, f_i \in F\}$. It can be shown that $\langle F\rangle$ is an ideal [26] and we call it the *ideal generated by* F. When an ideal is generated by a finite number of polynomials $g_1, \ldots, g_n \in K\langle V^{\oplus}\rangle$, we say that $g_1, \ldots, g_n$ is a *basis* of the ideal. It is known that every ideal in $K\langle V^{\oplus}\rangle$ has a basis (actually many, but the ones we are particularly interested in are the so-called Groebner bases) [26]. If one considers the set of polynomial equations $\{f = 0 \mid f \in F\}$, denoted by $F = \mathbf{0}$, then the set of all solutions of $F = \mathbf{0}$ is defined as $\{(r_1, r_2, \ldots, r_n) \in K^n \mid$

$f(r_1, \ldots, r_n) \equiv 0$, for all $f \in F\}$. Then, given two sets of polynomials F and G, if $\langle F \rangle = \langle G \rangle$ then the set of solutions of $F = \mathbf{0}$ coincides with the set of solutions of $G = \mathbf{0}$ [26].

To construct a Groebner basis of an ideal I, one needs to impose first a total ordering on the monomials of variables occuring in I. This choice is significant as different orderings lead to different Groebner bases with different properties. We are interested in computing Groebner bases with the *elimination property* for the initial variable X_1, i.e., bases where at least one polynomial depends only on X_1. Hence, we will always impose the *reverse lexicographic ordering* to construct Groebner bases.

Definition 5.3.9 (Reverse Lexicographic Order)**.** Let $V = \{X_1, \ldots, X_n\}$ be a set of variables. Let u and v be two monomials in $V^{\oplus}$. Then, u *is greater than* v *w.r.t. the reverse lexicographic ordering*, denoted by $u \succ_{revlex} v$, iff the first non-zero component of the vector $\wr u \int - \wr v \int$ is negative.

Note that Definition 5.3.9 implies an ordering of the variables: $X_n \succ_{revlex} X_{n-1} \succ_{revlex} \ldots \succ_{revlex} X_1$.

The reason for choosing the *reverse* lexicographic ordering is that, in order to compute a Groebner basis with the elimination property for the initial variable X_1, we need X_1 to be the least monomial (with one or more variable). In what follows, the phrase "w.r.t. the reverse lexicographic ordering" (for some given $V = \{X_1, \ldots, X_n\}$) will refer to the one described in Definition 5.3.9 with variables V, unless stated otherwise.

Fixed a total monomial ordering, we define the *leading monomial* (LM) of a polynomial p as the greatest monomial in p, and we denote it by $LM(p)$. We define the *leading term* (LT) of p as the leading monomial of p together with its coefficient, and we denote it by $LT(p)$.

Finally, we introduce the notion of a *reduced* Groebner basis which allows to define uniquely a Groebner basis of an ideal of polynomials.

Definition 5.3.10 (Reduced Groebner Basis)**.** Let F be a set of polynomials and G a Groebner basis of $\langle F \rangle$. Then, G is a *reduced* Groebner basis of $\langle F \rangle$ iff for each $g_i \in G$:

1. the coefficient of $LT(g_i) = 1$, and
2. $LM(g_i)$ does not divide any term of any g_j with $i \neq j$.

For a given set of polynomials F and monomial ordering $\succ$, there exists exactly one reduced Groebner basis of $\langle F \rangle$ w.r.t. $\succ$ [26]. We abuse notation

and write $K\langle X\rangle$ instead of $K\langle\{X\}^{\oplus}\rangle$ to refer to the ring of polynomials in the variable X with coefficients in K. Now we are ready to give the theorem.

Theorem 5.3.11. *Let K be a field and $V = \{X_1, \ldots, X_n\}$ a set of variables. Let $F \subseteq K\langle V^{\oplus}\rangle$ be a set of polynomials such that the strong solution of the system $F = \mathbf{0}$ is $(r_1, \ldots, r_n)$ where r_i corresponds to the solution for X_i. Let G be the reduced Groebner basis of $\langle F\rangle$ w.r.t. the reverse lexicographic ordering. Then the following properties are satisfied:*

1. *$(r_1, \ldots, r_n)$ is also the strong solution of the system $G = \mathbf{0}$, and*
2. *there is exactly one polynomial $g \in G$ s.t. $g \in K\langle X_1\rangle$, and for that g we have $g(r_1) \equiv 0$.*

Proof. Property 1. follows from the fact that G is a basis of $\langle F\rangle$. Now we prove property 2.

G is a Groebner basis of $\langle F\rangle$ w.r.t. the reverse lexicographic ordering. Then, as a result of the Elimination Theorem [26, Theorem 3.1.2], $G \cap K\langle X_1\rangle$ is a Groebner basis of $\langle F\rangle \cap K\langle X_1\rangle$. Assume first that $G \cap K\langle X_1\rangle$ contains only the zero polynomial (the constant polynomial whose coefficients are equal to 0). Then the ideal $\langle F\rangle \cap K\langle X_1\rangle$ also contains only the zero polynomial. But this contradicts Theorem 5.3.5. Then $G \cap K\langle X_1\rangle$ contains at least one nonzero polynomial g. Assume now that $G \cap K\langle X_1\rangle$ contains two different elements g_1 and g_2 in $K\langle X_1\rangle$. W.l.o.g., let g_1 be such that $LM(g_1) \succeq_{revlex} LM(g_2)$. Thus, $LM(g_1)$ divides (at least) the leading term of g_2. Then G is not in reduced form (contradiction). We conclude that there is exactly one (nonzero) polynomial $g \in G$ such that $g \in K\langle X_1\rangle$. Finally, $g(r_1) \equiv 0$ follows from 1. and the fact that $g \in (G \cap K\langle X_1\rangle)$.

Now we show in Theorem 5.3.12 how to construct q using Groebner bases. Finally, we give in Theorem 5.3.14 the main result of this section.

Theorem 5.3.12. *Let S be a $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$-algebraic system in commuting variables corresponding to a cycle-free WCFG and r_1 be the first component of its strong solution. Then an irreducible polynomial $q(X_1)$ with coefficients in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ such that $q(r_1) \equiv 0$ can be effectively constructed.*

Proof. We begin with the first part of the algorithm. Let K be the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, i.e., the smallest field (w.r.t. inclusion) containing $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$. Consider S as defined in (5.2) (page 99) where now each polynomial p_i has its coefficients in K and its variables in V, and let $F \subseteq K\langle V^{\oplus}\rangle$ be the set of polynomials $\{p_i \mid 1 \leq i \leq n\}$. Construct the reduced Groebner basis G of F w.r.t. the reverse lexicographic ordering. Let $G = \{g_1, \ldots, g_s\}$ with $s \geq 1$. By Theorem 5.3.11, there is exactly one $g \in G$ such that $g \in K\langle X_1\rangle$, and g satisfies $g(r_1) \equiv 0$.

We cannot conclude yet that $g(X_1)$ is the polynomial $q(X_1)$ we are looking for since $g(X_1)$ might not be irreducible in the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$. This constitutes the second part of the algorithm which follows the method given in [67] to obtain from $g(X_1)$ an irreducible polynomial $q(X_1)$ such that $q(r_1) \equiv 0$.

Compute the factorization[a] of g in the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ and let $\{q_1(X_1), \ldots, q_m(X_1)\}$ with $m \geq 1$ be the set of all irreducible polynomials obtained thus as factors. Because $g(r_1) \equiv 0$, there exists an index j_0 with $1 \leq j_0 \leq m$ such that $q_{j_0}(r_1) \equiv 0$ and $q_j(r_1) \not\equiv 0$ for $j \neq j_0$ and $1 \leq j \leq m$. Now we show how to find j_0.

Using the operator R_k introduced in the beginning of Section 5.1.2, we have that $R_k(q_{j_0}(R_k(r_1))) \equiv 0$ for all $k \geq 0$, while for each $j \neq j_0$ there is always an index k_j such that $R_{k_j}(q_j(R_{k_j}(r_1))) \not\equiv 0$. Then, eventually an index j_0 is always found.

Let $q_{j_0}(X_1) = \frac{n_k}{d_k}X_1^k + \frac{n_{k-1}}{d_{k-1}}X_1^{k-1} + \ldots + \frac{n_0}{d_0}$ with $k \geq 0, n_i, d_i \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle$ and $d_i \neq 0$ for all i. Let $lcm(d_0, \ldots, d_k)$ denote the least common multiple of $d_0, \ldots, d_k$ and define $q(X_1) = lcm(d_0, \ldots, d_k) \cdot q_{j_0}(X_1)$. Now $q(X_1) \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle\langle X_1\rangle$ and this completes the algorithm.

[a] Polynomial factorizations are performed w.r.t. polynomials with coefficients in the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ which is a computable field.

Remark 5.3.13. It is worth noting that, even though $q(X_1)$ is an irreducible polynomial over K, the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, it might not be irreducible over $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ since it might have a factorization consisting of a polynomial $\widetilde{q}(X_1) \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle\langle X_1\rangle$ of the same degree and one or more constant polynomials over $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, i.e., polynomials of degree zero, that are not units in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$. However, since constant factors are not relevant for the result, we say that a polynomial over $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ is *irreducible* iff either no factorization exists, or, if there is one, then it is of the aforementioned form.

Theorem 5.3.14. *Let $(\mathcal{G}, W)$ be a cycle-free WCFG with W defined over*

$\mathbb{Q}$. *Then, it is decidable whether or not* $(\mathcal{G}, W)$ *verifies the Parikh property.*

> ***Proof.*** Let S be the $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$-algebraic system corresponding to $\mathcal{G}$ and let r_1 be the first component of its strong solution. Construct the irreducible polynomial $q(X_1)$ with coefficients in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ as in Theorem 5.3.12. By Theorem 5.3.4, we only need to check whether or not the equation $q(X_1) = 0$ can be written as a linear equation of the form: $(1-s)X_1 - t = 0$, with $s, t \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle$ and $(s, \varepsilon) = 0$.
>
> Observe that the procedure given in Theorem 5.3.12 is complete, i.e., if the polynomial q obtained is not linear in X_1 then there cannot exist a polynomial $q_\ell(X_1)$ with coefficients in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ and linear in X_1 such that $q_\ell(r_1) \equiv 0$. If it were the case, then q_ℓ would be necessarily a factor of q, and this contradicts the fact that q is irreducible over $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$.
>
> Then, if q is not linear in X_1, we conclude that (G, W) does not satisfy the Parikh property. Otherwise, $q(X_1)$ can be rewritten as $q(X_1) = (1-s)X_1 - t$ with $s, t \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle$ and $(s, \varepsilon) = 0$, and we conclude that (G, W) satisfies the Parikh property.

Consider a WCFG $(\mathcal{G}, W)$ with r_1 the first component of the solution of its corresponding algebraic system. Observe that, if the decision procedure returns a positive answer for $(\mathcal{G}, W)$ then the polynomial $q(X_1)$ constructed as in Theorem 5.3.12 is of the form:

$$q(X_1) = (s_0 - s_1)X_1 - t = 0 \ ,$$

with $s_0 \in \mathbb{Q}, s_0 \neq 0$ and $s_1, t \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle$ with $(s_1, \varepsilon) = (t, \varepsilon) = 0$. It follows that the algebraic system consisting of the equation:

$$X_1 = \frac{1}{s_0} s_1 X_1 + \frac{1}{s_0} t \ , \tag{5.4}$$

has also r_1 as solution. Then a regular WCFG Parikh-equivalent to $(\mathcal{G}, W)$ is the one corresponding to the regular algebraic system (5.4).

Now we complete Examples 5.3.6 and 5.3.7 by following the decision procedure given in Theorem 5.3.14 and giving the construction of a Parikh-equivalent regular WCFG (if exists). Additionally, we give a third example.

The Groebner bases we show in these examples were computed using the `groebner_basis` method of the open-source mathematics software system SageMath.

Example 5.3.15. Consider the WCFG $(\mathcal{G}, W)$ given in Example 5.3.6. Recall that its corresponding algebraic system S is given by the equation $X = a\,X^2 + a$. Let r be its strong solution. Now we construct the irreducible polynomial $q(X) \in \mathbb{Q}\langle\{a\}^{\oplus}\rangle\langle X\rangle$ following the procedure given in Theorem 5.3.12. Let $F = \{a\,X^2 - X + a\}$. The reduced Groebner basis G of F w.r.t. reverse lexicographic ordering is (trivially) $G = \{X^2 - \frac{1}{a}X + 1\}$. Then the polynomial $g \in G$ such that $g \in K\langle X\rangle$ where K is the fraction field of $\mathbb{Q}\langle\{a\}^{\oplus}\rangle$, and $g(r_1) \equiv 0$ is:

$$g(X) = X^2 - \frac{1}{a}X + 1 \ .$$

Note that this polynomial cannot be reduced into factors in the fraction field of $\mathbb{Q}\langle\{a\}^{\oplus}\rangle$. Multiplying g by a, we get $q(X) = aX^2 - X + a \in \mathbb{Q}\langle\{a\}^{\oplus}\rangle\langle X\rangle$ and we conclude that $q(X)$ is the irreducible polynomial described by Theorem 5.3.5. As $q(X)$ is not linear we conclude that $(\mathcal{G}, W)$ does not satisfy the Parikh property. ◀

Example 5.3.16. Now consider the WCFG given in Example 5.2.3 and its corresponding algebraic system S. We construct the irreducible polynomial $q(X_2) \in \mathbb{Q}\langle\{a, \overline{a}\}^{\oplus}\rangle\langle X_2\rangle$ following the procedure given in Theorem 5.3.12. Given F, the set of polynomials in the left-hand sides of the equations of S after moving all monomials from right to left, we construct the reduced Groebner basis G of F w.r.t. reverse lexicographic ordering. For clarity, we just show the polynomial $g \in G$ such that $g \in K\langle X_2\rangle$ where K is the fraction field of $\mathbb{Q}\langle\{a, \overline{a}\}^{\oplus}\rangle$, and verifies $g(r_1) \equiv 0$:

$$g(X_2) = X_2 - \frac{1}{1 - (a + \overline{a})} \ .$$

This polynomial is linear so it is irreducible over the fraction field of $\mathbb{Q}\langle\{a, \overline{a}\}^{\oplus}\rangle$. Now we multiply g by $(1 - (a + \overline{a}))$, and thus obtain $q(X_2) = (1 - (a + \overline{a}))X_2 - 1 \in \mathbb{Q}\langle\{a, \overline{a}\}^{\oplus}\rangle\langle X_2\rangle$ which is the irreducible polynomial described by Theorem 5.3.5. Now we apply the decision procedure described in Theorem 5.3.14. We observe that q can be written as follows:

$$q(X_2) = (1 - s)X_2 - t = (1 - (a + \overline{a}))X_2 - 1 \ ,$$

with $(s, \varepsilon) = 0$. Thus, we conclude that $(\mathcal{G}, W)$ satisfies the Parikh property. Finally, we give a regular Parikh-equivalent WCFG $(\mathcal{G}_\ell, W_\ell)$. The regular algebraic system:

$$(1 - (a + \overline{a}))X_2 - 1 = 0 \iff X_2 = (a + \overline{a})X_2 + 1 \tag{5.5}$$

has r_1 as solution. Then, the WCFG $(\mathcal{G}_\ell, W_\ell)$ corresponding to (5.5) is given by $\mathcal{G}_\ell = (\{X_2\}, \{a, \overline{a}\}, R_\ell, X_2)$ with R_ℓ defined as:

$$\begin{aligned}\pi_1 &= X_2 \to aX_2\\ \pi_2 &= X_2 \to \overline{a}X_2\\ \pi_3 &= X_2 \to \varepsilon\end{aligned}$$

and W_ℓ defined over $(\mathbb{Q}, +, \cdot, 0, 1)$ as $W_\ell(\pi_i) = 1$, for all i. Notice that $(\mathcal{G}_\ell, W_\ell)$ coincides with $(\mathcal{G}_1, W_1)$ in Example 5.2.3. ◀

Example 5.3.17. Let $(\mathcal{G}, W)$ be a WCFG with $\mathcal{G} = (\{X_1, X_2\}, \{a, b\}, R, X_1)$, R defined as:

$$\begin{aligned}X_1 &\to aX_2X_2\\ X_2 &\to bX_2 \mid a\ ,\end{aligned}$$

and the weight function W over $(\mathbb{Q}, +, \cdot, 0, 1)$ that assigns 1 to each production in the grammar. The algebraic system S corresponding to $(\mathcal{G}, W)$ is defined as follows:

$$\begin{cases}X_1 = a\,X_2^2\\ X_2 = b\,X_2 + a\ .\end{cases}$$

Let $\sigma = (r_1, r_2)$ be its strong solution. Now we construct the irreducible polynomial $q(X_1) \in \mathbb{Q}\langle\{a, b\}^{\oplus}\rangle\langle X_1\rangle$ following the procedure given in Theorem 5.3.12. Let $F = \{X_1 - a\,X_2^2, X_2 - b\,X_2 - a\}$. The reduced Groebner basis G of F w.r.t. lexicographic ordering is:

$$G = \left\{X_1 - \frac{a^3}{b^2 - 2b + 1}, X_2 + \frac{a}{b-1}\right\}\ .$$

Clearly, the polynomial $g \in G$ such that $g \in K\langle X_1\rangle$ where K is the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, and $g(r_1) \equiv 0$ is:

$$g(X_1) = X_1 - \frac{a^3}{b^2 - 2b + 1}\ .$$

This polynomial cannot be reduced into factors in the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$. By multiplying g by $(b^2 - 2b + 1)$ we obtain $q(X_1) = (b^2 - 2b + 1)X_1 - a^3$ in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle\langle X_1\rangle$ which is the irreducible polynomial described by Theorem 5.3.5. Now, we apply the decision procedure described in Theorem 5.3.14. We observe that q is linear in X_1 and can be written as:

$$q(X_1) = (1 - s)X_1 - t = (1 - (2b - b^2))X_1 - a^3\ ,$$

with $(s, \varepsilon) = 0$. Then we conclude that $(\mathcal{G}, W)$ satisfies the Parikh property. Note that this is the result expected as $(\mathcal{G}, W)$ is nonexpansive. Finally, we give a regular Parikh-equivalent WCFG $(\mathcal{G}_\ell, W_\ell)$. We know that the algebraic system:

$$X_1 = (2b - b^2)X_1 + a^3 \tag{5.6}$$

has r_1 as solution. Then the WCFG $(\mathcal{G}_\ell, W_\ell)$ corresponding to the regular system (5.6) is given by $\mathcal{G}_\ell = (\{X_1\}, \{a, b\}, R_\ell, X_1)$ with R_ℓ defined as:

$$\begin{aligned} \pi_1 &= X_1 \to bX_1 \\ \pi_2 &= X_1 \to b^2X_1 \\ \pi_3 &= X_1 \to a^3 \end{aligned}$$

and W_ℓ defined over $(\mathbb{Q}, +, \cdot, 0, 1)$ as:

$$W_\ell(\pi) = \begin{cases} 2 & \text{if } \pi = \pi_1 \\ -1 & \text{if } \pi = \pi_2 \\ 1 & \text{if } \pi = \pi_3 \end{cases} .$$

◀

5.4 Related Work

The problem of extending Parikh's Theorem to the weighted case has been significantly considered in the literature [10, 66, 68, 77].

Petre [77] establishes that the family of power series in commuting variables that can be generated by regular WCFGs is *strictly* contained in that of the series generated by arbitrary WCFGs. In this way, he shows that Parikh's Theorem does not hold in the weighted case.

It is well-known that the Parikh property holds in a commutative and idempotent semiring [10, 66, 68]. Luttenberger et al. [68] deal with WCFGs where the weight of a word corresponds to its ambiguity (or commutative ambiguity when considering monomials instead of words) and they show that if a CFG is nonexpansive then its commutative ambiguity can be expressed by a weighted rational expression, relying on the fact that all the parse trees of a nonexpansive CFG are of bounded dimension. We used this fact to give a Parikh-equivalent regular WCFG construction, for a given nonexpansive WCFG defined over *any* commutative semiring.

Baron and Kuich [8] gave a similar characterization of nonexpansive grammars using rational power series to that of Luttenberger et al. They

also conjectured that an unambiguous WCFG is nonexpansive iff it has the Parikh property. This conjecture appears to be false as evidenced by Example 5.2.3.

Bhattiprolu et al. [10] also show that the class of *polynomially ambiguous* WCFGs over the unary alphabet satisfies the property. In the unary case, this class is strictly contained in the class of nonexpansive grammars (a proof is given in Section 5.6.2).

Finally, our decision procedure relies on a result by Kuich and Saloma [67] that decides if an *algebraic* series in commuting variables with coefficients in $\mathbb{Q}$ is rational. To the best of our knowledge, the connection of this result to a decidability result for the Parikh property was only implicit.

5.5 Concluding Remarks

Note that from the theoretic point of view, our decision procedure can be applied to WCFGs over any arbitrary *field*. For arbitrary semirings, the decidability of the Parikh property remains open.

Finally, Theorem 5.2.2 shows an equivalent characterization of the Parikh property. Namely, the Parikh property holds for a WCFG $(\mathcal{G}, W)$ iff there exists a Parikh-equivalent nonexpansive WCFG, i.e., iff $(\mathcal{G}, W)$ is not *inherently* expansive. It is known that inherent expansiveness is undecidable in the noncommutative and unweighted case [47], but the question remains unsolved in the commutative case when weights are considered.

5.6 Supplementary Proofs

5.6.1 Proof of Theorem 5.2.2

First, let us introduce some useful definitions we will use throughout this section. Given a CFG $\mathcal{G} = (V, \Sigma, S, R)$, define the *degree* of $\mathcal{G}$ as $\max\{|\alpha\downarrow_V| : (X \to \alpha) \in R\} - 1$, where $\alpha\downarrow_V$ denotes the word that results from projecting α onto the variables V. For instance, given $\mathcal{G} = (\{X, Y\}, \{a, b\}, X, \{X \to aXYa, X \to a, Y \to b\})$, the degree of $\mathcal{G}$ is 1 since $|(aXYa)\downarrow_V| = |XY| = 2$, $|a\downarrow_V| = |\varepsilon| = 0$ and $|b\downarrow_V| = |\varepsilon| = 0$.

We will use ψ to denote derivation sequences. Given a derivation sequence $\psi = \alpha_1 \overset{\pi_1}{\Longrightarrow} \alpha_2 \overset{\pi_2}{\Longrightarrow} \ldots \overset{\pi_n}{\Longrightarrow} \alpha_n$, we call the derivation step $\alpha_{i-1} \overset{\pi_i}{\Longrightarrow} \alpha_i$ the i-step of the derivation sequence. A derivation sequence

$\psi = \alpha_1 \overset{\pi_1}{\Rightarrow} \alpha_2 \overset{\pi_2}{\Rightarrow} \ldots \overset{\pi_n}{\Rightarrow} \alpha_n$ has *index* j, denoted by $\mathrm{idx}(\psi)$, iff for every $i \in \{1, \ldots, n\}$, no word $\alpha_i|_V$ is longer than j. For instance, given the CFG $\mathcal{G}$ defined above, the derivation sequence $X \Rightarrow aXYa \Rightarrow aaXYaYa \Rightarrow aaaYaYa \Rightarrow aaababYa \Rightarrow aaababa$ has index 3. Finally, given a derivation sequence $\psi = \alpha_0 \Rightarrow \ldots \Rightarrow \alpha_n$ and β_0, β_1 (possibly empty) sequences of $(V \cup \Sigma)^*$, we will denote by $\beta_0\,\psi\,\beta_1$ the derivation sequence $\beta_0\,\alpha_0\,\beta_1 \Rightarrow \ldots \Rightarrow \beta_0\,\alpha_n\,\beta_1$

Now we present the proof of Theorem 5.2.2. All the definitions, lemmas and theorems used there will be presented below the proof.

Theorem 5.2.2. *Let $(\mathcal{G}, W)$ be an arbitrary WCFG. If $\mathcal{G}$ is nonexpansive then $(\mathcal{G}, W)$ satisfies the Parikh property.*

Proof. The proof is constructive. For every nonexpansive WCFG $(\mathcal{G}, W)$, we give a 2-step construction that results in a Parikh-equivalent regular WCFG $(\mathcal{G}_\ell, W_\ell)$. The steps are:

1. construct a new WCFG $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$, where $k \in \mathbb{N}$, language-equivalent to $(\mathcal{G}, W)$; and
2. construct a regular WCFG $(\mathcal{G}_\ell, W_\ell)$ Parikh-equivalent to $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$.

The first part of the construction consists in building a new WCFG $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ (Definition 5.6.1 below), so-called *at-most-k-dimension* WCFG of $(\mathcal{G}, W)$, which is language-equivalent to the original and where grammar variables are annotated with information about the dimension of the parse trees that can be obtained from these variables. Let us give an intuition on its construction.

For a given CFG $\mathcal{G}$ and $k \in \mathbb{N}$ (the choice of $k \in \mathbb{N}$ will be described later on), we define $\mathcal{G}^{\lceil k \rceil}$ using the same construction as Luttenberger et al. [68]. They show how to construct, for a given CFG $\mathcal{G}$, a new grammar $\mathcal{G}^{\lceil k \rceil}$ with the property that $\mathcal{T}_{\mathcal{G}^{\lceil k \rceil}}$ corresponds to the subset of $\mathcal{T}_{\mathcal{G}}$ of trees of dimension at most k. They annotate each grammar variable with the superscript $[d]$ (resp. $\lceil d \rceil$) to denote that only parse trees of dimension exactly d (resp. at most d), where $d \leq k$, can be obtained from these variables. When constructing the grammar, they also consider those rules containing two or more variables in its right-hand side and distinguish which cases yield an increase of dimension. We recall the construction of $\mathcal{G}^{\lceil k \rceil}$ in Definition 5.6.1.

To define the weight function $W^{\lceil k \rceil}$, we assign to each rule in $\mathcal{G}^{\lceil k \rceil}$ the same weight as its corresponding version in $\mathcal{G}$ (note that for those rules in $\mathcal{G}^{\lceil k \rceil}$ with no corresponding version in $\mathcal{G}$, i.e. the so-called e-rules, we assign the identity 1_A with respect to $\cdot$, where A denotes the weight domain).

Let us discuss the choice of k in $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$. Luttenberger et al. show that if $\mathcal{G}$ is a nonexpansive CFG then there exists an upper bound such that the dimension of every parse tree in $\mathcal{T}_{\mathcal{G}}$ is bounded by this value [68, Theorem 3.3]. Moreover, the bound is at most the number of grammar variables of $\mathcal{G}$. Then, for a given nonexpansive WCFG $(\mathcal{G}, W)$, we define k as this bound. Because k is at most equal to the number of variables of $\mathcal{G}$, such a value is always found and consequently, the first part of the construction always terminates.

Finally, we show that the WCFG $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ is language-equivalent to $(\mathcal{G}, W)$ (Lemma 5.6.2).

In the second part of the construction, we build a regular WCFG $(\mathcal{G}_\ell, W_\ell)$ that is Parikh-equivalent to $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$. Esparza et al. show that if the dimension of a parse tree is bounded by k then there exists a derivation sequence for the yield of the tree whose index is bounded by an affine function of k [32, Lemma 2.2]. We rely on this result to define a special derivation policy on at-most-k-dimension WCFGs, for which we know the dimension of every parse tree is bounded by k, called *lowest-dimension-first (LDF) derivations*. We prove that, for every WCFG $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$, the index of an LDF derivation sequence is bounded by an affine function of k (Lemma 5.6.4). Then, each grammar variable of $(\mathcal{G}_\ell, W_\ell)$ represents each possible sentence (without the terminals) along an LDF derivation sequence of $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$, and each grammar rule is intended to simulate an LDF derivation step of $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$. Because the number of variables in these sentences is bounded, the sets of variables and rules of $(\mathcal{G}_\ell, W_\ell)$ are necessarily finite. A formal definition of the weighted regular $(\mathcal{G}_\ell, W_\ell)$ is given in Definition 5.6.5 . Finally we show that $(\mathcal{G}_\ell, W_\ell)$ is Parikh-equivalent to $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ (Lemma 5.6.6) and this concludes the proof.

Now we build the at-most-k-dimension WCFG $(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil})$ for a given WCFG $(\mathcal{G}, W)$ and $k \in \mathbb{N}$. For the construction of $\mathcal{G}^{\lceil k \rceil}$, we rely on the one given by Luttenberger et al. [68].

Definition 5.6.1 (At-most-k-dimension WCFG)**.** Let $(\mathcal{G}, W)$ be a WCFG

with $\mathcal{G} = (V, \Sigma, S, R)$ and W defined over the commutative semiring $\mathbb{S}$, and let $k \in \mathbb{N}$. Define the *at-most-k-dimension WCFG* $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ with $\mathcal{G}^{\lceil k \rceil} = (V^{\lceil k \rceil}, \Sigma, S^{\lceil k \rceil}, R^{\lceil k \rceil})$ of $(\mathcal{G}, W)$ (with $w_0, \ldots, w_n \in \Sigma^*$) as follows:

- The set $V^{\lceil k \rceil}$ of variables is given by
$$\{X^{[d]}, X^{\lceil d \rceil} \mid X \in V, 0 \leq d \leq k\} \ .$$
- The set $R^{\lceil k \rceil}$ of production rules is given by
  1. Linear rules:
     - $r_0(\pi) \stackrel{\text{def}}{=} \{X^{[0]} \to w_0\}$ for each $\pi = (X \to w_0) \in R$.
     - $r_1(\pi) \stackrel{\text{def}}{=} \{X^{[d]} \to w_0\, Y^{[d]}\, w_1 \mid 0 \leq d \leq k\}$ for each $\pi = (X \to w_0\, Y\, w_1) \in R$.
  2. Non-linear rules:
     For each $\pi = (X \to w_0\, X_1\, w_1 \ldots w_{n-1}\, X_n\, w_n) \in R$
     - $r_2(\pi) \stackrel{\text{def}}{=} \{X^{[d]} \to w_0\, Z_1\, w_1 \ldots w_{n-1}\, Z_n\, w_n \mid 1 \leq d \leq k, J \subseteq \{1, \ldots, n\}$ with $|J| = 1 : Z_i = X_i^{[d]}$ if $i \in J$, and $Z_i = X_i^{\lceil d-1 \rceil}$ for all $i \in \{1, \ldots, n\} \setminus J\}$, and
     - $r_3(\pi) \stackrel{\text{def}}{=} \{X^{[d]} \to w_0\, Z_1\, w_1 \ldots w_{n-1}\, Z_n\, w_n \mid 1 \leq d \leq k, J \subseteq \{1, \ldots, n\}$ with $|J| \geq 2 : Z_i = X_i^{[d-1]}$ for all $i \in J$ and $Z_i = X_i^{\lceil d-1 \rceil}$ for all $i \in \{1, \ldots, n\} \setminus J\}$.
  3. e-rules:
     - $r_4 \stackrel{\text{def}}{=} \{X^{\lceil d \rceil} \to X^{[e]} \mid 0 \leq e \leq d \leq k\}$.
- The weight function $W^{\lceil k \rceil}$ is given by
$$W^{\lceil k \rceil}(\varphi) \stackrel{\text{def}}{=} \begin{cases} W(\pi) & \text{if } \varphi \in r_0(\pi) \text{ with } \pi = (X \to w_0) \in R \\ W(\pi) & \text{if } \varphi \in r_1(\pi) \text{ with } \pi = (X \to w_0\, X_1\, w_1) \in R \\ W(\pi) & \text{if } \varphi \in r_2(\pi) \cup r_3(\pi) \text{ with} \\ & \pi = (X \to w_0\, Z_1\, w_1, \ldots, w_{n-1}\, Z_n\, w_n) \in R \\ 1_A & \text{if } \varphi \in r_4 \end{cases} .$$

We say that a variable $Z \in V^{\lceil k \rceil}$ is *of dimension* d iff either $Z = X^{\lceil d \rceil}$, or $Z = X^{[d]}$, with $X \in V$.

Lemma 5.6.2. $[\![\mathcal{G}]\!]_W = [\![\mathcal{G}^{\lceil k \rceil}]\!]_{W^{\lceil k \rceil}}$.

Proof. First recall that k corresponds to the nonnegative value such that every parse tree τ in $\mathcal{G}$ has dimension at most k, which we denote by $d(\tau) \leq k$. We want to show that there is a bijection μ from $\mathcal{T}_{\mathcal{G}^{\lceil k \rceil}}$ to $\mathcal{T}_G$ that preserves the yield and the weight of each parse tree.

First, define $\mathcal{T}_{\mathcal{G}}^{\leq k} \stackrel{\text{def}}{=} \{\tau \mid \tau \in \mathcal{T}_{\mathcal{G}}, d(\tau) \leq k\}$. Luttenberger et al. [68] prove that there is a bijection μ from $\mathcal{T}_{\mathcal{G}^{\lceil k \rceil}}$ to $\mathcal{T}_{\mathcal{G}}^{\leq k}$ that preserves the yield of parse trees. Roughly speaking, μ contracts the edges corresponding to the e-rules and removes the superscripts from the labels of the trees. Note that $X^{\lceil d \rceil}$ can only be rewritten to $X^{[e]}$ for some $e \leq d$. Then, contracting the corresponding edges cannot change the yield of the corresponding tree. Furthermore, the rules of $\mathcal{G}^{\lceil k \rceil}$ that rewrite the variable $X^{[d]}$ are obtained from the rules of $\mathcal{G}$ that rewrite X by only adding a superscript. Hence, by removing these annotations again, every tree $\tau \in \mathcal{T}_{\mathcal{G}^{\lceil k \rceil}}$ is mapped by μ to a tree in $\mathcal{T}_{\mathcal{G}}^{\leq k}$ with the same yield. The complete proof of this fact is in [68, Lemma 3.2].

Furthermore, because $\mathcal{G}$ is nonexpansive, we have that $\mathcal{T}_{\mathcal{G}}^{\leq k} = \mathcal{T}_{\mathcal{G}}$ [68]. Thus, if $\mathcal{G}$ is nonexpansive, then μ is a bijection from $\mathcal{T}_{\mathcal{G}^{\lceil k \rceil}}$ to $\mathcal{T}_{\mathcal{G}}$ that preserves the yield of parse trees.

Now we show that μ also preserves the weights of parse trees, i.e., for each $\tau \in \mathcal{T}_{\mathcal{G}^{\lceil k \rceil}} : W^{\lceil k \rceil}(\tau) = W(\mu(\tau))$. We proceed by induction on the number of nodes of τ. In the base case, τ has one node, i.e., it has no children. Then $\tau = \varphi$ with $\varphi = X^{[0]} \to w_0$ and $w_0 \in \Sigma^*$, and $\mu(\tau) = \varphi'$ with $\varphi' = X \to w_0$. Then we have:

$$\begin{aligned} W^{\lceil k \rceil}(\tau) &= W^{\lceil k \rceil}(\varphi) && \tau = \varphi \\ &= W(\varphi') && \text{by def. of } W^{\lceil k \rceil} \\ &= W(\mu(\tau)) && \mu(\tau) = \varphi' \end{aligned}$$

For the induction step, assume $\tau = \varphi(\tau_1, \ldots, \tau_n)$ with $n \geq 1$ and φ is a rule from the set r_i with $i \in \{1, \ldots, 5\}$ (see Definition 5.6.1). We distinguish three cases:

- Assume $\varphi \in r_1$. Then $\tau = \varphi(\tau_1)$ and $\mu(\tau) = \varphi'(\,\mu(\tau_1))$, with

$\varphi' = X \to w_0\, X_1\, w_1$ and $w_0, w_1 \in \Sigma^*$.

$$\begin{aligned} W^{\lceil k \rceil}(\tau) &= W^{\lceil k \rceil}(\varphi) \cdot W^{\lceil k \rceil}(\tau_1) && \text{by def. of weight of } \tau \\ &= W(\varphi') \cdot W(\mu(\tau_1)) && \text{by def. of } W^{\lceil k \rceil} \text{ and induction hyp.} \\ &= W(\mu(\tau)) && \mu(\tau) = \varphi'(\mu(\tau_1)) \end{aligned}$$

- Assume $\varphi \in r_4$. Then $\tau = \varphi(\tau_1)$ and $\mu(\tau) = \mu(\tau_1)$.

$$\begin{aligned} W^{\lceil k \rceil}(\tau) &= W^{\lceil k \rceil}(\varphi) \cdot W^{\lceil k \rceil}(\tau_1) && \tau = \varphi(\tau_1) \\ &= 1_{\mathbb{S}} \cdot W(\mu(\tau_1)) && \text{by induction hyp. and def. of } W^{\lceil k \rceil} \\ &= W(\mu(\tau)) && \mu(\tau) = \mu(\tau_1) \end{aligned}$$

- Assume $\varphi \in r_2 \cup r_3$. Then $\tau = \varphi(\tau_1, \ldots, \tau_n)$ and $\mu(\tau) = \varphi'(\mu(\tau_1), \ldots, \mu(\tau_n))$ with $\varphi' = X \to w_0\, X_1\, w_1 \ldots w_{n-1}\, X_n\, w_n$ and $w_0, \ldots, w_n \in \Sigma^*$.

$$\begin{aligned} W^{\lceil k \rceil}(\tau) &= W^{\lceil k \rceil}(\varphi) \prod_{i=1}^{n} W^{\lceil k \rceil}(\tau_i) && \tau = \varphi(\tau_1, \ldots, \tau_n) \\ &= W(\varphi') \prod_{i=1}^{n} W(\mu(\tau_i)) && \text{by def. of } W^{\lceil k \rceil} \text{ and induction hyp.} \\ &= W(\mu(\tau)) && \mu(\tau) = \varphi'(\mu(\tau_1), \ldots, \mu(\tau_n)) \end{aligned}$$

Finally, for each $w \in \Sigma^*$:

$$[\![\mathcal{G}]\!]_W(w) = \sum_{\substack{w = \mathcal{Y}(\tau) \\ \tau \in \mathcal{T}_{\mathcal{G}}}} W(\tau) = \sum_{\substack{w = \mathcal{Y}(\tau') \\ \tau' \in \mathcal{T}_{\mathcal{G}^{\lceil k \rceil}}}} W^{\lceil k \rceil}(\tau') = W^{\lceil k \rceil}(w) = [\![\mathcal{G}^{\lceil k \rceil}]\!]_{W^{\lceil k \rceil}}(w) \ .$$

Now we define a derivation policy over at-most-k-dimension WCFGs. We call these derivations *lowest-dimension-first* (LDF) derivations.

Intuitively, given a parse tree τ of an at-most-k-dimension WCFG, we define the LDF derivation sequence of τ by performing a depth-first traversal of τ where nodes in the same level of the tree are visited from lower to greater dimension and, if more than one node has the same dimension, then from left to right. Recall that the dimension of a node corresponds to the dimension of the parse tree that it roots.

Definition 5.6.3 (Lowest-dimension-first derivation)**.** Let $\mathcal{G}^{\lceil k \rceil}$ be an at-most-k-dimension CFG as in Definition 5.6.1. Let $\tau = \pi(\tau_1, \ldots, \tau_n)$ be a parse tree of $\mathcal{G}^{\lceil k \rceil}$. Define the *lowest-dimension-first* (LDF) derivation sequence ψ of τ inductively as follows:

- If $n = 0$, then π is of the form $\pi = X^{[0]} \to w_0$, and $\tau = \pi$. Then, the LDF derivation sequence of τ is:

$$\psi = X^{[0]} \Rightarrow^{\pi}_{\text{ldf}} w_0 \ .$$

- If $n \geq 1$, we distinguish the following cases:

  1. If $\pi \in r_1$, i.e., π is of the form $\pi = X^{[d]} \to w_0 X_1^{[d]} w_1$ with $0 \leq d \leq k$, and $\tau = \pi(\tau_1)$. Then, the LDF derivation sequence of τ is:
  $$\psi = X^{[d]} \Rightarrow^{\pi}_{\text{ldf}} w_0 X_1^{[d]} w_1 \Rightarrow_{\text{ldf}} w_0 \psi_1 w_1 \ ,$$
  where ψ_1 is the LDF derivation sequence of τ_1.

  2. If $\pi \in r_4$, i.e., π is of the form $\pi = X^{\lceil d \rceil} \to X^{[e]}$ with $0 \leq e \leq d \leq k$, and $\tau = \pi(\tau_1)$. Then, the LDF derivation sequence of τ is:
  $$\psi = X^{\lceil d \rceil} \Rightarrow^{\pi}_{\text{ldf}} X^{[e]} \Rightarrow_{\text{ldf}} \psi_1 \ ,$$
  where ψ_1 is the LDF derivation sequence of τ_1.

  3. If $\pi \in r_2$, w.l.o.g., we assume that π is of the form:
  $$\pi = X^{[d]} \to w_0 X_1^{[d]} w_1 X_2^{\lceil d-1 \rceil} w_2 \ldots w_{n-2} X_{n-1}^{\lceil d-1 \rceil} w_{n-1} X_n^{\lceil d-1 \rceil} w_n \ ,$$
  with $1 \leq d \leq k$, and $\tau = \pi(\tau_1, \ldots, \tau_n)$. Define, for each $i \in \{2, \ldots, n\}$, the derivation sequence $\tilde{\psi}_i$ as follows:
  $$\begin{aligned} \tilde{\psi}_i &\stackrel{\text{def}}{=} w_0 \, X_1^{[d]} \, w_1 \, \mathcal{Y}(\tau_2) \, w_2 \, \ldots \, \mathcal{Y}(\tau_{i-1}) \, w_{i-1} \, X_i^{\lceil d-1 \rceil} \, w_i \, \ldots \\ &\quad w_{n-1} \, X_n^{\lceil d-1 \rceil} w_n \\ &\quad \Rightarrow^{*}_{\text{ldf}} w_0 \, X_1^{[d]} \, w_1 \ldots \mathcal{Y}(\tau_{i-1}) \, w_{i-1} \, \psi_i \, w_i \, X_{i+1}^{\lceil d-1 \rceil} \, w_{i+1} \ \ldots \\ &\quad w_{n-1} \, X_n^{\lceil d-1 \rceil} w_n \ , \end{aligned}$$
  where ψ_i is the LDF derivation sequence of τ_i. And define:
  $$\begin{aligned} \tilde{\psi}_1 &\stackrel{\text{def}}{=} w_0 \, X_1^{[d]} \, w_1 \, \mathcal{Y}(\tau_2) \, w_2 \, \ldots \, w_{n-1} \, \mathcal{Y}(\tau_n) \, w_n \\ &\quad \Rightarrow^{*}_{\text{ldf}} w_0 \, \psi_1 \, w_1 \, \mathcal{Y}(\tau_2) \, w_2 \, \ldots \, w_{n-1} \, \mathcal{Y}(\tau_n) \, w_n \ , \end{aligned}$$

where ψ_1 is the LDF derivation sequence of τ_1. Then the LDF derivation ψ of τ is:

$$\psi = X^{[d]} \Rightarrow^{\pi}_{\text{ldf}} \tilde{\psi}_2 \Rightarrow_{\text{ldf}} \ldots \Rightarrow_{\text{ldf}} \tilde{\psi}_n \Rightarrow_{\text{ldf}} \tilde{\psi}_1 \ .$$

4. If $\pi \in r_3$, w.l.o.g., we assume that π is of the form:

   $$\pi = X^{[d]} \to w_0 X_1^{\lceil d-1 \rceil} w_1 X_2^{\lceil d-1 \rceil} w_2 \ldots w_{n-2} X_{n-1}^{[d-1]} w_{n-1} X_n^{[d-1]} w_n \ ,$$

   with $1 \leq d \leq k$, and $\tau = \pi(\tau_1, \ldots, \tau_n)$. Define, for each $i \in \{1, \ldots, n\}$, the derivation sequence $\tilde{\psi}_i$ as follows:

   $$\begin{aligned} \tilde{\psi}_i &\stackrel{\text{def}}{=} w_0\, \mathcal{Y}(\tau_1)\, w_1\, \mathcal{Y}(\tau_2)\, w_2 \ \ldots\ \mathcal{Y}(\tau_{i-1})\, w_{i-1}\, X_i^{\lceil d-1 \rceil}\, w_i \ \ldots \\ &\quad w_{n-1}\, X_n^{[d-1]}\, w_n \\ &\quad \Rightarrow^{*}_{\text{ldf}} w_0\, \mathcal{Y}(\tau_1)\, w_1 \ldots \mathcal{Y}(\tau_{i-1})\, w_{i-1}\, \psi_i\, w_i\, X_{i+1}^{\lceil d-1 \rceil}\, w_{i+1} \ \ldots \\ &\quad w_{n-1}\, X_n^{[d-1]}\, w_n \ , \end{aligned}$$

   where ψ_i is the LDF derivation sequence of τ_i. Then the LDF derivation ψ of τ is:

   $$\psi = X^{[d]} \Rightarrow_{\text{ldf}} \tilde{\psi}_1 \Rightarrow_{\text{ldf}} \ldots \Rightarrow_{\text{ldf}} \tilde{\psi}_n \ .$$

Note that, given a parse tree τ of $\mathcal{G}^{\lceil k \rceil}$, the LDF derivation sequence of τ is uniquely defined.

Lemma 5.6.4. *Let $\mathcal{G}^{\lceil k \rceil}$ be an at-most-k-dimension CFG of degree m and $\tau \in \mathcal{T}_{\mathcal{G}^{\lceil k \rceil}}$ such that $d(\tau) \leq k$. Then, the LDF derivation sequence of τ verifies* $\text{idx}(\psi) \leq km + 1$.

Proof. Let $\mathcal{G}^{\lceil k \rceil} = (V^{\lceil k \rceil}, \Sigma, S^{\lceil k \rceil}, R^{\lceil k \rceil})$. We prove the more general statement:

Let m be the degree of $\mathcal{G}^{\lceil k \rceil}$ and let $\tau \in \mathcal{T}_{\mathcal{G}^{\lceil k \rceil}}$ such that $d(\tau) \leq d$. (5.7)

Then, the LDF derivation sequence ψ of τ satisfies $\text{idx}(\psi) \leq dm + 1$. (5.8)

The proof goes by induction on the number of nodes of τ. In the base case, τ has one node, i.e., it has no children. Then, $d = 0$ and the LDF derivation of τ is $\psi = X^{[0]} \Rightarrow^{\pi}_{\text{ldf}} w_0$ with $\pi = (X^{[0]} \to w_0) \in R^{\lceil k \rceil}$.

Clearly, the index of ψ is 1.

For the induction step, assume that $\tau = \pi(\tau_1, \ldots, \tau_n)$ with $n \geq 1$. We split the proof into the following four cases:

- If $\pi \in r_1$, then π is of the form $\pi = X^{[d]} \to w_0 X_1^{[d]} w_1$ with $0 \leq d \leq k$, and $\tau = \pi(\tau_1)$ with $d(\tau) \leq d$. By induction hypothesis, the LDF derivation sequence ψ_1 of τ_1 verifies $\mathrm{idx}(\psi_1) \leq (dm + 1)$. Then, the LDF derivation of τ is:

$$\psi = X^{[d]} \Rightarrow_{\mathrm{ldf}} w_0 X_1^{[d]} w_1 \Rightarrow_{\mathrm{ldf}}^{*} w_0 \psi_1 w_1,$$

  and verifies $\mathrm{idx}(\psi) \leq dm + 1$.

- If $\pi \in r_4$, then π is of the form $\pi = X^{\lceil d \rceil} \to X^{[e]}$ with $0 \leq e \leq d \leq k$, and $\tau = \pi(\tau_1)$ with $d(\tau) \leq d$. By induction hypothesis, the LDF derivation sequence ψ_1 of τ_1 s.t. $\mathrm{idx}(\psi_1) \leq (em + 1)$. Then, the LDF derivation of τ is:

$$\psi = X^{\lceil d \rceil} \Rightarrow_{\mathrm{ldf}} X^{[e]} \Rightarrow_{\mathrm{ldf}}^{*} \psi_1,$$

  and verifies $\mathrm{idx}(\psi) \leq em + 1 \leq dm + 1$.

- If $\pi \in r_2$, then, w.l.o.g., π is of the form:

$$\pi = X^{[d]} \to w_0 X_1^{[d]} w_1 X_2^{\lceil d-1 \rceil} w_2 \ldots w_{n-2} X_{n-1}^{\lceil d-1 \rceil} w_{n-1} X_n^{\lceil d-1 \rceil} w_n,$$

  with $1 \leq d \leq k$, and $\tau = \pi(\tau_1, \ldots, \tau_n)$ with $d(\tau) \leq d$. By induction hypothesis, for each $i \in \{2, \ldots, n\}$, there is a derivation ψ_i of τ_i s.t. $\mathrm{idx}(\psi_i) \leq ((d-1)m + 1)$, and there is a derivation ψ_1 for τ_1 s.t. $\mathrm{idx}(\psi_1) \leq dm + 1$. Now, define, for each $i \in \{2, \ldots, n\}$, the derivation sequence $\tilde{\psi}_i$ as follows:

$$\begin{aligned}
\tilde{\psi}_i \stackrel{\mathrm{def}}{=}\ & w_0\, X_1^{[d]}\, w_1\, \mathcal{Y}(\tau_2)\, w_2\, \ldots\, \mathcal{Y}(\tau_{i-1})\, w_{i-1}\, X_i^{\lceil d-1 \rceil}\, w_i \ldots \\
& w_{n-1}\, X_n^{\lceil d-1 \rceil} w_n \\
& \Rightarrow_{\mathrm{ldf}}^{*} w_0\, X_1^{[d]}\, w_1\, \mathcal{Y}(\tau_2)\, w_2\, \ldots\, \mathcal{Y}(\tau_{i-1})\, w_{i-1}\, \psi_i\, w_i\, X_{i+1}^{\lceil d-1 \rceil}\, w_{i+1} \ldots \\
& w_{n-1}\, X_n^{\lceil d-1 \rceil} w_n\ .
\end{aligned}$$

And define:

$$\begin{aligned}\tilde{\psi_1} &\stackrel{\text{def}}{=} w_0\, X_1^{[d]}\, w_1\, \mathcal{Y}(\tau_2)\, w_2\, \ldots\, w_{n-1}\, \mathcal{Y}(\tau_n)\, w_n \\ &\Rightarrow^*_{\text{ldf}} w_0\, \psi_1\, w_1\, \mathcal{Y}(\tau_2)\, w_2\, \ldots\, w_{n-1}\, \mathcal{Y}(\tau_n)\, w_n \ .\end{aligned}$$

Then, the LDF derivation ψ of τ is:

$$\psi = X^{[d]} \Rightarrow^{\pi}_{\text{ldf}} \tilde{\psi_2} \Rightarrow_{\text{ldf}} \ldots \Rightarrow_{\text{ldf}} \tilde{\psi_n} \Rightarrow_{\text{ldf}} \tilde{\psi_1} \ .$$

Observe that $n-1 \leq m$ where n is the number of variables occurring in the right-hand side of π and m is the degree of $\mathcal{G}^{\lceil k \rceil}$. For each $i \in \{2, \ldots, n\}$, the index of $\tilde{\psi_i}$ is at most $(d-1)m + 1 + (n-i) \leq dm + 1$. On the other hand, the index of $\tilde{\psi_1}$ is at most $dm+1$. Then, performing the derivation steps of ψ in the order shown above we have that $\text{idx}(\psi) \leq dm + 1$.

- If $\pi \in r_3$, then, w.l.o.g., π is of the form:

$$\pi = X^{[d]} \to w_0 X_1^{\lceil d-1 \rceil} w_1 X_2^{\lceil d-1 \rceil} w_2 \ldots w_{n-2} X_{n-1}^{[d-1]} w_{n-1} X_n^{[d-1]} w_n,$$

with $1 \leq d \leq k$, and $\tau = \pi(\tau_1, \ldots, \tau_n)$ with $d(\tau) \leq d$. By induction hypothesis, for each $i \in \{1, \ldots, n\}$, there is a derivation ψ_i of τ_i s.t. $\text{idx}(\psi_i) \leq ((d-1)m + 1)$. Now, define, for each $i \in \{1, \ldots, n\}$, the derivation sequence $\tilde{\psi_i}$ as follows:

$$\begin{aligned}&\tilde{\psi_i} \stackrel{\text{def}}{=} w_0\, \mathcal{Y}(\tau_1)\, w_1\, \mathcal{Y}(\tau_2)\, w_2\, \ldots\, \mathcal{Y}(\tau_{i-1})\, w_{i-1}\, X_i^{\lceil d-1 \rceil}\, w_i \ldots \\ &\quad w_{n-1}\, X_n^{[d-1]}\, w_n \\ &\quad \Rightarrow^*_{\text{ldf}} w_0\, \mathcal{Y}(\tau_1)\, w_1\, \mathcal{Y}(\tau_2)\, w_2\, \ldots\, \mathcal{Y}(\tau_{i-1})\, w_{i-1}\, \psi_i\, w_i\, X_{i+1}^{\lceil d-1 \rceil}\, w_{i+1} \ldots \\ &\quad w_{n-1}\, X_n^{[d-1]}\, w_n \ .\end{aligned}$$

Then, the LDF derivation ψ of τ is:

$$\psi = X^{[d]} \Rightarrow^{\pi}_{\text{ldf}} \tilde{\psi_1} \Rightarrow_{\text{ldf}} \ldots \Rightarrow_{\text{ldf}} \tilde{\psi_n} \ .$$

For each $i \in \{1, \ldots, n\}$, the index of $\tilde{\psi_i}$ is at most $(d-1)m + 1 + (n-i) \leq dm + 1$. It follows that $\text{idx}(\psi) \leq dm + 1$.

Let us define the set of variables $V^{(d)} \stackrel{\text{def}}{=} \{X^{\lceil d \rceil}, X^{[d]} \in V^{\lceil k \rceil} \mid X \in V\}$, for each $0 \leq d \leq k$. Thus, given a derivation sentence $\alpha \in \left(\Sigma \cup V^{\lceil k \rceil}\right)^*$ of

an at-most-k-dimension CFG, define $\mathrm{ldf}(\alpha) \stackrel{\text{def}}{=} \alpha\downarrow_{\Sigma} \alpha\downarrow_{V^{(0)}} \alpha\downarrow_{V^{(1)}} \ldots \alpha\downarrow_{V^{(k)}}$ and $\mathrm{ldf}_{V^{\lceil k \rceil}}(\alpha) \stackrel{\text{def}}{=} (\mathrm{ldf}(\alpha))\downarrow_{V^{\lceil k \rceil}}$. Using this notation, we will define a regular $(\mathcal{G}_\ell, W_\ell)$ that is Parikh-equivalent to $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ in a similar way to Bhattiprolu et al. [10].

Definition 5.6.5 (Regular WCFG for $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$)**.** Let $\left(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ be an at-most-k-dimension WCFG with $\mathcal{G}^{\lceil k \rceil} = (V^{\lceil k \rceil}, \Sigma, S^{\lceil k \rceil}, R^{\lceil k \rceil})$ and degree m, and $W^{\lceil k \rceil}$ defined over the commutative semiring $\mathbb{S}$. Define the WCFG $(\mathcal{G}_\ell, W_\ell)$ with $\mathcal{G}_\ell = (V_\ell, \Sigma, S_\ell, R_\ell)$ as follows:

- Each variable in V_ℓ corresponds to a sequence $\alpha \in \left(V^{\lceil k \rceil}\right)^{km+1}$ where $\left(V^{\lceil k \rceil}\right)^{km+1}$ denotes the set $\{w \mid w \in \left(V^{\lceil k \rceil}\right)^*, |w| \leq km+1\}$. We will denote each variable by $\langle\alpha\rangle$. Formally:

$$V_\ell \stackrel{\text{def}}{=} \{\langle\alpha\rangle \mid \alpha \in \left(V^{\lceil k \rceil}\right)^{km+1}\} \ .$$

- The initial variable is defined as $S_\ell \stackrel{\text{def}}{=} S^{\lceil k \rceil}$.

- For each rule $\pi = (X \to \beta) \in R^{\lceil k \rceil}$ define:

$$\pi^{\alpha} \stackrel{\text{def}}{=} (\langle X\,\alpha\rangle \to \beta\downarrow_{\Sigma} \langle \mathrm{ldf}_{V^{\lceil k \rceil}}(\beta)\,\alpha\rangle) \ .$$

  The set R_ℓ of rules is given by:

$$\{\pi^{\alpha} \mid \pi = (X \to \beta) \in R^{\lceil k \rceil} \text{ and } \langle X\alpha\rangle, \langle \mathrm{ldf}_{V^{\lceil k \rceil}}(\beta)\,\alpha\rangle \in V_\ell\} \ .$$

- The weight function W_ℓ is given by:

$$W_\ell(\pi^{\alpha}) \stackrel{\text{def}}{=} W^{\lceil k \rceil}(\pi) \text{ for all } \pi^{\alpha} \in R_\ell \ .$$

Lemma 5.6.6. $\wr\mathcal{G}^{\lceil k \rceil}\int_{W^{\lceil k \rceil}} = \wr\mathcal{G}_\ell\int_{W_\ell}$.

> ***Proof.*** For convenience, we will give an alternative definition of the weight of a word using derivation sequences. Recall that we assume that the derivation policy of a grammar defines for each parse tree one unique derivation sequence. Given a CFG $\mathcal{G}$ and $w \in \Sigma^*$, denote by $\mathrm{der}_{\mathcal{G}}(w)$ (or simply $\mathrm{der}(w)$) the subset of all derivations of $\mathcal{G}$ that yield to $w \in \Sigma^*$. Then, define for each derivation sequence $\psi =$

$\alpha_0 \Rightarrow^{\pi_1} \alpha_1 \Rightarrow^{\pi_2} \ldots \Rightarrow^{\pi_n} \alpha_n$ of $\mathcal{G}$ the *weight of* ψ as follows:

$$W(\psi) \stackrel{\text{def}}{=} \prod_{i=1}^{n} W(\pi_i) \ .$$

Finally, define for each $w \in \Sigma^*$,

$$W(w) \stackrel{\text{def}}{=} \sum_{\psi \in \text{der}(w)} W(\psi) \ .$$

If $\text{der}(w) = \varnothing$ then $W(w) \stackrel{\text{def}}{=} 0_A$.

We claim that there exists a one-to-one correspondence f that maps each LDF derivation sequence of $(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil})$ into a derivation sequence of $(\mathcal{G}_\ell, W_\ell)$ that preserves the Parikh images and the weights between derivations. Formally, there exists a one-to-one correspondence f such that for each LDF derivation sequence $\psi = X^{[d]} \Rightarrow^*_{\text{ldf}} w$ with $w \in \Sigma^*$ of $(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil})$, $f(\psi) = \langle X^{[d]} \rangle \Rightarrow^* w'$ with $w' \in \Sigma^*$ is a derivation sequence of $(\mathcal{G}_\ell, W_\ell)$ with the following properties:

1. $\wr w \int = \wr w' \int$ and,
2. $W^{\lceil k \rceil}(\psi) = W_\ell(f(\psi))$.

We now give an inductive definition of f. Along this definition we will prove inductively that: (i) f is an injective function from LDF derivation sequences of $(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil})$ to derivation sequences in $(\mathcal{G}_\ell, W_\ell)$; and (ii) properties 1. and 2. above hold.

Let ψ be an LDF derivation of $(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil})$.

1. If ψ is a 1-step derivation sequence then $\psi = X^{[0]} \Rightarrow^{\pi}_{\text{ldf}} w_0$ with $\pi \in r_0$. Then, define $f(\psi) \stackrel{\text{def}}{=} \langle X^{[0]} \rangle \Rightarrow^{\pi^\varepsilon} w_0$.

   Note that $f(\psi)$ is a one-step derivation sequence that uses the rule $(\langle X^{[0]} \rangle \to w_0) \in R_\ell$. It follows that f defines uniquely a derivation sequence of $(\mathcal{G}_\ell, W_\ell)$ for ψ. Note that property 1. holds trivially. By definition of $(\mathcal{G}_\ell, W_\ell)$, we have that:

   $$W_\ell(f(\psi)) = W_\ell(\langle X^{[0]} \rangle \to w_0) = W^{\lceil k \rceil}(X^{[0]} \to w_0) = W^{\lceil k \rceil}(\psi) \ .$$

2. If ψ is a n-step derivation sequence (with $n > 1$), then we have the following cases:

   - If $\psi = X^{[d]} \Rightarrow^{\pi}_{\text{ldf}} w_0 X_1^{[d]} w_1 \Rightarrow^{*}_{\text{ldf}} w_0 \psi' w_1$ where $\pi \in r_1$ and $\psi' = X_1^{[d]} \Rightarrow^{*}_{\text{ldf}} w$ with $w \in \Sigma^*$. Then, define $f(\psi) \stackrel{\text{def}}{=} \langle X^{[d]} \rangle \Rightarrow^{\pi^\varepsilon} w_0 w_1 \langle X_1^{[d]} \rangle \Rightarrow^{*} w_0 w_1 f(\psi')$.
     Note that the first step in the derivation $f(\psi)$ uses the rule $(\langle X^{[d]} \rangle \to w_0 w_1 \langle X_1^{[d]} \rangle) \in R_\ell$. Relying on this and the hypothesis of induction, f defines uniquely a derivation sequence of $(\mathcal{G}_\ell, W_\ell)$ for ψ. By hypothesis of induction, it is easy to check that property 1. holds. Finally, using the hypothesis of induction and the definition of $(\mathcal{G}_\ell, W_\ell)$ we have:

$$\begin{aligned} W_\ell(f(\psi)) &= W_\ell(\langle X^{[d]} \rangle \to w_0 w_1 \langle X_1^{[d]} \rangle) \cdot W_\ell(f(\psi')) \\ &= W^{\lceil k \rceil}(X^{[d]} \to w_0 X_1^{[d]} w_1) \cdot W^{\lceil k \rceil}(\psi') = W^{\lceil k \rceil}(\psi) \ . \end{aligned}$$

   - If $\psi = X^{\lceil d \rceil} \Rightarrow^{\pi}_{\text{ldf}} X^{[e]} \Rightarrow^{*}_{\text{ldf}} \psi'$ where $\pi \in r_2$ and $\psi' = X_1^{[e]} \Rightarrow^{*}_{\text{ldf}} w$ with $w \in \Sigma^*$. Then, define $f(\psi) \stackrel{\text{def}}{=} \langle X^{\lceil d \rceil} \rangle \Rightarrow^{\pi^\varepsilon} \langle X^{[e]} \rangle \Rightarrow^{*} f(\psi')$.
     Note that the first step in the derivation $f(\psi)$ uses the rule $(\langle X^{\lceil d \rceil} \rangle \to \langle X^{[e]} \rangle) \in R_\ell$. Relying on this and the hypothesis of induction, f defines uniquely a derivation sequence of $(\mathcal{G}_\ell, W_\ell)$ for ψ. By hypothesis of induction, property 1. holds trivially. Finally, using the hypothesis of induction and the definition of $(\mathcal{G}_\ell, W_\ell)$ we have:

$$\begin{aligned} W_\ell(f(\psi)) &= W_\ell(\langle X^{\lceil d \rceil} \rangle \to \langle X^{[e]} \rangle) \cdot W_\ell(f(\psi')) \\ &= W^{\lceil k \rceil}(X^{\lceil d \rceil} \to X^{[e]}) \cdot W^{\lceil k \rceil}(\psi') \\ &= W^{\lceil k \rceil}(\psi) \ . \end{aligned}$$

     In the next (and last) case, the derivation sentence $\beta\alpha\gamma$ will denote $\beta w \langle \alpha' \gamma' \rangle$ when $\gamma = \langle \gamma' \rangle$ and $\alpha = w \langle \alpha' \rangle$ with $\beta', \alpha' \in \left(V^{\lceil k \rceil}\right)^{km+1}$ and $w \in \Sigma^*$.

- Finally, assume w.l.o.g, that ψ has the form:

$$\begin{aligned}\psi = X^{[d]} &\Rightarrow^{\pi}_{\text{ldf}} w_0 Z_1 w_1 \dots w_{n-2} Z_{n-1} w_{n-1} Z_n w_n \\ &\Rightarrow^{*}_{\text{ldf}} w_0 Z_1 w_1 \dots w_{n-2} Z_{n-1} w_{n-1} \psi'_n w_n \\ &\Rightarrow^{*}_{\text{ldf}} w_0 Z_1 w_1 \dots w_{n-2} \psi'_{n-1} w_{n-1} \widetilde{w}_n w_n \\ &\Rightarrow^{*}_{\text{ldf}} \cdots \\ &\Rightarrow^{*}_{\text{ldf}} w_0 \widetilde{w}_1 w_1 \widetilde{w}_2 w_2 \dots w_{n-1} \widetilde{w}_n w_n \ ,\end{aligned}$$

where $\pi \in r_2 \cup r_3$ and, for each $i \in \{1, \dots, n\}$, $\psi'_i = Z_i \Rightarrow^{*}_{\text{ldf}} \widetilde{w}_i$ with $\widetilde{w}_i \in \Sigma^*$ for all i. Then, define:

$$\begin{aligned}f(\psi) \stackrel{\text{def}}{=} \langle X^{[d]} \rangle &\Rightarrow^{\pi^{\varepsilon}} w_0 w_1 w_2 \dots w_n \langle \text{ldf}(Z_1 \dots Z_n) \rangle \\ &\Rightarrow^{*} w_0 w_1 w_2 \dots w_n f(\psi'_n) \langle \text{ldf}(Z_1 \dots Z_{n-1}) \rangle \\ &\Rightarrow^{*} w_0 w_1 w_2 \dots w_n \widetilde{w'}_n f(\psi'_{n-1}) \langle \text{ldf}(Z_1 \dots Z_{n-2}) \rangle \\ &\Rightarrow^{*} \dots \\ &\Rightarrow^{*} w_0 w_1 w_2 \dots w_n \widetilde{w'}_n \widetilde{w'}_{n-1} \dots \widetilde{w'}_1 \ ,\end{aligned}$$

where each $w'_i \in \Sigma^*$ corresponds to the word generated by each $f(\psi'_i)$ inductively. Note that the first step in the derivation $f(\psi)$ uses a rule of the form $(X^{[d]} \to w_0 w_1 w_2 \dots w_n \langle \text{ldf}(Z_1 \dots Z_n) \rangle)$ which, according to the definition of $(\mathcal{G}_\ell, W_\ell)$, defines a rule in R_ℓ. Relying on this and the hypothesis of induction, f defines uniquely a derivation sequence of $(\mathcal{G}_\ell, W_\ell)$ for ψ. It is easy to see property 1. holds since, by hypothesis of induction, each $\widetilde{w'}_i$ satisfies $\lceil \widetilde{w'}_i \rfloor = \lceil \widetilde{w}_i \rfloor$. Finally, using the hypothesis of induction and the definition of $(\mathcal{G}_\ell, W_\ell)$ we have:

$$\begin{aligned}W_\ell(f(\psi)) &= W_\ell(X^{[d]} \to w_0 w_1 \dots w_n \langle \text{ldf}(Z_1 \dots Z_n) \rangle \prod_{i=1}^{n} W_\ell(f(\psi'_i)) \\ &= W^{\lceil k \rceil}(X^{[d]} \to w_0 Z_1 w_1 \dots w_{n-1} Z_n w_n) \prod_{i=1}^{n} W^{\lceil k \rceil}(\psi'_i) \\ &= W^{\lceil k \rceil}(\psi) \ .\end{aligned}$$

Finally, the fact that f is a surjective function follows from its construction. First, note that each rule in R_ℓ is intended to simulate an LDF derivation step of $(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil})$, while each variable in V_ℓ represents a derivation sequence of $(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil})$. On the other hand, the reader can check that, in each case, the definition of f intends to simulate an LDF derivation of $(\mathcal{G}^{\lceil k \rceil}, W^{\lceil k \rceil})$ using the convenient definition of rules and variables of $(\mathcal{G}_\ell, W_\ell)$. It follows that every derivation sequence of $(\mathcal{G}_\ell, W_\ell)$ is covered by the image of f.

Relying on the definition of f and its properties, the following equalities hold. For each $v \in \Sigma^{\oplus}$:

$$\begin{aligned}
\wr \mathcal{G}^{\lceil k \rceil} \int_{W^{\lceil k \rceil}} (v) &= \sum_{v = \wr w \int} [\![\mathcal{G}^{\lceil k \rceil}]\!]_{W^{\lceil k \rceil}}(w) \\
&= \sum_{v = \wr w \int} \sum_{\psi \in \mathrm{der}(w)} W^{\lceil k \rceil}(\psi) \\
&= \sum_{v = \wr w \int} \sum_{\psi \in \mathrm{der}(w)} W_\ell(f(\psi)) \\
&=^{\dagger} \sum_{v = \wr w' \int} \sum_{\psi' \in \mathrm{der}(w')} W_\ell(\psi') \\
&=^{\dagger\dagger} \sum_{v = \wr w' \int} [\![\mathcal{G}_\ell]\!]_{W_\ell}(w') \\
&= \wr \mathcal{G}_\ell \int_{W_\ell} (v),
\end{aligned}$$

where equality † holds since f satisfies property 2, and equality †† holds since f is a bijection and satisfies property 1.

5.6.2 Unary Polynomially Ambiguous Grammars

Bhattiprolu et al. [10] consider the class of *polynomially ambiguous* WCFGs over the unary alphabet. They show that every WCFG in this class satisfies the Parikh property. In this section we show that in the unary case the class of nonexpansive CFGs strictly contains the class of polynomially ambiguous grammars.

First, let us recall the definition of *polynomially ambiguous* CFG. To do so, the notion of *ambiguity function* of a CFG is needed. Bhattiprolu et al. [10] define the *ambiguity function* of a CFG $\mathcal{G}$ as a function $f : \mathbb{N} \mapsto \mathbb{N}$ that maps each $n \geq 0$ to the number of parse trees whose yield is a word of

length n. Thus, a polynomially ambiguous grammar is defined as follows.

Definition 5.6.7 (Polynomially Ambiguous CFG)**.** A CFG $\mathcal{G}$ is *polynomially ambiguous* iff its ambiguity function $f(n)$ is bounded by a polynomial $p(n)$, with $n \geq 0$.

Second, let us give some notation as given in [10]. Given a CFG $\mathcal{G}$, denote by $\mathcal{T}_\mathcal{G}(X)$ (or simply $\mathcal{T}(X)$) the set of all parse trees τ rooted at a rule with head X, i.e., τ is of the form $\tau = (X \to \alpha)(\ldots)$ where $\alpha \in (V \cup \Sigma^*)$ and $X \in V$. A parse tree $\tau \in \mathcal{T}(X)$ is an *X-pumping tree* iff $\mathcal{Y}(\tau)\downarrow_V = X$. The set of all X-pumping trees of $\mathcal{G}$ is $\mathcal{T}^P(X) \stackrel{\text{def}}{=} \{\tau \in \mathcal{T}(X) \mid \mathcal{Y}(\tau)\downarrow_V = X\}$. The set of all pumping trees of $\mathcal{G}$ is given by $\mathcal{T}^P \stackrel{\text{def}}{=} \{\tau \in \mathcal{T}(X) \mid X \in V\}$. Finally, define the concatenation of two parse trees $\tau_1, \tau_2 \in \mathcal{T}$ with $\mathcal{Y}(\tau_1)\downarrow_V \neq \varepsilon$, denoted by $\tau_1 \circ \tau_2$, by identifying the root of τ_2 with the first variable of $\mathcal{Y}(\tau_1)$.

Theorem 5.6.8. *Every unary polynomially ambiguous CFG is nonexpansive.*

Proof. The proof goes by contradiction. Let $\mathcal{G} = (V, \{a\}, S, R)$ be a polynomially ambiguous grammar and assume that $\mathcal{G}$ is expansive. Then, there is a derivation sequence of the form $X \Rightarrow^* \alpha_0 X \alpha_1 X \alpha_2$ with $X \in V$ and $\alpha_i \in (V \cup \Sigma)^*$. Assuming that every derivation sequence in $\mathcal{G}$ can produce a word of terminals (i.e., $\mathcal{G}$ does not contain useless rules), there exist necessarily at least two distinct parse trees $\tau_1 = (X \to \alpha_1)(\ldots)$ and $\tau_2 = (X \to \alpha_2)(\ldots)$ with $\mathcal{Y}(\tau_1), \mathcal{Y}(\tau_2) \in \Sigma^*$ (not necessarily $\mathcal{Y}(\tau_1) \neq \mathcal{Y}(\tau_2)$).

Let τ be the parse tree that corresponds to the derivation sequence $X \Rightarrow^* \alpha_0 X \alpha_1 X \alpha_2$ and consider the pumping trees $\tau \circ \tau_1$ and $\tau \circ \tau_2$ with:

$$\mathcal{Y}(\tau \circ \tau_1) = \alpha_0\, \mathcal{Y}(\tau_1)\, \alpha_1\, X\, \alpha_2 \quad \text{and} \quad \mathcal{Y}(\tau \circ \tau_2) = \alpha_0\, \mathcal{Y}(\tau_2)\, \alpha_1\, X\, \alpha_2 \ .$$

Now define the X-pumping trees τ' and τ'' as follows:

$$\tau' = (\tau \circ \tau_1) \circ (\tau \circ \tau_2) \quad \text{and} \quad \tau'' = (\tau \circ \tau_2) \circ (\tau \circ \tau_1) \ .$$

Then,

$$\begin{aligned} \mathcal{Y}(\tau') &= \alpha_0\, \mathcal{Y}(\tau_1)\, \alpha_1\, \alpha_0\ \mathcal{Y}(\tau_2)\, \alpha_1\, X\, \alpha_2\, \alpha_2 \quad \text{and} \\ \mathcal{Y}(\tau'') &= \alpha_0\, \mathcal{Y}(\tau_2)\, \alpha_1\, \alpha_0\ \mathcal{Y}(\tau_1)\, \alpha_1\, X\, \alpha_2\, \alpha_2 \ . \end{aligned}$$

Because the alphabet is unary,

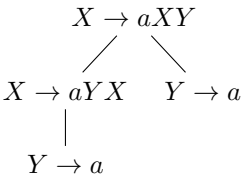
$$\mathcal{Y}(\tau') = \mathcal{Y}(\tau'') \ .$$

However, $\tau' \neq \tau''$.

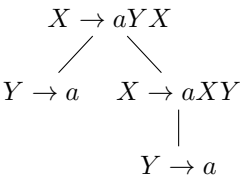
We conclude the proof by using a result by Wich [91] that gives a characterization of polynomially ambiguous grammars. He proves that a CFG $\mathcal{G}$ is polynomially ambiguous iff no two distinct trees τ and τ' of $\mathcal{T}^P$ are such that $\mathcal{Y}(\tau) = \mathcal{Y}(\tau')$. As there exist two distinct trees in $\mathcal{T}^P$ with the same yield, it follows that $\mathcal{G}$ is not polynomially ambiguous (contradiction).

We show that the converse is not true with the following counterexample.

Example 5.6.9. In this example we use the same notation of parse trees that we used in the proof above. Let $\mathcal{G} = (\{X, Y\}, \{a\}, X, \{X \to aXY,\ X \to aYX,\ X \to a,\ Y \to a\})$. Note that X produces derivation sequences with at most one occurence of itself, and Y only produces one terminal symbol. Thus, $\mathcal{G}$ is nonexpansive. However, there are two distinct X-pumping trees τ_1 and τ_2 with $\mathcal{Y}(\tau_1) = \mathcal{Y}(\tau_2)$ (Figures 5.1a and 5.1b). Then, $\mathcal{G}$ is not polynomially ambiguous. ◀



(a) $\tau_1 = (X \to aXY)((X \to aYX)((Y \to a)), (Y \to a))$.



(b) $\tau_1 = (X \to aYX)((Y \to a), (X \to aXY)((Y \to a)))$.

Figure 5.1: Two distinct X-pumping trees with the same yield.

6
Conclusions and Future Work

In this dissertation we offer new language-theoretic perspectives on classical automata constructions using equivalences over words as abstractions of formal languages. Now we conclude this work by giving some final remarks and possible future research directions.

First, we explore well-known automata minimization methods from the point of view of congruences over words of finite index. While these equivalence relations allow us to define different classes of finite-state automata with important roles in automata minimization, they also provide a way to abstract languages and shed light on the relation between independent automata minimization techniques.

- **A view on the double-reversal method using congruences and their left-right duality.** The diagram of Figure 3.1 commutes due to the left-right duality between the automata-based congruences through the reverse operation. In consequence, we prove that the concatenation of two well-known operations, reverse, determinization and reverse, yields to an automaton that is isomorphic to the co-DFA Det^{ℓ}, the automata construction based on the left automata-based congruence.

  The existence of an underlying duality through the reverse operation in the double-reversal method has been rediscovered several times before [25, 12]. In this work, we establish this duality in terms of congruences on words, a simple language-theoretic notion that is fundamental to minimization algorithms; and we exploit it to provide a framework of automata constructions that offer an alternative and clear view on this method.

  One of the main advantages of this perspective is that it allows

us to derive more efficient versions of this method in a systematic way by instantiating congruences that define coarser partitions than the automata-based congruence. We give some remarks on this generalization in the next point.

- **An improved double-reversal method.** We observe that, given an NFA $\mathcal{N}$, *any* co-DFA without empty states that is language-equivalent to $\mathcal{N}$ can be determinized to obtain the minimal DFA. In other words, we can generalize the double reversal method by replacing Det^ℓ by $\mathsf{H}^\ell(\sim^\ell, L)$ for any left congruence $\sim^\ell$, where L is a given regular language. This reveals that a potential line of improvement of this minimization method may result from considering coarser, but effectively computable, left congruences that yield to co-DFAs with at most as many states as the one produced by Det^ℓ.

- **Congruences interpreted as language abstractions allow to relate different minimization methods.**

  It is when interpreting congruences as abstractions of languages, in particular, the left languages of the given NFA, when we observe that if all left languages of the automaton are precisely represented by Nerode's right congruence then determinizing the automaton yields the minimal DFA for its language. This is essentially a reformulation of the generalization of the double-reversal method by Brzozowski and Tamm [18] using the left-right duality between our congruences.

  As a consequence, we are able to connect the double-reversal method with the state-partition-based techniques, specifically Moore's algorithm, in a way that was not possible by exploiting the original formulation. In that respect, it is interesting that Moore's algorithm does not only build Nerode's equivalence at the end of the execution, but the correspondence with the fixpoint computation of Nerode's classes occurs at every step of the computation.

To sum up, congruences over words have been proved crucial to characterize the notion of regular language as well as to define its minimal deterministic form. Now, we revisit them and show that, when interpreting them as language-abstractions, they shed light on the common denominator between independent minimization methods whose connection was notoriously difficult to understand.

An interesting line of work is to generalize these notions using general *quasiorders* on words, i.e., reflexive and transitive relations that may not

satisfy the symmetric property, as a way to obtain nondeterministic automata constructions. In fact, we have recently showed [39] that the deterministic automata constructions given in Chapter 3 can be adapted using finite quasiorders on words to obtain residual automata, i.e., automata for which the right language of each state defines a left quotient of the language. Note that residual automata strictly includes the deterministic ones. In that work we also address a double-reversal-like method to obtain the minimal residual automaton for a given regular language, a technique previously proposed by Denis et al. [28]. Furthermore, we generalize this method along the lines of the generalization of the double-reversal method for building the minimal DFA given by Brzozowski and Tamm [18].

Another line of future research is extending this study to *tree automata* minimization methods. Tree automata are state machines that deal with tree structures rather than words, and thus they recognize regular languages of trees. There exists an analogue version of Nerode's theorem for tree automata [63], and therefore a canonical deterministic (bottom-up[1]) form for every regular tree language. In consequence, similar minimization methods to the ones we have for finite-state automata, based on partition refinement and aggregation, have been proposed. Interestingly, Björklund and Cleophas [11] propose a double-reversal method for tree automata. They observe that the reverse operation will be embedded within the notion of *top-down* and *bottom-up determinization*. However, they do not provide details of the algorithm nor a correctness proof. Thus, exploring determinization and minimization operations on tree automata using equivalences on trees to give a proof of correctness of Brzozowski's method on tree automata is an appealing direction of future work.

As opposed to the congruences we define in Chapter 3, Parikh equivalence is an infinite-index congruence. However, it has the property of making regular languages and context-free languages comparable. In Chapter 4, we address the question about the conciseness of different ways to represent the Parikh image of context-free languages: PDAs, CFGs and NFAs.

- **The standard conversion procedure for PDAs into CFGs is optimal in the unary case, and thus for Parikh equivalence.** The fact that our infinite family of pushdown automata accepts

[1]There exist *bottom-up* and *top-down* tree automata. While non-deterministic top-down tree automata have the same expressive power as non-deterministic bottom-up ones (they are converted one into the other by simply reversing their transitions and converting final states into initial, and viceversa), deterministic top-down tree automata are less powerful than their bottom-up counterparts.

one single word is a key aspect to simplify its comparison with the smallest grammar and the smallest finite-state automaton that are language-equivalent. Our semantics of PDAs computations as *actrees* also facilitates this comparison. On the other hand, the fact that the family is defined over a singleton alphabet solves directly the question for Parikh equivalence.

All at once, we deduce that the standard conversion algorithm [52] for PDAs into equivalent CFGs is optimal for unary nondeterministic PDAs, a case that is excluded in the family defined by Goldstine et al. [45], and for Parikh equivalence.

We also conclude that if we move from the class of unary deterministic PDAs, for which polynomial time conversion procedures exist [23, 78], to to either deterministic pushdown automata over alphabets of size greater than 1 or unary nondeterministic pushdown automata, then no other translation algorithm produces equivalent grammars with fewer variables than the standard conversion algorithm [45, 36].

- **The standard conversion algorithm can be optimized for unary deterministic PDAs.** We show that the standard conversion procedure [52] can be naturally adapted to obtain a new polynomial time conversion algorithm for unary deterministic PDAs.

There is a gap between our lower bounds and the upper bounds given by existing constructions. Namely, we show that the smallest CFG that is language-equivalent to each member $\mathcal{P}(n, k)$ needs at least $\Omega(n^2 k)$ variables (Theorem 4.3.4), while the upper bound established by the standard conversion algorithm is $\mathcal{O}(n^2 k + n^3)$. It follows that the two bounds are not asymptotically tight (see Remark 4.3.5). This gap comes from the fact that the number of stack symbols of each PDA of the family depends on n which also corresponds to the number of states. One possibility to avoid this divergence would be to find a family where the number of stack symbols is independent from the number of states (and viceversa).

Finally, Parikh's Theorem does not extend to weighted languages. In Chapter 5, we focus on characterizing the class of weighted context-free grammars that satisfy this result.

- **Inherent nonexpansiveness as a sufficient and necessary condition for the Parikh property.** In our work, we show that nonexpansiveness is not necessary for the Parikh property, even in the unary case.

However, we observe that the Parikh property holds for a WCFG $(\mathcal{G}, W)$ if and only if there exists a Parikh-equivalent nonexpansive WCFG, i.e., iff $(\mathcal{G}, W)$ is *inherently* nonexpansive. It is known that inherent expansiveness is undecidable in the noncommutative and unweighted case [47], but the question remains open in the commutative and weighted case.

Regarding the applications, we conclude that for programs with procedures and costs over a commutative ring, such as the real or the probability semiring, with a nonexpansive structure, i.e., with no double recursion, can be transformed into one without procedures, such that the weight of Parikh-equivalent traces is preserved. Even though applications are restricted to programs with such an structure, we believe the class is large enough to allow for a variety of applications in the quantitative analysis of asynchronous and multithreaded programs. Another alternative is to under-approximate the verification problem by translating programs with an expansive structure into another program with a nonexpansive structure. To do so, one option is to use the at-most-k-dimension WCFG (see Definition 5.6.1) for some adequate $k \geq 0$. Loosely speaking, this enables the translation of the original program into one without procedures such that the weight of every Parikh-equivalent trace of bounded dimension[2] is preserved.

Finally, recall that, if the semiring of weights is idempotent, like the tropical semiring, the Parikh property always holds, and thus every program with procedures and costs can be translated into one without procedures that preserves the costs. [10, 66, 68].

- **Groebner bases simplify the decision procedure for the Parikh property of weighted CFGs over the rationals.** We focus on weighted CFGs with weights over the rationals, a field of high interest due to the applications of stochastic grammars. We adapt a result by Kuich and Salomaa [67] to give a decision procedure for the Parikh property over this field, that relies on the use of Groebner bases and produces a Parikh-equivalent weighted grammar, if exists. The fact that Groebner bases can be computed by means of standard computer algebra systems makes our procedure practical.

[2]Namely, the trace corresponds to a parse tree of the original program (or CFG) of dimension less or equal than k.

It is worth to remark that this method can also be used to decide if a WCFG over the integers satisfies the Parikh property [67]. More pointedly, since Theorem 5.3.4 states that a WCFG over any ring satisfies the Parikh property if and only if there exist two polynomials that allow to write the grammar in the desired linear form, if the procedure does not find such polynomials with integer coefficients, then they do not exist and the WCFG does not satisfy the property. The naturals is not a ring, and thus the latter result does not hold for WCFGs over this weight domain. In fact, the decidability of the Parikh property over the natural numbers is an interesting open question.

To wrap up, in this dissertation we propose the use of equivalences on words as language abstractions in order to approach different questions from a language-theoretic point of view. More to the point, our language abstractions are *regular approximations* of languages. While our finite-index congruences provide regular approximations (which sometimes are precise representations) of regular languages, Parikh equivalence offers regular approximations of context-free languages.

Other powerful regular approximations of context-free languages have been proposed in the literature. One example is the *downward closure* of a language, i.e., the set of all (not necessarily contiguous) subwords of its members. It is well-known that the downward closure of any language over a given alphabet is regular [50], and concretely, for the class of context-free languages there exist finite-state automata constructions for this over approximation [86, 92].

An appealing direction of future work is to use our congruence-based automata constructions to define finite-state automata for the downward closure of context-free languages, or for regular approximations that lie between the context-free language and its downward closure. What is specially challenging about this question is providing a finite-index "pushdown-automata-based" congruence. While our automata-based congruences are of finite index relying on the fact that NFAs define only finitely many possible configurations, this is not the case for PDAs.

Finally, another interesting question is to study efficient ways to translate PDAs into CFGs that preserve the downward closure, in a similar way as we did for Parikh equivalence.

Funding Acknowledgments

This research was partially supported by:

- BES-2016-077136 grant from the Spanish Ministry of Economy, Industry and Competitiveness (MINECO),
- project No. TIN2015-71819-P, RISCO - RIgorous analysis of Sophisticated COncurrent and distributed systems from the Spanish Ministry of Economy, Industry and Competitiveness (MINECO), and
- project No. PGC2018-102210-B-I00, BOSCO, from the Spanish Ministry of Science, Innovation and Universities (MCIU).

Bibliography

[1] J. Adámek, F. Bonchi, M. Hülsbusch, B. König, S. Milius, and A. Silva. A coalgebraic perspective on minimization and determinization. In *FoSSaCS*, volume 7213 of *Lecture Notes in Computer Science*, pages 58–73. Springer, 2012.

[2] C. Allauzen and M. Mohri. Efficient algorithms for testing the twins property. *Journal of Automata, Languages and Combinatorics*, 8(2):117–144, 2003.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[4] S. Bader, S. Hölldobler, and A. Scalzitti. Semiring artificial neural networks and weighted automata. and an application to digital image encoding. In *KI*, volume 3238 of *Lecture Notes in Computer Science*, pages 281–294. Springer, 2004.

[5] C. Baier, N. Bertrand, and M. Größer. Probabilistic automata over infinite words: Expressiveness, efficiency, and decidability. In *DCFS*, volume 3 of *EPTCS*, pages 3–16, 2009.

[6] C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008.

[7] B. Balle and M. Mohri. Learning weighted automata. In *CAI*, volume 9270 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2015.

[8] G. Baron and W. Kuich. The characterization of nonexpansive grammars by rational power series. *Information and Control*, 48(2):109–118, 1981.

[9] J. Berstel, L. Boasson, O. Carton, and I. Fagnot. Minimization of automata, 2010.

[10] V. Bhattiprolu, S. Gordon, and M. Viswanathan. Extending Parikh's theorem to weighted and probabilistic context-free grammars. In *QEST*, volume 10503 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2017.

[11] J. Björklund and L. Cleophas. A taxonomy of minimisation algorithms for deterministic tree automata. *J. UCS*, 22(2):180–196, 2016.

[12] F. Bonchi, M. M. Bonsangue, H. H. Hansen, P. Panangaden, J. J. M. M. Rutten, and A. Silva. Algebra-coalgebra duality in Brzozowski's minimization algorithm. *ACM Trans. Comput. Log.*, 15(1):3:1–3:29, 2014.

[13] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[14] A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. In *POPL*, pages 62–73. ACM, 2003.

[15] T. Brázdil, J. Esparza, S. Kiefer, and A. Kucera. Analyzing probabilistic pushdown automata. *Formal Methods in System Design*, 43(2):124–163, 2013.

[16] J. A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. *Mathematical Theory of Automata*, 12(6):529–561, 1962.

[17] J. A. Brzozowski and H. Tamm. Theory of átomata. In *Developments in Language Theory*, volume 6795 of *Lecture Notes in Computer Science*, pages 105–116. Springer, 2011.

[18] J. A. Brzozowski and H. Tamm. Theory of átomata. *Theor. Comput. Sci.*, 539:13–27, 2014.

[19] J. R. Büchi. *Finite Automata, their Algebras and Grammars - Towards a Theory of Formal Expressions*. Springer, 1989.

[20] J. Champarnaud, A. Khorsi, and T. Paranthoën. Split and join for minimizing: Brzozowski's algorithm. In *Stringology*, pages

96–104. Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, 2002.

[21] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.

[22] K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. In *CSL*, volume 5213 of *Lecture Notes in Computer Science*, pages 385–400. Springer, 2008.

[23] D. Chistikov and R. Majumdar. Unary pushdown automata and straight-line programs. In *ICALP (2)*, volume 8573 of *Lecture Notes in Computer Science*, pages 146–157. Springer, 2014.

[24] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

[25] B. Courcelle, D. Niwinski, and A. Podelski. A geometrical view of the determinization and minimization of finite-state automata. *Math. Syst. Theory*, 24(2):117–146, 1991.

[26] D. A. Cox, J. Little, and D. O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (2. ed.)*. Undergraduate texts in mathematics. Springer, 1997.

[27] A. de Luca and S. Varricchio. *Finiteness and Regularity in Semigroups and Formal Languages*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[28] F. Denis, A. Lemay, and A. Terlutte. Residual finite state automata. *Fundam. Inform.*, 51(4):339–368, 2002.

[29] M. Droste and P. Gastin. Weighted automata and weighted logics. In *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 2005.

[30] M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[31] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inform.*, 31(1):13–25, 1997.

[32] J. Esparza, P. Ganty, S. Kiefer, and M. Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Inf. Process. Lett.*, 111(12):614–619, 2011.

[33] J. Esparza, M. Luttenberger, and M. Schlund. A brief history of Strahler numbers. In *LATA*, volume 8370 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2014.

[34] J. Esparza, P. Rossmanith, and S. Schwoon. A uniform framework for problems on context-free grammars. *Bulletin of the EATCS*, 72:169–177, 2000.

[35] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *INFINITY*, volume 9 of *Electronic Notes in theoretic Computer Science*, pages 27–37. Elsevier, 1997.

[36] P. Ganty and E. Gutiérrez. Parikh image of pushdown automata. In *FCT*, volume 10472 of *Lecture Notes in Computer Science*, pages 271–283. Springer, 2017.

[37] P. Ganty and E. Gutiérrez. The Parikh property for weighted context-free grammars. In *FSTTCS*, volume 122 of *LIPIcs*, pages 32:1–32:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[38] P. Ganty, E. Gutiérrez, and P. Valero. A congruence-based perspective on automata minimization algorithms. In *MFCS*, volume 138 of *LIPIcs*, pages 77:1–77:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[39] P. Ganty, E. Gutiérrez, and P. Valero. A quasiorder-based perspective on residual automata. In *MFCS*, volume 170 of *LIPIcs*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[40] P. Ganty and R. Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, 2012.

[41] P. Ganty, R. Majumdar, and B. Monmege. Bounded underapproximations. *Formal Methods Syst. Des.*, 40(2):206–231, 2012.

[42] P. Ganty and D. Valput. Bounded-oscillation pushdown automata. In *GandALF*, volume 226 of *EPTCS*, pages 178–197, 2016.

[43] P. García, D. López, and M. Vázquez de Parga. DFA minimization: Double reversal versus split minimization algorithms. *Theor. Comput. Sci.*, 583:78–85, 2015.

[44] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[45] J. Goldstine, J. K. Price, and D. Wotschke. A pushdown automaton or a context-free grammar - which is more economical? *Theor. Comput. Sci.*, 18:33–40, 1982.

[46] S. Göller, R. Mayr, and A. W. To. On the computational complexity of verifying one-counter processes. In *LICS*, pages 235–244. IEEE Computer Society, 2009.

[47] J. Gruska. A few remarks on the index of context-free grammars and languages. *Information and Control*, 19(3):216–223, 1971.

[48] E. Gutiérrez, T. Okudono, M. Waga, and I. Hasuo. Genetic algorithm for the weight maximization problem on weighted automata. In *GECCO*, pages 699–707. ACM, 2020.

[49] U. Hafner, J. Albert, S. Frank, and M. Unger. Weighted finite automata for video compression. *IEEE Journal on Selected Areas in Communications*, 16(1):108–119, 1998.

[50] L. H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94 – 98, 1969.

[51] J. E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.

[52] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 3rd Edition.* Pearson international edition. Addison-Wesley, 2007.

[53] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[54] D. T. Huynh. The complexity of semilinear sets. In *ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 324–337. Springer, 1980.

[55] D. T. Huynh. Deciding the inequivalence of context-free grammars with 1-letter terminal alphabet is Σ_p^2-complete. *Theor. Comput. Sci.*, 33:305–326, 1984.

[56] D. T. Huynh. The complexity of equivalence problems for commutative grammars. *Information and Control*, 66(1/2):103–121, 1985.

[57] S. Iván. Complexity of atoms, combinatorially. *Inf. Process. Lett.*, 116(5):356–360, 2016.

[58] J. Kari. Image processing using finite automata. In *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, pages 171–208. Springer, 2006.

[59] B. Khoussainov and A. Nerode. *Automata Theory and Its Applications*. Birkhauser Boston, Inc., Secaucus, NJ, USA, 2001.

[60] J. C. Kieffer and E. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Inf. Theory*, 46(3):737–754, 2000.

[61] B. Knudsen and J. Hein. Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Research*, 31(13):3423–3428, 2003.

[62] O. Kolak, W. J. Byrne, and P. Resnik. A generative probabilistic OCR model for NLP applications. In *HLT-NAACL*. The Association for Computational Linguistics, 2003.

[63] D. Kozen. On the Myhill-nerode theorem theorem for trees. *Bulletin of the EATCS*, 47:170–173, 1992.

[64] D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 04(03):405–425, 1994.

[65] D. Krob. Some consequences of a Fatou property of the tropical semiring. *Journal of Pure and Applied Algebra*, 93(3):231 – 249, 1994.

[66] W. Kuich. The Kleene and the Parikh theorem in complete semirings. In *ICALP*, volume 267 of *Lecture Notes in Computer Science*, pages 212–225. Springer, 1987.

[67] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*, volume 5 of *EATCS Monographs on theoretic Computer Science*. Springer, 1986.

[68] M. Luttenberger and M. Schlund. Convergence of Newton's method over commutative semirings. *Inf. Comput.*, 246:43–61, 2016.

[69] R. Löhr. *La máquina de ajedrez*. Círculo de Lectores, 2007.

[70] C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 2001.

[71] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[72] M. Mohri and M. Riley. Weighted determinization and minimization for large vocabulary speech recognition. In *EUROSPEECH*. ISCA, 1997.

[73] E. F. Moore. Gedanken-experiments on sequential machines. *Automata studies*, 23(1):60–60, 1956.

[74] C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Intell. Res.*, 7:67–82, 1997.

[75] T. Okudono, M. Waga, T. Sekiyama, and I. Hasuo. Weighted automata extraction from recurrent neural networks via regression on state spaces. *AAAI 2020, to appear*, CoRR abs/1904.02931, 2019.

[76] R. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.

[77] I. Petre. Parikh's theorem does not hold for multiplicities. *Journal of Automata, Languages and Combinatorics*, 4(1):17–30, 1999.

[78] G. Pighizzini. Deterministic pushdown automata and unary languages. *Int. J. Found. Comput. Sci.*, 20(4):629–645, 2009.

[79] A. Rajasekaran, J. O. Shallit, and T. Smith. Additive number theory via automata theory. *Theory Comput. Syst.*, 64(3):542–567, 2020.

[80] T. W. Reps, S. Schwoon, S. Jha, and D. Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Sci. Comput. Program.*, 58(1-2):206–263, 2005.

[81] J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[82] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science. Springer, 1978.

[83] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.

[84] K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 300–314. Springer, 2006.

[85] N. Sharkey. The programmable robot from Ancient Greece. *New Scientist*, 2611, 2017.

[86] J. van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discret. Math.*, 21(3):237–252, 1978.

[87] M. Vázquez de Parga, P. García, and D. López. A polynomial double reversal minimization algorithm for deterministic finite automata. *Theor. Comput. Sci.*, 487:17–22, 2013.

[88] K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational Horn clauses. In *CADE*, volume 3632 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2005.

[89] F. M. Waltz. *Image Processing Using Finite-State Machines*, pages 871–902. Springer London, London, 2012.

[90] G. Weiss, Y. Goldberg, and E. Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5244–5253. PMLR, 2018.

[91] K. Wich. Exponential ambiguity of context-free grammars. In *Developments in Language Theory*, pages 125–138. World Scientific, 1999.

[92] G. Zetzsche. An approach to computing downward closures. In *ICALP (2)*, volume 9135 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 2015.

Acronyms

2-1-NF	2-1 normal form 89
CFG	context-free grammar 4, 26, 76, 97, 137
CFL	context-free language 4
co-DFA	co-deterministic finite-state automaton 14, 23, 38, 135
DCFL	deterministic context-free language 25, 102
DFA	deterministic finite-state automaton 7, 22, 37, 136
DPDA	deterministic pushdown automaton 25, 76
ID	instantaneous description 24, 77
LDF	lowest-dimension-first 119, 123
LIFO	last in, first out 3
LM	leading monomial 110
LT	leading term 110
NFA	(nondeterministic) finite-state automaton 3, 20, 37, 76, 136
PDA	(nondeterministic) pushdown automaton 3, 23, 76, 137
RNA	ribonucleic acid 5
UDPDA	unary deterministic pushdown automaton 86
WCFG	weighted context-free grammar 4, 31, 96, 139
WFA	weighted finite-state automaton 5
WPDA	weighted pushdown automaton 5
XML	extensible markup language 5

This book was printed in Madrid
in January 2021.