Commutative images of languages over infinite alphabets

Author : Mohnish Pattathurajan
Supervisor : prof. dr hab. Sławomir Lasota
Co-supervisor : dr. Piotr Hofman

A thesis presented for the degree of
Doctor of Philosophy in Informatics

Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
Poland

Author's declaration
This thesis is ready to be reviewed
February 5, 2023

Mohnish Pattathurajan

Supervisor's declaration
This thesis is ready to be reviewed
February 5, 2023

prof. dr hab. Sławomir Lasota

Co-supervisor's declaration
This thesis is ready to be reviewed
February 5, 2023

dr. Piotr Hofman

Abstract

Commutative images (Parikh images) are extensively studied for languages of finite automata and context-free grammars over finite alphabets. Parikh's theorem is a well-known fundamental result stating that Parikh images of finite automata and context-free languages are equal, and coincide with semi-linear sets of vectors. In this thesis, we explore Parikh images for a widely studied extension of finite automata, *non-deterministic register automata*, which in contrast to a finite automata, input words over infinite alphabets. Our main objective is establishing to what extent Parikh's theorem keeps holding true when one extends the setting from finite alphabets to infinite ones.

As the first step, we demonstrate that semi-linear sets (suitably extended to infinite alphabets) are not sufficient even for capturing Parikh images of one-register automata, and propose a larger class of *rational* sets (also suitably extended to infinite alphabets) in place thereof. This is motivated by the fact that in finite-alphabet case, semi-linear sets and rational sets of vectors coincide. As our main result we obtain a version of Parikh's theorem for non-deterministic one-register automata: Parikh images of their languages are always rational. Building on this result, we prove rationality of languages of *hierarchical* register automata, a syntactic subclass of non-deterministic register automata strictly extending one-register automata. Furthermore, also building on the main result, we show that the language of every one-register context-free grammar has rational Parikh image, and is therefore Parikh-equivalent to the language of some non-deterministic (and even hierarchical) register automaton.

All the above-mentioned results may be seen as a partial confirmation of validity of Parikh's theorem in the setting of infinite alphabets. Nevertheless, it still remains open if Parikh images of languages of *all* non-deterministic register automata are rational. Likewise, we do not know if the same property holds for languages of all context-free grammars. If this property holds, it would yield the full generalisation of Parikh's theorem to infinite alphabets.

Keywords : automaton, Parikh image, commutative image, register automaton, register context-free language, rational set, rational language.

Streszczenie

Obrazy przemienne (obrazy Parikha) języków regularnych, językow bezkontekstowych i innych klas języków, są od zarania informatyki obiektem intensywnych badań. Szeroko znane i często stosowane twierdzenie Parikha to fundamentalny wynik o równości obrazów przemiennych jęyków bezkontekstowych i regularnych. Ponadto obrazy te są zawsze zbiorami semiliniowymi. W niniejszej rozprawie badamy własności obrazów przemienne języków rozpoznawanych przez *automaty rejestrowe*, stanowiące rozszerzenie automatów skończonych do alfabetów nieskończonych. Głównym celem rozprawy jest ustalenie, w jakim stopniu twierdzenie Parikha pozostaje prawdziwe dla alfabetów nieskończonych.

Jako pierwszy krok pokazujemy, że zbiory semiliniowe (w wersji odpowiednio rozszerzonej do alfabetów nieskończonych) nie są wystarczające, mianowicie obrazy przemienne języków rozpoznawanych przez automaty rejestrowe mogą nie być semiliniowe. W związku z tym proponujemy inny formalizm, w przypadku alfabetów skończonych równoważny zbiorom semiliniowym, mianowicie zbiory regularne (ang. *rational*), czyli zbiory definiowalne za pomocą wyrażeń regularnych, znów w wersji odpowiednio rozszerzonej do alfabetów nieskończonych. Centralnym i najtrudniejszym wynikiem rozprawy jest wersja twierdzenia Parikha dla automatów z jednym rejestrem: obrazy przemienne języków rozpoznawanych przez te automaty są zawsze regularne. Opierając się na tym zaskakująco trudnym do uzyskania wyniku pokazujemy dodatkowo, że obrazy przemienne języków rozpoznawanych przez większą klasę automatów rejestrowych, mianowicie przez tzw. hierarchiczne automaty rejestrowe, są również regularne. Ponadto, znów wykorzystując powyższy centralny wynik pokazujemy, że obrazy przemienne języków bezkontekstowych z jednym rejestrem (języki generowane przez gramatyki bezkontekstowe z jednym rejestrem) są także regularne, a zatem przemiennie równoważne automatom rejestowym (a nawet hierarchicznym).

Powyższe wyniki stanowią częściowe potwierdzenie prawdziwości twierdzenia Parikha dla alfabetów nieskończonych. Wciąż jednak nie wiemy, czy twierdzenie to jest prawdziwe dla automatów z dowolną liczbą rejestrów, które nie są hierarchiczne. Nie wiemy także, czy podobną własność mają języki bezkontekstowe z dowolną liczbą rejestrów. W rozprawa stawiamy hipotezę, że tak w rzeczywistości jest.

Słowa kluczowe : automat skonczony, obraz Parikha, obraz przemienny, automat rejestrowy, rejestrowa gramatyka bezkontekstowa, wyrażenia regularne, języki regularne.

Acknowledgements

Of all the infinite universes, I consider myself luckiest in the ones where I am completing this chapter of my life. For this, I owe everything to my doctoral advisor, prof. Sławomir Lasota. He gave me a life-changing opportunity to study and perform reseatch at MIMUW, and live in the beautiful country of Poland. Apart from shedding some of his vast knowledge onto me, his kindness to me in difficult times (given the circumstances) for which I am forever gladly in debt till my last breath.

Another person to whom I owe is my co-supervisor, dr. Piotrek Hofman. I knew him from day 1 in MIMUW. I am very grateful for his commitment to our research and discussions, in which I learned a lot about computer science and parenting lessons with Max as well.

I was very fortunate to attend lectures, seminars, and teach in MIMUW. I was always surrounded by kind people in teaching as well as administration who are always ready to help with their best efforts, especially prof. Paweł Traczyk, Mr. Piotr Jarmoła, Mr. Paweł Adamski and Ms. Izabela Szabłowska-Petrycka. It is also the place where I met some of my friends Arka Ghosh, Janusz Szchmude, and Jędrek Kołodziejski. I am grateful to them and their families, whom I had met on multiple occasions, and during Christmas, for their efforts to make me feel as at home. It would be unfair if I forgot to mention the beautiful city where I spent the last few years. Poland is a wonderful country with a soul, and I had almost always the best experience with Poles. I thank my teachers from UW, Maria Kuc and Paulina Potasińska, who also taught history and culture. It only enriched when I enjoyed twelve-course meals during Christmas with friends and family of Jędrek. For those brief moments, I could relate to Lawrence of Arabia.

It is impossible to pass this period of my life without mentioning Hotel Hera, where I had some of my best times with my friends Arka, Deeksha, Jayanth, Gilbert, Harut, Tengo, Trivun, Alvaro, and many more. I would thank all of them. Apart from the students, I am very thankful for the administration of Hera, who is super kind and helpful.

I am also thankful for some of my friends who are not part of education or Hera, especially Arun, Ali, Joana, Han, Damian, Jowita, Idalia (for introducing me to classical music), Jana, Sasi, Jaisri, Bhisham, John, and Hari (especially for pushing me into completing my first half-marathon).

All the above would not happen if it were not to Dr. Astrid Kiehn, who introduced me to the beautiful field of Theoretical Computer Science. I was lucky enough to meet her and attend her lectures, for which I will be grateful.

All the good things happened to me because of my parents. I am no special, but a side-effect of their sacrifice and hard work. I am thankful to God for orchestrating such a beautiful symphony of life.

Contents

List of figures

Chapter 1

Outline

Objective

In this chapter we provide the brief overview on the results, and outline the structure of the thesis.

Contents

1 Register automata

Non-deterministic finite automata (NFA) are devices that accept/reject input words over some finite alphabet. An extension for infinite alphabets, *register automata* was introduced over 25 years ago by Francez and Kaminski[1].

The model is also known under the name of *finite-memory automata*[2]. These are non-deterministic finite-state devices equipped with a finite number of registers that can store data values from an infinite domain. We call this infinite domain *atoms* or *data* and we denote it by $\mathbb{A}$. A string over data is called a *data word*, and a *data language* is a set of data words. A *(non-deterministic) register automaton* (NRA) accepts/rejects input data words and recognizes a data language. An important property of this model is that data languages which are recognized are always closed under permutations of data, and that state spaces are formally infinite, but essentially finite, namely finite up to permutations of atoms (data).

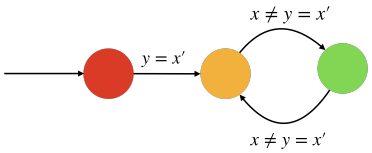


Figure 1.1: A deterministic 1-register automaton.

Example 1.1.1. We illustrate syntax and semantics of this model with the help of deterministic one-register automaton shown in Figure 1.1. Let the input alphabet of the automaton be the set of all atoms, denoted by $\mathbb{A}$. The automaton has three locations and three transition rules. The left-most (red) location is the only initial one, and the right-most (green) location is the only accepting one. The variables x and x' denote the values of the register before and after each transition. The variable y denotes the value of an input letter, an atom. Each transition rule is labelled with a Boolean combination of equalities and inequalities over those variables.

The automaton in Figure 1.1 works as follows. From the initial location, it reads an atom and stores it in the register ($y = x'$). The transitions in the two remaining locations likewise store the input atom in the register, but also ensure that the next data is different from the previous one by the constraint $x \neq y$. This automaton recognizes the data language of words of even length, where every two consecutive letters are distinct:

$$L = \{a_1 a_2 \dots a_n \in \mathbb{A}^* : a_1 \neq a_2 \neq \dots a_{n-1} \neq a_n,\ n \text{ even}\}.$$

Register automata lack most of the properties known from the classical theory of finite automata, like determinization or closure under complement, while they are closed under some standard operations on languages, like union or intersection. Unlike standard finite automata,

[1] Francez and Kaminski, 1994.

[2] Overview of results on automata over infinite alphabets can be found e.g. in Neven, Schwentick, and Vianu, 2004, Segoufin, 2006 and Bojańczyk, 2019.

register automata do not admit equivalent characterizations, like monoid recognisability or rational expressions (regular expressions). Monoid recognisability is weaker than register automata[3], and the proposals of rational expressions for register automata either apply to a restricted subclass of the model, or go beyond the monoid of data words and introduce involved syntax significantly extending the classical concept of regular expressions[4]. It thus seems that there is no satisfactory extension of Kleene theorem to infinite alphabet. The data language recognised by the automaton in the Figure 1.1 is a known *hard* language for developing the setting of rational expressions.

2 Register context-free grammars

Likewise, one extends classical context-free grammars to infinite alphabets. They are first introduced by Cheng and Kaminski[5] (and studied recently in the setting of sets with atoms[6]), who also show the equivalence of register grammars and register pushdown automata, in analogy to the finite alphabet case.

Example 1.2.1. As in the previous example, we consider the input alphabet $\mathbb{A}$. For illustration, consider the following one-register context-free grammar, consisting of non-terminals $Q = \{\bullet, \bullet\}$, the initial non-terminal $\bullet$, and production rules

$$\bullet(x) \xrightarrow{x \neq y = y'} y \; \bullet(y') \qquad \bullet(x) \xrightarrow{x = y = y'} \bullet(y) \; y' \qquad \bullet(x) \to \varepsilon.$$

By $\bullet(x)$ we denote, intuitively speaking, a nonterminal $\bullet$ with the value x in the register. Thus from $\bullet(x)$ the grammar generates a word starting with some data y, where $y \neq x$, and the rest of the word is generated from $\bullet(y)$ (as $y' = y$). From $\bullet(y)$ the grammar generates a word ending with y (as $y' = y$), and the rest of this word is generated from $\bullet(y)$. By repeating this process a number of times, the grammar generates all palindrome-like words of the following form

$$\{a_1 a_2 \dots a_n \, a_n \dots a_2 a_1 \in \mathbb{A}^* : a_1 \neq a_2 \neq \dots \neq a_n\}.$$

3 Commutative images

A finite multiset of atoms is called a *data vector*. For a given data word w, we define its *commutative image*, also known as *Parikh image*, as a data vector $\textsc{Par}(w)$ that counts the number of occurrences of each letters in the word w. The notion of commutative image is extended to data languages in a standard way. We call two data languages *Parikh-equivalent* if they have same Parikh images.

[3] Bojańczyk, 2011.
[4] Bojańczyk, 2020, Libkin, Tan, and Vrgoc, 2015, Kurz, Suzuki, and Tuosto, 2012, Kaminski and Tan, 2006.
[5] Cheng and Kaminski, 1998.
[6] Bojańczyk, Klin, and Lasota, 2014, Clemente and Lasota, 2015a.

In this thesis, we advocate a natural extension of *rational* sets, namely sets definable by an extension of rational expressions, where we allow for sums which are infinite but finite up to permutations of data. As our main research task we attempt to develop a connection between sets definable by the extension of rational expressions and Parikh images of data languages recognised by register automata and register context-free grammars. We consider languages definable by the above mentioned extension of rational expressions (called *rational data languages*), but most importantly sets of data vectors definable by the extension of rational expressions (called *rational sets of data vectors*). As one of our main results, we prove that Parikh images of languages of one-register automata are rational sets of data vectors. As another main result, we prove that languages of one-register context-free grammars are Parikh-equivalent to languages of register automata. Intuitively speaking, this latter result can be seen as a commutative variant of Kleene theorem.

Example 1.3.1. In order to illustrate the notion of rational expressions we work with, and the intuitive idea behind our approach towards proving the above-mentioned result, consider the following data language:

$$L' = \{a_1 a_2 \dots a_n \in \mathbb{A}^* : a_1 \neq a_2 \wedge a_3 \neq a_4 \wedge \dots,\ n \text{ even}\}.$$

Thus L' relaxes the language L from Example 1.1.1 by requiring only every second pair of consecutive letters to be distinct, namely only pairs $(a_i, a_i + 1)$ where i is odd. Unline L, this data language can be easily described by a rational expression as follows:

$$L' = \left(\bigcup_{a,b \in \mathbb{A},\ a \neq b} ab \right)^* .$$

Note that the union is indexed by an infinite set which, up to permutations of atoms, intuitively, consists just of a single pair (a, b) of atoms, where $a \neq b$.

Every $w \in L'$ can be transformed, by swapping letters, to a word in the data language L. Let $w = a_1 a_2 \dots a_n$. If $a_2 = a_3$ we swap non-equal letters a_3 and a_4 thus achieving $a_1 \neq a_2 \neq a_3 \neq a_4$. Next, if $a_4 = a_5$ we swap analogously a_5 and a_6, and so on. Continuing in this way we finally arrive at a word in L. Therefore the two languages are equal up to rearranging, i.e., they are *Parikh-equivalent*. In consequence, as L' is defined by a rational expression, its Parikh image is also rational, and hence Parikh image of L is rational too.

Example 1.3.2. As an illustration of Parikh-equivalence of a context-free language and the language of a register automaton, we note that the language of the grammar from Example 1.2.1 is Parikh-equivalent to the following variation on the language L from Example 1.1.1:

$$\{a_1 a_1\, a_2 a_2\, \dots\, a_n a_n \in \mathbb{A}^* : a_1 \neq a_2 \neq \dots \neq a_{n-1} \neq a_n\},$$

which is also recognisable by a deterministic one-register automaton.

4 Contributions of the thesis

4.1 Parikh images of register automata and context-free grammars

The above examples of register automata and register context-free grammars raise the following natural questions:

(1) Are Parikh images of languages of non-deterministic register automata always semi-linear? This question requires a suitable extension of classical semi-linear sets with unions which are infinite but finite up to permutations of data.

(2) Are Parikh images of languages of non-deterministic register automata always rational sets? This question also requires a suitable extension of classical rational sets with unions which are infinite but finite up to permutations of data.

(3) Does (2) hold for register context-free languages?

(4) Are Parikh images of register automata and register context-free languages the same?

In this thesis, we attempt to answer these questions. Question (1) is answered negatively - as one of the first main result we show that semi-linear sets (naturally extended as mentioned above) are not sufficient, even for one-register deterministic automata[7].

Theorem 5.3.1: (Semi-linear sets are not sufficiently expressive)

Parikh images of languages of deterministic one-register automata are not always semi-linear.

The proof is based on a surprisingly subtle analysis of certain ratio values in Parikh images of words, when the length of words tends towards infinity.

The next main result applies to languages of one-register automata, which are a strict subclass of languages of all register automata.

Theorem 6.1.1: (Parikh images of one-register automata)

Parikh images of languages of non-deterministic one-register automata are rational.

This gives a partial answer to question (2). The crucial part of the proof resorts to a graph-theoretical characterisation of these Parikh images, and uses a sufficient condition for a Hamiltonian cycle in directed graphs.

We use the above theorem to prove its strengthening, which partially answers question (3), namely in case of one-register context-free grammars, whose languages are also a strict subclass of languages of all register context-free grammars.

[7] Juzepczuk, 2013

Theorem 8.1.1: (Parikh images of one-register grammars)

Parikh images of one-register context-free languages are rational.

The result is obtained by a novel type of transformations of derivation trees.

We conjecture that the restriction on the number of registers in Theorems 6.1.1 and 8.1.1 can be dropped; the combinatoric complexity we have encountered already in one-register case makes it however difficult to envisage a generalisation of our approach to the general case.

As we show in Theorem 5.3.1, one-register automata and grammars fail to have semi-linear Parikh images in general. Nevertheless, in the classical finite alphabet setting semi-linear sets coincide with the a priori larger class of rational sets of vectors[8]. Therefore, Theorems 6.1.1 and 8.1.1 can be considered as faithful extensions of Parikh theorem to one-register automata and grammars. Furthermore, as a corollary of the above two results one derives one-register context-free grammars are Parikh-equivalent to register automata, but not necessarily to one-register automata (see also Theorem 8.6.1 below):

Theorem 1.4.1: (1-CFG are Parikh-equivalent to NRA)

Every one-register context-free language is Parikh-equivalent to the language of some non-deterministic register automaton.

This gives a partial answer to question (4).

4.2 New model - hierarchical register automata

In attempt to generalise Theorem 6.1.1, we identify a syntactic subclass of register automata. We call this subclass *hierarchical* register automata (HRA). Unlike classical register automata in which registers are unordered, in HRA registers are ordered. A transition rule changing the value of register i does not affect lower registers, but it may alter higher ones. Another way to see this restriction is that the register values are in a stack in the order of registers. If the value of register i has to be updated, the stack is popped down to register i before register i is updated according to a transition rule. Then arbitrarily chosen (fresh) values are pushed as the new contents of the popped registers.

As an example, consider the two-register HRA (2-HRA) shown in Figure 1.2. It has three locations. The left one is both the initial and the acccepting location. The variables x_1 and x_2 denote the value of the first and second register before a transition. Similarly x'_1 and x'_2 denote the values of registers after a transition. The variable y denotes the input letter. The transition rule from the initial state checks that the input atom is different than contents of registers. Furthermore, the contents of registers are preserved. The next transition rule checks whether the input atom equals the second register. The last transition rule checks whether the

[8] Eilenberg and Schützenberger, 1969.

input atom equals the first register, and updated both registers with fresh atoms (as suggested by its annotation, which does not specify any constraint on new values x_1' and x_2').

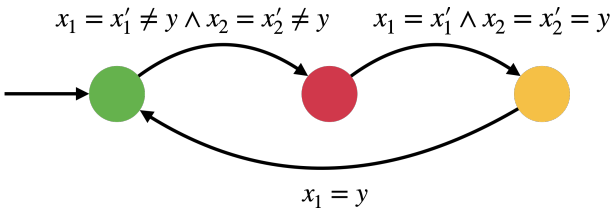


Figure 1.2: 2-HRA accepting blocks of 3 different atoms.

The language of this automaton is similar to the language L' from Example 1.3.1, except that blocks of length 3 are considered instead of blocks of length 2.

$$\{a_1 a_2 \dots a_n \in \mathbb{A}^* : a_1 \neq a_2 \neq a_3 \neq a_1 \wedge a_4 \neq a_5 \neq a_6 \neq a_4 \wedge \dots,\ n \text{ divisible by } 3\}.$$

Figure 1.3 below shows a run of the automaton in Figure 1.2. The orange arrow shows the 'stack' order of the registers. The transitions reading a_3 and a_6 updates both the registers to fresh values different than the input atom, other transitions preserve both the registers (but compare an input atoms against them).

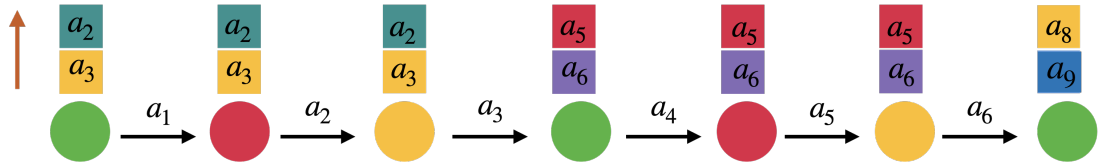


Figure 1.3: An accepting run of the 2-HRA from Figure 1.2.

HRA are strictly more expressive than 1-NRA, both with respect to the class of data languages they recognise, and with respect to Parikh images thereof. Additionally, the languages of HRA are strictly contained in the languages of NRA.

For HRA, we obtain the following result:

Theorem 7.1.1: (Parikh images of hierarchical register automata)

Parikh images of languages of hierarchical register automata are rational.

These result is a step towards the ultimate (but still unreachable) goal: generalise Theorem 6.1.1 the languages of all non-deterministic register automata.

We also show that every rational language of data words is recognised by an HRA:

Theorem 7.5.1

Rational data languages are recognised by HRA.

This, together with Theorem 8.1.1, implies (a strengthening of) Theorem 1.4.1:

Theorem 8.6.1: (1-CFG are Parikh-equivalent to HRA)

Everry one-register context-free language is Parikh-equivalent to the languages of some hierarchical register automaton.

As a corollary, we deduce that an NRA language has rational Parikh image if, and only if it is Parikh-equivalent to some HRA (with, possibly, more registers). In consequence, the ultimate goal can be equivalently achieved by proving that the language of every non-deterministic register automaton is Parikh-equivalent to the language a hierarchical one.

Finally, we believe that the subclass of HRA is interesting on its own, as it seems to be equally well-behaved as one-register automata.

5 Organisation of the thesis

(a) In Chapter 2, we provide the necessary preliminaries about *set with atoms*.

(b) In Chapter 3, we define *data words*, *data languages* and *Parikh images*. Then we define our main objects of interest, *register automata* and *register context-free grammars*. We also provide some examples, recall some fundamental results known in the literature, and overview the most popular variants of register automata.

(c) In Chapter 4, we introduce the concept of *rational set of data words* (rational data languages) and *rational set of data vectors*, and discuss their relationship. We essentially follow the classical definition of rational sets, except that we allow for unions which are infinite but finite up to permutations of data. We also discuss properties of rational sets, in particular *substitution lemma* which is a core tool in later chapters.

(d) In Chapter 5, we extend the classical semi-linear sets to semi-linear sets with unions that are infinite but finite up to permutations of data. First, we show that they are a subclass of rational sets of data vectors. In the second part, we demonstrate that the inclusion is strict, by proving Theorem 5.3.1: semi-linear sets are not expressive enough to capture Parikh images of one-register automata.

(e) In Chapter 6, we prove Theorem 6.1.1. This chapter contains three parts. The first part contains an adaptation of a classical technique of proving Kleene's theorem, to Parikh images of one-register automata. In the second part, we use the closure of rational sets under substitutions recursively, to reduce rationality of Parikh images of one-register automata to rationality of languages over some extended alphabets. In the third part, using a sufficient condition for Hamiltonian cycle in a directed graph, we obtain the required result.

(f) In Chapter 7, we introduce hierarchical register automata and prove Theorem 7.1.1. We start by defining the model, and by exploring its expressiveness in comparison to non-deterministic register automata. Next we show that all rational languages are recognised by hierarchial register automata (Theorem 7.5.1). Finally we show that Parikh images of hierarchical register automata are rational (Theorem 7.1.1), crucially applying Theorem 6.1.1.

(g) In Chapter 8, we prove Theorem 8.1.1. This proof relies on a novel kind of cut-and-paste transformations of derivation trees of one-register context-free grammars. After introducing necessary definitions related to derivation, we show that every such tree can be transformed to a tree of *smaller width* by preserving Parikh images, as long as the width is above a specified constant bound. In the later part we show that languages generated by derivation trees of bounded width have rational Parikh images, again crucially applying Theorem 6.1.1. Using the fact that all rational languages are recognised by hierarchial register automata (Theorem 7.5.1), which implies, together with Theorem 8.1.1, that one-register grammars are Parikh-equivalent to hierarchical register automata (Theorem 8.6.1).

(h) In Chapter 9, we summarise the main results of the thesis and end with some open problem associated with rational sets and Parikh images of register automata.

6 Sources

This thesis is based on the results of the following articles:

- Piotr Hofman, Marta Juzepczuk, Sławomir Lasota, Mohnish Pattathurajan, *Parikh's theorem for infinite alphabets*, Proc. LICS 2021:1-13;
- Sławomir Lasota, Mohnish Pattathurajan, *Parikh images of register automata*, Proc. FSTTCS 2021, 50:1-50:14.

The content of Chapters 4, 5, 6 and 8 is build on the results of the first article, while the content of Chapter 7 is built on the results of the latter one.

Chapter 2

Preliminaries on set with atoms

Objective

In this chapter we introduce the fundamentals of set with atoms and orbit-finite sets. We also recall some basic closure properties of these sets.

Contents

1 Set with atoms

In this thesis, *atoms* are indivisible objects, and we assume an infinite countable set of them. If we can only check whether two atoms are equal or not, we call this structure *equality atoms* - this most basic case is studied most extensively in the literature. There are other possible choices for the structure of atoms, for instance *ordered atoms*, which are rational numbers with their natural order. In forthcoming sections, we formally define atoms and other derived structures that are built from atoms. Using these, we define our main objects of interest, *register automata, register grammars, data vectors* and so on. In this chapter, we rely on the definitions used in the "atom book"[1] which is commonly referred to in the literature[2].

Definition 2.1.1: (Logical structure)

A logical structure is a pair $\mathbb{A} = (\mathcal{A}, \sigma_1, \ldots, \sigma_k)$, where $\mathcal{A}$ is a set and $\sigma_1, \ldots, \sigma_k$ is a collection of functions and relations on $\mathcal{A}$.

Definition 2.1.2: (Atoms)

In the following we always fix a logical structure $\mathbb{A}$, and call its elements *atoms*. We use the symbol $\mathbb{A}$ also for the carrier set of the structure, which should not lead to confusion.

Example 2.1.3. (Equality Atoms) In this thesis, we focus mainly on *equality atoms*:

$$\mathbb{A} = (\mathbb{N}, =).$$

In this structure, we are only allowed to compare two atoms for equality. We denote $a_1, a_2, \cdots$ to refer to individual atoms.

Example 2.1.4. (Ordered atoms) The structure $(\mathbb{Q}, \leq)$ of rational numbers with the usual order we call *ordered atoms*.

We define the *cumulative hierarchy* over $\mathbb{A}$, a hierarchy of sets indexed by ordinal *ranks*. The only set of rank 0 is the empty set. For an ordinal number $\alpha > 0$, a set of rank at most α is any set whose every element is either an atom, or a set of rank strictly smaller than α.

Definition 2.1.5: (Cumulative hierarchy)

The *cumulative hierarchy* $(H_\alpha)_\alpha$ over atoms $\mathbb{A}$ is an increasing collection of sets indexed

[1] Bojańczyk, 2019, Sect. 3.

[2] See for instance Bojańczyk, Klin, and Lasota, 2011; Bojańczyk, Klin, and Lasota, 2014; Clemente and Lasota, 2015a; Bojańczyk, Klin, and Moerman, 2021; Klin, Lasota, and Toruńczyk, 2021.

by ordinals α, defined inductively as follows:

1. $H_0 = \{\emptyset\}$,
2. For an ordinal number $\alpha > 0$,

$$H_\alpha \;=\; \mathcal{P}(\mathbb{A} \cup \bigcup_{\beta<\alpha} H_\beta).$$

The rank of a set X is the smallest ordinal number α such that $X \in H_\alpha$.

Example 2.1.6. The set of all atoms $\mathbb{A}$, as well as the singleton $\{a\}$ for any $a \in \mathbb{A}$, have rank 1, i.e. they belong to H_1. The set $\{\{a_1, a_2\}, a_3\}$ has rank 2.

2 Orbit-finite sets

We are not going to investigate all set in the cumulative hierarchy. We focus only on sets that are finite under *renaming* or *permutations* of atoms. In more formal terms, our desired sets are the ones that are finite up to *atom automorphisms*. The formal definitions follow.

Definition 2.2.1: (Atom automorphism)

An *atom automorphism* is any permutation of the universe of $\mathbb{A}$ that preserves the relations and functions of that structure.

Example 2.2.2. (Atom automorphisms)

1. In case of equality atoms, atom automorphisms are all permutations $\mathbb{N} \to \mathbb{N}$.
2. In case of ordered atoms, atom automorphisms are monotone bijective maps $\mathbb{Q} \to \mathbb{Q}$.

Let x be a set in the cumulative hierarchy and π be an automorphism. The action of π on x, denoted by $\pi(x)$, is a set obtained from x by applying π inductively from atoms, that is

$$\pi(x) = \{\ \pi(y) : y \in x\ \}.$$

Definition 2.2.3: (Support)

Let x be in the cumulative hierarchy, $S \subseteq \mathbb{A}$, and π be an atom automorphism. We say that x is *supported* by S if the following implication

$$(\forall a \in S\ \pi(a) = a) \implies \pi(x) = x \tag{2.1}$$

holds for every atom automorphism π. We also say that S is a *support* of x, or x is *supported* by S. If S is finite, we say that x is *finitely supported* and if S is empty, we say the set x is *equivariant*. In case of equality or ordered atoms, each finitely supported set x has the least support with respect to set inclusion[a]; this least support we denote by $\text{SUPP}(x)$ and call *the support* of X.

[a] Pitts, 2013.

Definition 2.2.4: (S-atom automorphisms)

An atom automorphism that satisfies the premise of (2.1), i.e., which fixes all elements in S, we call *S-atom automorphism*.

Except for some examples in this section that use ordered atoms, in this thesis we work with equality atoms only. Therefore, we may silently assume that each finitely supported set has the least support.

Example 2.2.5. (Supports) Let $\mathbb{A}$ be equality atoms. The set $\mathbb{A}$ is equivariant. Consider the subset of $\mathbb{A}$ that excludes a_1 and a_5,

$$X = \{a \in \mathbb{A} : a \neq a_1, a \neq a_5\}.$$

This set is supported by the excluded set $\{a_1, a_5\}$, which is its (least) support $\{a_1, a_5\} = \text{SUPP}(X)$, but, accoding to the above definition, the set X is also supported by any superset of $\{a_1, a_5\}$.

Remark 2.2.6. (Historical note) Historically, sets with atoms were developed as a model of set theory that violates the axiom of choice. It was originated from the work of Frankel (1922) and later developed by Mostowski (in 1930s). In computer science it was introduced as a formalism for name binding by Pitts and Gabbay (2002)[3] under the name of nominal sets. The concept identical to nominal sets was also independently discovered by Ugo Montanari and Marco Pistore[4], and then investigated from the process calculi perspective. Most recently, sets with atoms have been rediscovered when investigating infinite-state automata[5].

Example 2.2.7. (Non-finitely supported sets) Consider equality atoms. As an example of a set that is not finitely supported, take an arbitrary subset $X \subset \mathbb{A}$ such that both X and $\mathbb{A} \setminus X$ are infinite, for instance the set of atoms that have even index in some enumeration $a_0, a_1, a_2, \ldots$ of atoms. Suppose there is finite support $S \subseteq \mathbb{A}$ of X. Since $X = \{a_0, a_2, \ldots\}$ is infinite and S is finite, for some k all atoms $a_{2k}, a_{2k+1}, \ldots$ are outside of S. Then any S-atom automorphism

[3] Gabbay and Pitts, 2002.
[4] Montanari and Pistore, 1999.
[5] Bojańczyk, Klin, and Lasota, 2011; Bojańczyk, Klin, and Lasota, 2014.

π that swaps a_{2k} and a_{2k+1} does not preserve X, that is $\pi(X) \neq X$, and in consequence S is not a support of X.

Here are sets in the cumulative hierarchy which are of interest to us:

Definition 2.2.8: (Set with atoms)

A *set with atoms* over $\mathbb{A}$ is any set x in the cumulative hierarchy which is hereditarily finitely supported: the set itself is finitely supported, all its elements are finitely supported and so on, till the very atoms are reached.

Unless stated otherwise, all sets mentioned in this thesis are implicitely assumed to be hereditarily finitely supported.

Example 2.2.9. (Set with atoms) Over equality atoms, the set from Example 2.2.5 is hereditarily finitely supported, i.e., is a set with atoms. Over ordered atoms, considered the following set,

$$\{\{c \in \mathbb{A} : a < c < b\} \subseteq \mathbb{A} : a, b \in \mathbb{A} \wedge a < 0 < b\}$$

containing all the open intervals of $\mathbb{Q}$ that contain 0. The set is hereditarily finitely supported: the set itself is supported by the singleton set $\{0\}$, and any its element is supported by its ends, for instance

$$\{c : -1.4 < c < 1\}$$

is supported by $\{-1.4, 1\}$.

Observe that, in particular, all *pure* (atomless) sets are automatically hereditarily finitely supported (and even hereditarily equivariant), i.e., are sets with atoms.

Example 2.2.10. (Encoding tuples) We can encode an ordered pair (a, b) as a set in cumulative hierarchy with help of Kuratowski paring method as

$$\{a, \{a, b\}\}.$$

Using the encoding of pairs, one encodes naturally tuples of any fixed length; in this way words and sets of words (languages) can be also represented as sets in the hierarchy.

Example 2.2.11. Over equality atoms, consider the set $\mathbb{A}^*$ containing all words over $\mathbb{A}$. Each word, encoded using Kuratowski pairs, has a finite rank, while the rank of $\mathbb{A}^*$ is ω. The set $\mathbb{A}^*$ is equivariant and it is a set with atoms, as all its elements (words) are finitely supported (but the support size is not bounded).

Example 2.2.12. Over ordered atoms $(\mathbb{Q}, \leq)$, consider the set of all "increasing" words in $\mathbb{Q}^*$, i.e. words where each next letter is greater or equal to the previous one:

$$L = \{a_1 a_2 \dots a_n \in \mathbb{A}^* : a_1 \leq a_2 \wedge a_2 \leq a_3 \dots a_{n-1} \leq a_n\}.$$

This set is equivariant.

We denote by $\mathbb{A}^{(k)}$ the set of all k-tuples of atoms that are pairwise distinct:

$$\mathbb{A}^{(k)} = \{(a_1, \ldots, a_k) \in \mathbb{A}^k \ : \ a_i \neq a_j \text{ for } i \neq j\}.$$

When working in sets with atoms, we encode all mathematical objects as sets. For instance, a function $f : X \to Y$ is supported by S if its graph $\{\langle x, f(x)\rangle \ : \ x \in X\}$ is so. According to the definition of support, functions supported by S are exactly those commuting with all S-atom automorphisms:

Lemma 2.2.13: (Finitely supported functions)

Suppose X is supported by S. A function $f : X \to Y$ is supported by S if, and only if for every S-atom automorphism π and $x \in X$, we have[a]

$$f(\pi(x)) = \pi(f(x)).$$

[a] Bojańczyk, 2019

Over equality atoms, consider the set $X = \binom{\mathbb{A}}{2}$ of all two-element subsets of atoms. This set is *equivariant*, and each its element is finitely supported by itself. This set is infinite but, up to atom automorphisms, it has only *one equivalence class*: each element is mapped to each other by applying some atom automorphism. Therefore, the set X can be considered as a singleton, up to atom automorphisms. More generally, some sets in the cumulative hierarchy are *finite* up to atom automorphisms, but some are not. For instance, the sets in Examples 2.2.11 and 2.2.12 consists of infinitely many equivalence classes, as atom automorphisms preserve the lenght of a word. In order to develop computational models and algorithms for such models, we focus only on those sets which are finite in the above sense. The formal definition follows.

Definition 2.2.14: (Orbits)

Let X and Y be two sets with atoms and let $S \subseteq \mathbb{A}$ be a finite set of atoms. We say that X and Y are S-equivalent if there is an S-atom automorphism π such that $\pi(X) = Y$. The equivalence classes of this relation are called *S-orbits*. When S is empty we speak of equivariant orbits.

When the set S is clear from the context or is irrelevant, we speak of an *orbit* instead of S-orbit.

Example 2.2.15. (Orbits) Consider equality atoms. The set of atoms $\mathbb{A}$ is one equivariant orbit, and the set from Example 2.2.5 is one $\{a_1, a_5\}$-orbit.

Consider ordered atoms $(\mathbb{Q}, \leq)$. The set $\mathbb{Q}^2$ of ordered pairs of atoms splits into the following three equivariant orbits as shown in the Figure 2.1: the upper region (shown in blue) depicts

the equivariant orbit $\{(x, y) \in \mathbb{Q}^2 : y > x\}$; symmetrically, the bottom region (shown in green) depicts the equivariant orbit $\{(x, y) \in \mathbb{Q}^2 : y < x\}$; and the diagonal line (shown in white) depicts the equivariant orbit $\{(x, y) \in \mathbb{Q}^2 : x = y\}$.

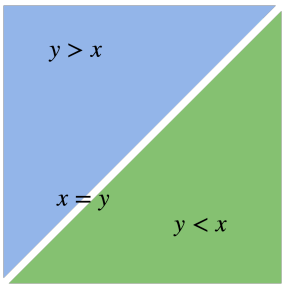


Figure 2.1: Ordered atoms $(\mathbb{Q}, \leq)$. Three equivariant orbits in $\mathbb{Q}^2$.

Definition 2.2.16: (Orbit-finite sets)

A set with atom X is *orbit-finite* if it is a finite union of S-orbits, for some finite $S \subseteq \mathbb{A}$.

It is known that orbit-finiteness does not depend on the choice of S, as long as the structure of atoms is well-behaved (oligomorphic), cf. Theorem 2.2.19 below. Both equality and ordered atoms are oligomorphic.

Example 2.2.17. (Orbit-finite sets) Over equality or ordered atoms, all sets mentioned in Examples 2.2.5 and 2.2.15 are orbit-finite, while the set of all words $\mathbb{A}^*$ from Example 2.2.10 is not orbit-finite.

Example 2.2.18. (Orbit refinement) Consider the set of words of length three $(\mathbb{A}^3)$ over equality atoms. It is an equivariant orbit-finite set that splits into five equivariant orbits, according to the "equality type" of tuples:

$$\begin{aligned}
O_1 &= \{(a, b, c) \in \mathbb{A}^3 : a = b = c\},\\
O_2 &= \{(a, b, c) \in \mathbb{A}^3 : a \neq b \neq c \neq a\} = \mathbb{A}^{(3)},\\
O_3 &= \{(a, b, c) \in \mathbb{A}^3 : a \neq b = c\},\\
O_4 &= \{(a, b, c) \in \mathbb{A}^3 : a = b \neq c\},\\
O_5 &= \{(a, b, c) \in \mathbb{A}^3 : a = c \neq b\}.
\end{aligned}$$

If we add an atom d to the support, the set splits into fifteen $\{d\}$-orbits: the orbit O_1 splits into two $\{d\}$-orbits

$$O_1 = \{(a, b, c) \in (\mathbb{A} \setminus \{d\})^3 : a = b = c\} \cup \{(d, d, d)\},$$

the orbit O_2 into four $\{d\}$-orbits depending on which of a, b, c, if any, equals d, and similarly each of the remaining three orbits splits into three $\{d\}$-orbits. In general, for the words of length

k, $\mathbb{A}^k$ over equality atoms, the number of equivariant orbits, according to the equality type is equal to the number of partitions of the set $\{1, \ldots, k\}$.

In general, given a finite union X of S-orbit, increasing the support S results in increasing of the number of orbits X splits into, but finiteness of the number of orbits is preserved. This justifies existential quantification over S in Definition 2.2.16.

The following fact shows that the orbit-finite property is retained irrespectively of the size of the support.

Theorem 2.2.19: (Orbit refinement)

Over equality or ordered atoms, A finite union of S-orbits is also a finite union of S'-orbits for every S' such that $S \subseteq S'$.[a]

[a] This follows directly from Theorem 3.16 in Bojańczyk, 2019.

3 Closure properties

Our objective is to build orbit-finite mathematical objects in addition to finite objects, and investigate their properties. In this part, we discuss some of the closure properties that will be helpful in developing these objects in later sections.

Due to Theorem 2.2.19 we easily show:

Proposition 2.3.1: (Closure properties)

Orbit-finite sets are closed under union, intersection, Cartesian product and projections[a].

[a] Bojańczyk, 2019

Example 2.3.2. (Closure properties) Over equality atoms $\mathbb{A} = \{a_1, a_2, \ldots\}$, consider the following two finitely supported subsets of $\mathbb{A}$:

$$X_1 = \{a \in \mathbb{A} : a \neq a_1 \wedge a \neq a_2\}$$
$$X_2 = \{a \in \mathbb{A} : a \neq a_2 \wedge a \neq a_3\}.$$

Thus X_1 is one $\{a_1, a_2\}$-orbit, and likewise X_2 is one $\{a_2, a_3\}$-orbit, as shown by green dots in Figures 2.2 and 2.3. The union $X_1 \cup X_2$ contains all atoms that are not a_2 and the intersection

Figure 2.2: The set X_1.

Figure 2.3: The set X_2.

Figure 2.4: $X_1 \cup X_2$.

Figure 2.5: $X_1 \cap X_2$.

$X_1 \cap X_2$ contains all atoms that are not a_1, a_2 and a_3, as shown in Figures 2.4 and 2.5. The latter one is just one $\{a_1, a_2, a_3\}$-orbit, while the former one splits into three $\{a_1, a_2, a_3\}$-orbits.

Example 2.3.3. (Closure properties, cont.) Continuing the previous example, Figure 2.6 illustrates Cartesian product $X_1 \times X_2$ as all black dots at intersection of two green lines. This set splits into four $\{a_1, a_2, a_3\}$-orbits, as depicted in the figure by different colors.

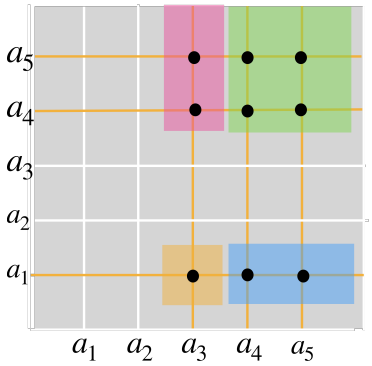


Figure 2.6: Cartesian product $X_1 \times X_2$.

Proposition 2.3.1 deals with finite unions and intersections. It is meaningful to ask what happens if we allow unions to be orbit-finite but not finite. Consider a function f that maps each element of an orbit-finite set X to some set with atoms. In general, the union

$$\bigcup_{x \in X} f(x) \tag{2.2}$$

needs not be finitely supported. As an example, in equality atoms $\mathbb{A} = \{a_1, a_2, \ldots\}$, consider the function $f : \mathbb{A} \to \mathcal{P}(\mathbb{A})$ that maps each a_i to $\{a_{2i}\}$. In this case, the union $\bigcup_{a \in \mathbb{A}} f(a) = \{a_0, a_2, a_4, \ldots\}$ is not finitely supported. However, if we restrict the function f to be finitely supported, the orbit-finite union becomes a meaningful operation, namely the resulting set is always finitely supported; indeed, due to Lemma 2.2.13 applied to the equivariant mapping

$$f \mapsto \bigcup_{x \in X} f(x),$$

we know that if S supports f then it also supports the union (2.2). Furthermore, the resulting set is orbit-finite whenever all sets $f(x)$ are so:

Proposition 2.3.4: (Orbit-finite unions)

Let X be an orbit-finite set and f be a finitely supported function that maps each element

$x \in X$ to an orbit-finite set $f(x)$. Then the union

$$\bigcup_{x \in X} f(x)$$

is orbit-finite[a].

[a] Bojańczyk, 2019, Excercise 62.

Notice that every finite union is a special case of orbit-finite union.

Remark 2.3.5. (Equality atoms) In the following chapters we work exclusively with equality atoms.

Summary

So far, we introduced set with atoms and associated definitions. In the next chapter we will define an automaton model that extends finite automata to orbit-finite alphabets. We also extend likewise classical context-free grammars.

Chapter 3

Register automata and register grammars

Objective

This chapter introduces data languages, their Parikh images, register automata and register context-free grammars. We also recall some known results on these models.

Contents

1 Data words, data languages and their Parikh images

Building on the fundamentals developed in the previous section, we extend the notions of words, languages and vectors to the orbit-finite setting.

Definition 3.1.1: (Data words and data languages)

Fix an orbit-finite alphabet Σ. A *data word* $w \in \Sigma^*$ is a string over Σ. A *data language* $L \subseteq \Sigma^*$ is any set of such strings. From now on we mostly consider input alphabets of the form $\Sigma = H \times \mathbb{A}$, where H is a finite pure (atomless) set.

Definition 3.1.2: (Data vectors)

A *data vector* over Σ is a finite multiset over Σ. Alternatively, a data vector may be seen as a function $v : \Sigma \to \mathbb{N}$ such that $v(\sigma) = 0$ for all except finitely many letters $\sigma \in \Sigma$.

In particular, the zero (empty) multiset $\mathbf{0}$ satisfies $\mathbf{0}(a) = 0$ for every $a \in \Sigma$. A singleton, written $\{a\}$, maps a to 1 and all other letters to 0.

Definition 3.1.3: (Size of data vector and length of data word)

For a data vector v, we define the domain of v as $\mathrm{DOM}(v) = \{a \in \Sigma : v(a) > 0\}$ and the size of v as $|v| = \sum_{a \in \mathrm{DOM}(v)} v(a)$. We also write $|w|$ for the *length* of a data word w, which should not lead to confusion.

Definition 3.1.4: (Multiset ordering and operations)

We order multisets pointwise: $v \sqsubseteq v'$ if $v(a) \leq v'(a)$ for all $a \in \Sigma$. Addition of multisets is pointwise: $(v + v')(a) = v(a) + v'(a)$ for every $a \in \Sigma$. Likewise we define subtraction $v - v'$, for $v' \sqsubseteq v$.

Example 3.1.5. In this example we consider equality atoms $\mathbb{A}$, and the alphabet is just $\mathbb{A}$. For future use we define the data language of words in which the first letter appears again:

$$L_1 = \{a_1 a_2 \dots a_n \in \mathbb{A}^* : a_1 = a_i \text{ for some } i \in \{2, \dots, n\}\};$$

the data language of words, in which some letter appears twice:

$$L_2 = \{a_1 a_2 \dots a_n \in \mathbb{A}^* : a_i = a_j \text{ for some distinct } i, j \in \{1, \dots, n\}\};$$

and its complement – the data language of words, in which no letter appears twice:

$$L_3 = \{a_1 a_2 \dots a_n \in \mathbb{A}^* : a_i \neq a_j \text{ for all distinct } i, j \in \{1, \dots, n\}\} = \bigcup_{n \in \mathbb{N}} \mathbb{A}^{(n)}.$$

Definition 3.1.6: (Parikh image)

The Parikh image (commutative image) of a word $w \in \Sigma^*$ is the multiset $\textsc{Par}(w) : \Sigma \to \mathbb{N}$, where $\textsc{Par}(w)(a)$ is the number of appearances of a letter $a \in \Sigma$ in w. For a language $L \subseteq \Sigma^*$, its Parikh image is $\textsc{Par}(L) = \{\textsc{Par}(w) : w \in L\}$.

Example 3.1.7. (Parikh images) Consider the data language L_1 from Example 3.1.5. The Parikh image $\textsc{Par}(L_1)$ is the set of all data vectors (multisets) in which some atom is mapped to a number equal or greater than 2. $\textsc{Par}(L_1) = \textsc{Par}(L_2)$. Parikh image of the complement L_3 of L_2, $\textsc{Par}(L_3)$, is the complement of $\textsc{Par}(L_2)$, i.e., contains all data vectors not present in $\textsc{Par}(L_2)$. In general, Parikh image does not commute with complement; it does exactly when the language (and in consequence its complement) is closed under permutations of letters in a word.

Definition 3.1.8: (Parikh-equivalence)

Two languages $L, L' \subseteq \Sigma^*$ are Parikh-equivalent if they have the same Parikh images: $\textsc{Par}(L) = \textsc{Par}(L')$.

Example 3.1.9. (Parikh-equivalent languages) The data languages L_1 and L_2 from Example 3.1.5 are Parikh-equivalent.

Parikh images are widely studied for finite automata and context-free grammars. The well-known Parikh's theorem[1] characterizes Parikh images of context-free grammars by semi-linear sets[2] and implies coincidence of Parikh images of grammars and finite automata:

Theorem 3.1.10: (Parikh's theorem)

Over finite alphabets, every context-free language has semi-linear Parikh image, and is

[1] Parikh, 1966.
[2] We define a generalisation of semi-linear sets to orbit-finite alphabets in Chapter 5.

therefore Parikh-equivalent to some regular language.

Example 3.1.11. Palindromes over $\Sigma = \{\sigma, \delta\}$ are Parikh-equivalent to the following regular language:

$$L = (\sigma\sigma + \delta\delta)^*(\epsilon \cup \Sigma).$$

We recall that all sets mentioned in this thesis are implicitely assumed to be hereditarily finitely supported. Orbit-finite sets of data words, or data vectors, are characterised by the following lemma:

Lemma 3.1.12: (Orbit-finite data languages)

A set X of data words, or data vectors, over an orbit-finite alphabet Σ is orbit-finite if, and only if $\{|v| : v \in X\} \subseteq \mathbb{N}$ is bounded.

Proof. Fix an orbit-finite set Σ. The 'only if' implication is immediate, as the length (or size) is invariant inside an orbit. Towards the 'if' implication for data languages, we observe that the set Σ^n of words of length n is orbit-finite, for every $n \in \mathbb{N}$, as Cartesian products preserve orbit-finiteness. Therefore a finitely supported language $X \subseteq \Sigma^*$ satisfying $|v| \leq n$ for $v \in X$, is a subset of a finite union of orbit-finite sets and hence orbit-finite itself. In consequence, $\textsc{Par}(X)$ is is also orbit-finite, as the image of X under an equivariant function, which proves the claim for sets of data vectors. □

2 Register automata

After having introduced fundamentals of set with atoms, data words, data vectors, and data languages, in this section we define an extension of finite automata that accept/reject data words, known as *register automata*. Register automata are also known under the name of *finite-memory automata*, and were introduced over 25 years ago by Francez and Kaminski[3]. These are non-deterministic finite-state devices equipped with a finite number of registers that can store data values (atoms) that are read from input. As input alphabet, one can consider in principle any orbit-finite set Σ. In this thesis, we focus on the model of register automata, as originally defined by Francez and Kaminski, that inputs letters from $\Sigma = H \times \mathbb{A}$, for some pure (atom-less) finite set H. On reading an *input letter*, the automaton compares the letter's atom with atoms stored in the registers and *fires a transition* if the input and register values satisfy the *constraints* described by a *transition rule*. As a result, the register values can be also *modified*, based on the constraints of transition rule. This section is devoted to formal definition of register automata, and to discussion of some of the fundamental results of this model.

[3]Francez and Kaminski, 1994.

Definition 3.2.1: (Non-deterministic register automata)

A *non-deterministic k-register automaton* (k-NRA) $\mathcal{A}$ consists of: a finite set H (finite component of input alphabet), a finite set of control locations[a] Q, subsets $I, F \subseteq Q$ of initial resp. accepting locations, and a finite set Δ of transition rules of the form

$$(q(x_1, x_2 \dots x_k), \langle h, y \rangle, \varphi, q'(x'_1, x'_2 \dots x'_k)) \tag{3.1}$$

where $q, q' \in Q$, $h \in H$, and the transition constraint $\varphi(x_1, x_2 \dots x_k, y, x'_1, x'_2 \dots x'_k)$ is a Boolean combination of equalities involving the variables $x_1, x_2 \dots x_k, y, x'_1, x'_2 \dots x'_k$.

[a] Locations can be also called *control states*, or just states.

Implicitely, the input alphabet of a k-register automaton is $H \times \mathbb{A}$, and its states (called *configurations* below) are of the form $Q \times \mathbb{A}^{(k)}$ and thus consist of a control location and a valuation of registers. A valuation is a tuple of k pairwise distinct atoms. Both the alphabet and configurations are in general infinite, but orbit-finite. The constraint φ in (3.1) specifies all possible relations between the register pre-values (denoted by $x_1, x_2 \dots x_k$), input atom (denoted by y), and register post-values (denoted by $x'_1, x'_2 \dots x'_k$) resulting from a transition. If φ entails the equality $x_i = x'_i$, we say that the ith register is *preserved* by the transition rule; otherwise we say that the ith register is *updated*.

Example 3.2.2. (Register automaton) Consider a register automaton with locations $Q = \{\bullet, \bullet\}$, as shown in Figure 3.1. In this and all the following examples in this section, we assume that H is a singleton (it is thus omitted in the figures). The initial location is $\bullet$ (the left one), and both locations are accepting.

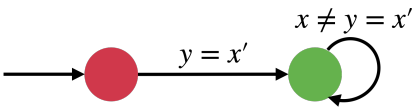


Figure 3.1: A deterministic one-register automaton.

The automaton is deterministic in the sense that transition rules determine uniquely the next configuration (control location and the value of the register), given a previous configuration and an input letter (atom).

Definition 3.2.3: (Configuration)

A *configuration* $\langle q, (a_1 a_2 \dots a_k) \rangle \in Q \times \mathbb{A}^{(k)}$ of $\mathcal{A}$, written briefly

$$q(a_1 a_2 \dots a_k),$$

consists of a control location $q \in Q$ and a tuple of pairwise distinct[a] register values $a_i \in \mathbb{A}$, for $1 \leq i \leq k$. A configuration is *initial* if $q \in I$ and it is accepting if $q \in F$ (irrespectively of register values).

[a]Distinctness of register values is not relevant for expressiveness of register automata.

For each tuple $\mathbf{r} = (a_1 a_2 \dots a_k) \in \mathbb{A}^{(k)}$, atom $b \in \mathbb{A}$, and tuple $\mathbf{r}' = (a'_1 a'_2 \dots a'_k) \in \mathbb{A}^{(k)}$ that satisfy the transition constraint, i.e., $(a_1 a_2 \dots a_k, b, a'_1 a'_2 \dots a'_k) \models \varphi$, a rule (3.1) induces a transition

$$q(a_1 a_2 \dots a_k) \xrightarrow{\langle h,b \rangle} q'(a'_1 a'_2 \dots a'_k)$$

labeled by $\langle h, b \rangle$ from the configuration $q(a_1 a_2 \dots a_k)$ to $q'(a'_1 a'_2 \dots a'_k)$. The semantics of k-NRA is defined as in case of classical finite automata, with configurations considered as states and $\Sigma = H \times \mathbb{A}$ as an alphabet.

Definition 3.2.4: (Run of a register automaton)

A run of $\mathcal{A}$ over a data word $w = \langle h_1, b_1 \rangle \langle h_2, b_2 \rangle \dots \langle h_n, b_n \rangle \in \Sigma^*$ is any sequence of configurations $q_0(\mathbf{r}_0), q_1(\mathbf{r}_1), \dots, q_n(\mathbf{r}_n)$, related by transitions labeled by consecutive letters of w:

$$q_0(\mathbf{r}_0) \xrightarrow{\langle h_1,b_1 \rangle} q_1(\mathbf{r}_1) \xrightarrow{\langle h_2,b_2 \rangle} \dots \xrightarrow{\langle h_n,b_n \rangle} q_n(\mathbf{r}_n), \tag{3.2}$$

where $q_0(\mathbf{r}_0)$ is an initial configuration (i.e., $q_0 \in I$). Note that the initial values of registers $\mathbf{r}_0$ are chosen arbitrarily (we say informally that they are *guessed*)[a]. A run is *accepting* if the ending configuration $q_n(\mathbf{r}_n)$ is accepting (i.e., $q_n \in F$). A data word w is *accepted* by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run over w.

[a]This is not the unique nor the most often used possibility, but this definitional choice seems most convenient technically in our later developments. In the literature, it is most often assumed that all registers are initially *undefined* (see for example Bojańczyk, 2019). These minor variations of the definition do not affect the expressive power of the model.

Example 3.2.5. (Run of a register automaton) Consider a run of the automaton from Example 3.2.2 over a word $a_1 a_2 a_3 \in \mathbb{A}^{(3)}$, as shown in Figure 3.2 below. In the figure, each circle with a letter represents a configuration. For example, the left most one (initial configuration) denotes the configuration $\bullet(a_0)$. As mentioned above, the initial register values are guessed. Then the automaton reads and stores a_1 in its register (thus forgetting the initial value a_0) and goes to $\bullet$. From there, it reads a_2 different from a_1 (according to the transition rule) and goes again to $\bullet$, and then likewise the automaton proceeds with a_3. The run is accepting since $\bullet$ is accepting.

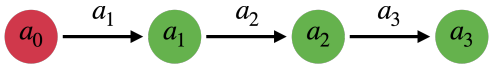


Figure 3.2: A run of a register automaton.

Definition 3.2.6: (Language recognised by a register automaton)

Let $L_{q(\mathbf{r})\,q'(\mathbf{r}')}(\mathcal{A})$ be the set of data words having an accepting run (3.2) that starts in $q_0(\mathbf{r}_0) = q(\mathbf{r})$ and ends in $q_n(\mathbf{r}_n) = q'(\mathbf{r}')$. The language $L(\mathcal{A})$ recognised by $\mathcal{A}$ is defined as:

$$L(\mathcal{A}) \;=\; \bigcup_{q\in I, q'\in F, \mathbf{r},\mathbf{r}'\in\mathbb{A}^{(k)}} L_{q(\mathbf{r})\,q'(\mathbf{r}')}(\mathcal{A}). \tag{3.3}$$

Example 3.2.7. (Language recongnised by a register automaton) The automaton described in Example 3.2.2 recognises the language containing all words in which each two consecutive letters are different:

$$L = \{a_1 a_2 \dots a_n \in \Sigma^* : a_1 \neq a_2 \neq \cdots \neq a_n\}.$$

Example 3.2.8. (2-NRA) We define a two-register automaton with three locations, where the red (left-most) location is the only initial one, and all locations are accepting.

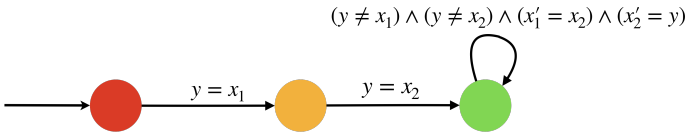


Figure 3.3: A (deterministic) 2-register automaton: each three consecutive letters are pairwise different.

The automaton has three transition rules. The first two rules check that the input atom equals the initially guessed value of the first (respectively, the second) register. Note that these two values are necessarily different. The third rule loops from green to green location, checks that the input atom is different from both registers, shifts the value of the second register to the first one, and updates the second register with the input atom. Therefore, the language L accepted by the automaton consists of all words in which each three consecutive letters are pairwise different (instead of each two consecutive letters, as in Example 3.2.7):

$$L = \{\, a_1 a_2 \dots a_n \in \mathbb{A}^* : \neq (a_1, a_2, a_3) \;\wedge\; \neq (a_2, a_3, a_4) \;\wedge\; \cdots \;\wedge\; \neq (a_{n-2}, a_{n-1}, a_n)\},$$

where the shorthand $\neq (a_1, a_2, a_3)$ is to say that a_1, a_2 and a_3 are pairwise distinct: $a_1 \neq a_2 \neq a_3 \neq a_1$.

Remark 3.2.9. (Guessing) Our definition of register automata allows for *guessing*[4], i.e., when firing a transition rule an automaton may non-deterministically choose, and store in its register, an atom not yet seen in the input, besides atoms stored in registers before firing the transition.

In this thesis we mostly consider register automata with guessing. The literature considers both register automata with guessing as well as register automata without this feature. In our setting, one can enforce a syntactical restriction on transition constraints that corresponds to automata without guessing: each constraint should entail equality of each post (primed) variable x'_i either to some pre (non-primed) variable x_j, or to input variable y, and initial value of each register should be updated before being tested in constraints.

Remark 3.2.10. (Initial values and distinct values in the registers) As we discussed already above, initial register values are guessed. Furthermore, in each configuration the automaton is restricted to store distinct vales in its registers. These decisions are not the unique possible ones, and alternative variants often appear in the literature. For instance, initial values of registers may be undefined, and become defined at first update[5]; then, depending on a variant of definition, a defined register stays defined forever during all subsequent updates, or may become again undefined due to firing a specified transition rule. There are also variants allowing for storing the same atom in multiple registers. All these variants have the same espressive power.

Unlike finite automata, where most of the well-known variations are expressively equivalent, many variations of register automata are not equivalent to each other. For instance, *guessing* is a critical feature, and it gives additional expressiveness to the model. Here is an example:

Example 3.2.11. (1-NRA with guessing) The automaton in Figure 3.4 recognises all the three letter words, in which all the three letters are pairwise distinct. The red location is the only initial one, and the green one is the only accepting one.

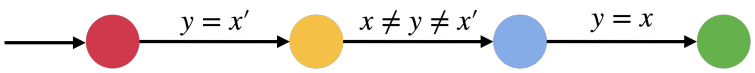


Figure 3.4: Automaton recognising pairwise distinct triples.

The automaton uses one register. The first transition loads the input atom to the register. The second transition inputs an atom different than the atom stored in the register, and updates the register with a (guessed) fresh atom different from both. The last transition inputs the atom which is stored in the register.

Proposition 3.2.12: (Guessing is more powerful)

The language L from Example 3.2.11, while being recognised by a 1-NRA with guessing, as

[4] Segoufin, 2006; Neven, Schwentick, and Vianu, 2004.
[5] Bojańczyk, 2019.

well as by a 2-NRA without guessing, is not recognised by any 1-NRA without guessing.

Remark 3.2.13. (Deterministic register automata) We can distinguish a subclass of *deterministic* register automata, by requiring that

- transition rules determine uniquely the next configuration (location and register values), given a previous configuration and an input letter, and
- initial register values are irrelevant, i.e. no register is tested before its first update.

We do not detail this definition, as in this thesis we concentrate on properties of non-deterministic register automata.

2.1 Closure properties

Regular languages admit several closure, like closure under union, intersection, and complementation. We point in this section to the fact that NRA display some of these closure properties, but fail to satisfy some others.

Proposition 3.2.14: (Unions and intersections)

Languages of NRA are closed under finite unions and intersections[a].

[a] Francez and Kaminski, 1994, Theorem 3.

What about complements of NRA languages? The following example answers that question negatively.

Example 3.2.15. (Complementation) Consider the 1-NRA shown in Figure 3.5. The red locations is initial and the green one is accepting. The transition rule going from red to yellow location non-deterministically chooses an input letter to reappear later, and stores it in the register. If the letter again appears, then the automaton reaches the accepting location. This automaton recognizes the language L_2 from Example 3.1.5.

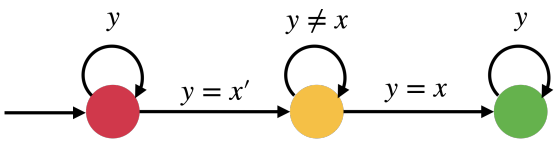


Figure 3.5: Automaton recognising language L_2 from Example 3.1.5.

The complement of the language L_2 is L_3 from the Example 3.1.5.

Proposition 3.2.16: (No closure under complementation)

The language L_3 from Example 3.1.5 is not recognised by any NRA.

Proof. Suppose such a k-NRA $\mathcal{A}$ exists. Consider a word w of length $k+2$ such that all $k+2$ letters are pairwise different, and an accepting run of $\mathcal{A}$ over w. After reading the first $k+1$ letters, at least one of letters read so far, say a_i, is not stored in any of registers. Then the word w with a_i replaced by a_{k+2} is also accepted, yielding a contradiction. □

Remark 3.2.17. (Complementation) As languages of deterministic register automata are closed under complement we deduce that non-deterministic register automata are strictly more expressive than deterministic ones. Surprisingly, it has been recently shown that if both data language L and its complement L^c are recognized by NRA without guessing then both the languages are recognized by deterministic ones[6].

2.2 Decision problems

In this part, we discuss some of the widely studied decision problems for register automata.

2.3 Emptiness

The *emptiness problem* asks, given a non-deterministic register automaton $\mathcal{A}$, if the language $L(\mathcal{A}) = \emptyset$. For finite automata the problem is decidable in polynomial time. The following theorem gives the complexity for register automata:

Theorem 3.2.18: (Emptiness)

Emptiness problem is PSPACE-complete for NRA that can store duplicates in their registers[a].

[a] Francez and Kaminski, 1994, Theorem 1, Demri and Lazic, 2009, Theorem 5.1.

For our model, NRA that do not store duplicates in their registers, the complexity significantly reduces. The following theorem provides the complexity:

Theorem 3.2.19: (Emptiness)

Emptiness problem is NP-complete for NRA that do not store duplicates in their registers

[6] Klin, Lasota, and Toruńczyk, 2021.

(as in our definition)[a].

[a]Sakamoto and Ikeda, 2000, Theorem 4.

2.4 Universality

The *universality problem* asks, given a non-deterministic register automaton $\mathcal{A}$, if the language $L(\mathcal{A}) = \Sigma^*$. The problem is unfortunately undecidable:

Theorem 3.2.20: (Universality)

Universality problem is undecidable for NRA[a].

[a]Neven, Schwentick, and Vianu, 2004.

The proof consists of a reduction from the *Post correspondence problem (PCP)*. As a corollary we get the following:

Corollary 3.2.21: (Equivalence and inclusion)

Equivalence and inclusion problems for languages of NRA are undecidable.

Remark 3.2.22. (Alternating register automata) Undecidability proof works for 2-NRA without guessing, and also for 1-NRA with guessing. On the other hand, the universality problem is decidable for alternating one-register automata without guessing[7]. Since 1-NRA are a special case of alternating register automata where all states are existential, the universality problem is also decidable for 1-NRA without guessing. The proof relies on a *well-quasi order* on the reachable sets of configurations.

Remark 3.2.23. (Unambiguous register automata) A non-deterministic register automaton is *unambiguous* if every data word admits at most one accepting run (in consequence, every accepted word admits exactly one accepting run). Unambiguous register automata are strictly more expressive than deterministic ones. For example, the reverse of the language L_1 from Example 3.1.5:

$$L = \{a_1 a_2 \dots a_n \in \mathbb{A}^* \ : \ a_n = a_i \text{ for some } i \in \{1, 2, \dots, n-1\}\},$$

is not recognised by any deterministic register automata but is recognised by an unambiguous one.

[7]Demri and Lazic, 2009.

Universality for this model was first shown decidable in 2-ExpSpace[8], then the complexity was improved to 2-ExpTime[9], and finally to ExpTime[10].

2.5 Variants of register automata

In this part, we mention some variants of the model of register automata, and discuss the relationship between expressive power of these variants.

2.5.1 Orbit-finite automata

The definition of *orbit-finite automaton* is obtained in a very natural way, namely by allowing in a classical definition of non-deterministic finite automaton (NFA) orbit-finite state spaces in place of finite ones[11]. In this setting the input alphabet of an automaton, as well as its state-space and its transition relation, are all arbitrary orbit-finite sets. This elegant model is a generalization of non-deterministic register automata to arbitrary orbit-finite alphabets. However, this model is not more powerful than register automata on alphabets of the form $H \times \mathbb{A}$, where H is finite[12].

Example 3.2.24. (Orbit-finite automaton) Let $\Sigma = \mathbb{A}^{(2)}$. The following language of "paths" is recognised by an orbit-finite automaton:

$$L = \{\langle a_1, b_1\rangle\langle a_2, b_2\rangle \dots \langle a_n, b_n\rangle \in \Sigma^* : b_1 = a_2 \wedge b_2 = a_3 \wedge \dots \wedge b_{n-1} = a_n\}.$$

2.5.2 Alternating register automata

In *alternating register automata*, control locations are partitioned into existential and universal ones, and acceptance of a word is determined by the means of an acceptance *game* between the existential and universal player. The alternating model is strictly more expressive than non-deterministic one. As mentioned in Remark 3.2.22, alternating one-register automata without guessing have decidable emptiness and henceforth decidable universality[13].

2.5.3 Two-way register automata

All variants discussed so far are *one-way*. One can allow however a register automaton to go back and forth along the input string. In case of finite automata, this extension does not increase expressiveness. The situation is different in case of register automata: two-way model is strictly more expressive than one-way model, both in the case of deterministic and non-deterministic

[8] Mottet and Quaas, 2019.
[9] Barloy and Clemente, 2021.
[10] Bojańczyk, Klin, and Moerman, 2021.
[11] Bojańczyk, 2019.
[12] Bojańczyk, Klin, and Lasota, 2014.
[13] Demri and Lazic, 2009.

register automata[14]. For instance, this model is able to recognize the language

$$L_3 = \{a_1 a_2 \dots a_n \in \mathbb{A}^* : a_i \neq a_j \text{ for all distinct } i, j \in \{1, \dots, n\}\} \tag{3.4}$$

in Example 3.1.5, which is not recognized by any one-way non-deterministic register automata. Furthermore, the emptiness problem becomes undecidable for two-way register automata[15].

2.5.4 Single-use deterministic register automata

The model of *single-use* deterministic register automata is obtained by imposing a restriction: each register value may be testes only once, and it "disappears" as a result of a test. This subclass of deterministic register automata has been recently introduced by Bojańczyk and Stefański[16]. The paper provides a number of good properties of single-use deterministic register automata. In particular, the two-way model is expressively equivalent to one-way model. Single-use deterministic register automata recognize exactly data languages recognisable by *orbit-finite monoids*[17], and also languages definable in a variant of monadic second-order logic called *guarded-MSO*[18]. The single-use model is, on the other hand, strictly weaker than unrestricted deterministic register automata. For instance, the model is not able to recognise the language

$$L_1 = \{a_1 a_2 \dots a_n \in \mathbb{A}^* : a_1 = a_i \text{ for some } i \in \{2, \dots, n\}\}$$

in Example 3.1.5, which is recognised by a deterministic one-register automaton.

2.5.5 Mutual relationships

The relationships (inclusions, non-empty intersections, etc.) between the classes of languages recognized by the models discusses above are shown in Figure 3.6[19]:

[14] Bojańczyk, 2019.
[15] Neven, Schwentick, and Vianu, 2004.
[16] Bojańczyk and Stefański, 2020.
[17] Bojańczyk, 2011.
[18] Colcombet, Ley, and Puppis, 2015.
[19] Bojańczyk, 2019.

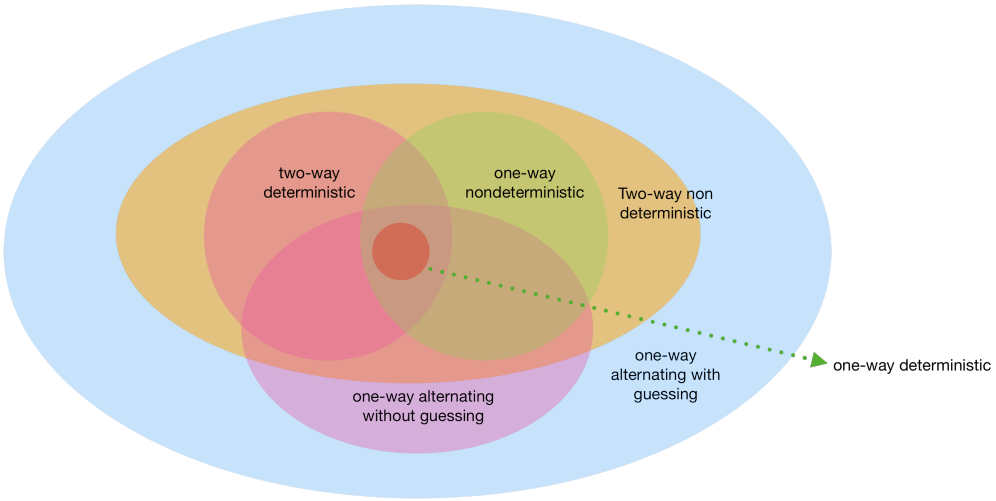


Figure 3.6: Relationships between various models of register automata.

We show languages that separate some of the models. Consider the language L where last letter never appears before:

$$L = \{a_1 a_2 \dots a_n \in \mathbb{A}^* \ : \ a_n \neq a_i \text{ for all } 1 \leq i < n\}$$

This language is recognised by a non-deterministic register automaton but not by any deterministic one. The language L_3 (3.4) from Example 3.1.5 is recognised by a two-way automaton and not by any one-way model. In fact all the regions in Figure 3.6 are non-empty and contain some language[20].

There are numerous other variations on the model of register automata studied in the literature, to mention just a few: *fresh-register automata*[21], and *history-register automata*[22]. There are also models of different nature, still recognizing data languages, for instance *data automata*[23] and *class automata*[24].

3 Register context-free grammars

Similarly like non-deterministic register automata are an extension of non-deterministic finite automata with registers, register context-free grammars are an extension of classical context-free grammars. For the sake of this thesis we provide the definition of context-free grammars with one register only.

[20] Bojańczyk, 2019, Excercise 28.
[21] Tzevelekos, 2011.
[22] Grigore and Tzevelekos, 2016.
[23] Bojańczyk, David, Muscholl, et al., 2011.
[24] Bojańczyk and Lasota, 2010.

Definition 3.3.1: (one-register context-free grammars)

A *one-register context-free grammar* (1-CFG) $\mathcal{G}$ consists of: two finite sets H and Q of terminals and nonterminals respectively, an initial nonterminal $q_0 \in Q$, and two finite sets Δ_2 and Δ_0 of binary and nullary *production rules*, of the forms

$$\begin{aligned} q(x) &\xrightarrow{\varphi} p(y)\ p'(y') \ \in\ \Delta_2, \\ q(x) &\to \varepsilon \ \in\ \Delta_0, \end{aligned} \tag{3.5}$$

where $q \in Q$, $p, p' \in Q \cup H$, and $\varphi(x, y, y')$ is a Boolean combination of equalities involving only the variables x, y, y'.

Similarly as before, a configuration $\langle q, (a)\rangle \in Q \times \mathbb{A}$ of $\mathcal{G}$, written also as $q(a)$, consists of a non-terminal $q \in Q$ and a register value $a \in \mathbb{A}$. Elements of $\Sigma = H \times \mathbb{A}$ we denote either as $h(a)$ or as $\langle h, a\rangle$. Production rules (3.5) induce *productions*

$$\begin{aligned} q(a) &\to p(b)\ p'(b'), \\ q(a) &\to \varepsilon, \end{aligned} \tag{3.6}$$

the former one under the condition $(a, b, b') \models \varphi$. We denote by Π_2 and Π_0, respectively, the (infinite) sets of productions induced by the rules from Δ_2 and Δ_0.

The semantics of a 1-CFG is defined as for classical context-free grammars, with configurations considered as non-terminals, input alphabet $\Sigma = H \times \mathbb{A}$, and productions $\Pi_2 \cup \Pi_0$ (all these sets are in general infinite, but orbit-finite). Derivation trees $\mathcal{T}$ of $\mathcal{G}$ are labeled by configurations, alphabet letters $\langle h, a\rangle = h(a) \in \Sigma$, or the empty word ε, in a way consistent with productions (3.6):

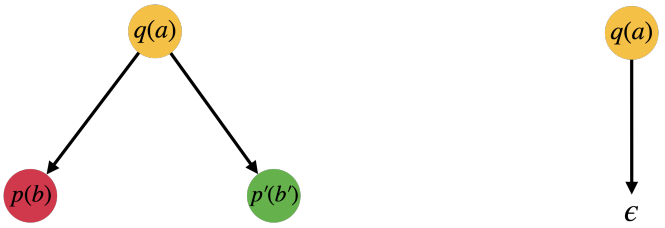


Figure 3.7: Pieces of derivation trees corresponding to the productions (3.6).

Complete derivation trees have all leaves labeled by elements of $\Sigma \cup \{\varepsilon\}$. We use below a similar notation as for NRA. We write $L_{q(a)}(\mathcal{G}) \subseteq \Sigma^*$ for the language of yields of all complete derivation trees $\mathcal{T}$ with root labeled by $q(a)$, as usual, where $\textsc{yield}(\mathcal{T}) \in \Sigma^*$ is obtained as concatenation of labels of all the leaves of $\mathcal{T}$, from left to right. The language $L(\mathcal{G}) \subseteq \Sigma^*$ generated by $\mathcal{G}$ is defined as the union (as in case of k-NRA, the initial register value is guessed

non-deterministically):

$$L(\mathcal{G}) \;=\; \bigcup_{a \in \mathbb{A}} L_{q_0(a)}(\mathcal{G}).$$

Remark 3.3.2. (Context-free languages) Context-free languages (with registers) over infinite alphabets were first introduced by Cheng and Kaminski[25]. They also showed the equivalence of register-grammars and register pushdown automata, as in the finite-alphabet case. Later this equivalence was generalized to arbitrary orbit-finite grammars and orbit-finite pushdown automata, over an arbitrary orbit-finite alphabet[26]. Orbit-finite pushdown automata, as well as their variant, *timed* pushdown automata, have been extensively studied recently[27], in order to show decidability of the reachability (emptiness) problem for these automata, and effective computability of the reachability sets. As in the finite case it is possible to extend register pushdown automata to orbit-finite Turing machines [28]. This model is interesting on its own, and has often different properties than the classical one. For example, it has been shown that non-deterministic orbit-finite Turing machines are more powerful than deterministic ones.

Example 3.3.3. (1-CFG) As an illustration, consider the 1-CFG consisting of non-terminals $Q = \{\bullet, \bullet\}$, terminals $H = \{\bullet, \bullet\}$, initial non-terminal $\bullet$, and rules

$$\bullet(x) \xrightarrow{x \neq y = y'} \langle \bullet, y \rangle \bullet(y') \qquad \bullet(x) \xrightarrow{x = y = y'} \bullet(y) \langle \bullet, y' \rangle \qquad \bullet(x) \to \varepsilon.$$

The first rule rewrites a non-terminal $\bullet$ into a terminal $\bullet$ and a non-terminal $\bullet$, both with the same value of register which is arbitrary but different than the initial register value of $\bullet$. Then, the second rule rewrites non-terminal $\bullet$ back into a non-terminal $\bullet$ and a terminal $\bullet$, both with the same register value as $\bullet$. Therefore the application of the two rules may rewrite $\bullet(a)$, where $a \in \mathbb{A}$, into

$$\langle \bullet, b \rangle \, \bullet(b) \, \langle \bullet, b \rangle$$

for any atom $b \neq a$. Derivation continues in this way until the non-terminal $\bullet$ disappears, using the third production rule. The grammar generates therefore palindrome-like words of the form

$$\langle \bullet, a_1 \rangle \langle \bullet, a_2 \rangle \ldots \langle \bullet, a_n \rangle \, \langle \bullet, a_n \rangle \ldots \langle \bullet, a_2 \rangle \langle \bullet, a_1 \rangle$$

where $n \geq 0$ and $a_1 \neq a_2 \neq \ldots \neq a_n$. For instance, Figure 3.8 shows a derivation tree generating the string $\langle \bullet, b \rangle \langle \bullet, a \rangle \langle \bullet, a \rangle \langle \bullet, b \rangle$.

[25] Cheng and Kaminski, 1998.

[26] Bojańczyk, Klin, and Lasota, 2014.

[27] Clemente and Lasota, 2015a, Clemente and Lasota, 2015b, Clemente, Lasota, Lazic, and Mazowiecki, 2017, Murawski, Ramsay, and Tzevelekos, 2017, Clemente and Lasota, 2018, Clemente, Lasota, Lazic, and Mazowiecki, 2019, Clemente and Lasota, 2021.

[28] The model, under the name of *Turing machines with atoms*, has been introduced by Bojańczyk, Klin, Lasota, and Toruńczyk, 2013.

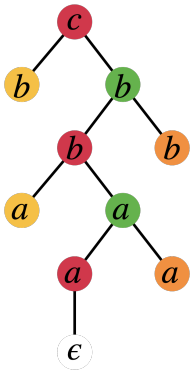


Figure 3.8: A derivation tree.

Remark 3.3.4. (Arity of production rules) According to our definition, production rules of register context-free grammars are of arity 2 (binary rules) or 0 (mullary rules). Other arities can be simulated easily: unary rules (arity 1) are simulated by a binary rule followed by a nullary one, while rules of arity higher than 2 are simulated by a sequence of binary rules (this may require increasing of the number of registers in general).

Remark 3.3.5. (One register) In the rest of this thesis, we focus mostly on automata and context-free grammars with one register, except for only Chapter 7 in which we introduce and investigate a subclass of NRA that extends 1-NRA.

Summary

We introduced register automata and register grammars. In forthcoming chapters, our main objective is to study Parikh images of their languages. Our main task is to check to what extent Parikh's Theorem (cf. Theorem 3.1.10) lifts to the setting of infinite (but orbit-finite) alphabets.

Chapter 4

Rational sets over orbit-finite alphabets

Objective

In the previous chapter, we have seen that languages of non-deterministic register automata do not satisfy certain standard properties like closure under complementation. This chapter starts by introducing a natural extension of rational expressions to orbit-finite alphabets, and showing that it is not expressive enough to define all the languages of NRA (not even the languages of deterministic one-register automata). Then, we focus on *Parikh images* of languages of register automata (alternatively, one can think of commutative closures of languages of register automata). We reinterpret rational expressions from languages to their Parikh images, obtaining *rational set of data vectors*. Then we discuss some of the properties of these rational sets. Finally, we motivate with examples that rational sets may be a suitable formalism for characterising Parikh images of the languages of NRA.

Contents

1 Rational data languages

Languages recognised by finite automata (regular languages) are characterised by rational (regular) expressions. We can naturally ask if a similar characterisation exists for the languages of register automata. All the proposals of rational expressions for register automata appearing in the literature either apply to a restricted subclass of the model, or go beyond the monoid of data words and introduce involved syntax significantly extending the classical concept of regular expressions[1]. It thus seems that none of these proposal may be considered as fully satisfactory.

In this thesis we propose and investigate a different but very natural approach, where we stick to the monoid of data words, and replace finite unions by *orbit-finite* ones. Our main objective is to prove that so obtained rational expressions, while not capturing languages of NRA, do capture their commutative images.

In this chapter we consider orbit-finite unions of sets of data vectors, or, formally speaking, unions of families of sets of data vectors indexed by an orbit-finite set. Given a finitely supported function f that maps each element of an orbit-finite set X (the domain of f) – an index – to a (hereditarily) finitely supported set of data vectors, the union

$$\bigcup_{x \in X} f(x)$$

is a (hereditarily) finitely supported set of data vectors as well. In particular, the support of f always supports the union, by Lemma 2.2.13.

Below we consider sets of data words (data languages), and later also sets of data vectors, over a fixed orbit-finite alphabet Σ.

Definition 4.1.1: (Rational data languages)

We define concatenation of two data languages $LL' = \{ww' : w \in L, w' \in L'\}$, and the Kleene star (iteration): $L^* = \{w_1 \dots w_n : n \geq 0, w_1, \dots, w_n \in L\}$, as usual. Let *rational data languages* be the smallest class of data languages that contains all singleton languages $\{a\}$, for a single-letter word $a \in \Sigma$, and is closed under concatenation, Kleene star, and orbit-finite unions.

In particular, the empty language $\emptyset$ is rational, as s sum of the empty family of sets. Moreover, the singleton of the empty word, $\{\varepsilon\}$, is a rational language definable as $\emptyset^*$. Using concatenation, we deduce that every singleton $\{w\}$, for $w \in \Sigma^*$, is a rational language. Therefore, using finite (resp. orbit-finite) unions we may derive all finite (resp. orbit-finite) languages, and hence all these languages are rational, as expected.

Similarly as in the case of rational (regular) languages over finite alphabets, we will write down rational data languages using *rational expressions* that denote derivations of rational languages according to Definition 4.1.1.

[1] Bojańczyk, 2020, Libkin, Tan, and Vrgoc, 2015, Kurz, Suzuki, and Tuosto, 2012, Kaminski and Tan, 2006.

Example 4.1.2. (Rational expressions) Let the alphabet be $\Sigma = \mathbb{A}$. The language $\mathbb{A}^{(3)}$ from Example 3.2.11 is expressed by the following rational expression:

$$\bigcup_{(a,b,c)\in\mathbb{A}^{(3)}} \{a\}\{b\}\{c\}.$$

As a convention, in order to simplify the expressions we omit the brackets of singletons. For instance, we write down the above rational expression as follows:

$$\bigcup_{(a,b,c)\in\mathbb{A}^{(3)}} abc.$$

The language L_2 from Example 3.2.15 is expressed by the following rational expression:

$$\bigcup_{a\in\mathbb{A}} \mathbb{A}^* \, a \, \mathbb{A}^* \, a \, \mathbb{A}^*,$$

where $\mathbb{A}$ (slightly overloading the notation) stand for the language of all data words of length 1:

$$\mathbb{A} = \bigcup_{a\in\mathbb{A}} a.$$

For finite alphabets Σ we obtain the classical rational (regular) sets of words or vectors. As expected, without the Kleene star we obtain exactly sets of words of bounded length, or equivalently, due to Lemma 3.1.12, orbit-finite languages.

The natural question is to ask if the languages of all NRA are rational. The following proposition provides a negative answer:

Proposition 4.1.3: (Non-rational language)

Let $\Sigma = \mathbb{A}$. The language L from Example 3.2.7:

$$L = \{a_1 a_2 \dots a_n \in \Sigma^* : a_1 \neq a_2 \neq \cdots \neq a_n\}$$

is not rational.

Proof. Indeed, towards contradiction suppose L is rational, and hence generated by a rational expression R. Consider the sublanguage $L' \subset L$ containing words in which all atoms are different. The language L' is orbit-infinite and hence it cannot be generated without star; indeed, concatenation and orbit-finite sums preserve orbit-finiteness of languages. Therefore, there must be a star subexpression R' of R such that the number of iterations of R' is unbounded in generation of words in L'. In other words, for every $n \in \mathbb{N}$ there is a word $w \in L'$ whose some infix u is generated by at least n iterations of R'. Thus $w = w'uw''$, the infix u splits into $u = u_1 \dots u_n$, and each of factors u_i is generated by R'. Choose n sufficiently large, namely $n > 2 \cdot |\textsc{supp}(R')|$

(considering union operations as atom-binding constructs, the support of a rational expression R' consists of those atoms appearing in R' which are not bounded by any union). As no atom repeats twice in words in L', some of words u_i are fresh for R', i.e., $\text{SUPP}(u) \cap \text{SUPP}(R') = \emptyset$, and u_i is either preceded or succeeded in w by an atom $a \notin \text{SUPP}(R')$. Consider w.l.o.g. the first case, and let a' be the first atom in u_i. Necessarily $a \neq a'$. As R' is invariant under the swap $a' \leftrightarrow a$, applying this swap to u_i yields a word u' still generated by R'. Replacing u_i by u' in w yields a word w' still generated by R, but $w' \notin L$ as it contains two consecutive atoms a. The contradiction completes the proof. □

Remark 4.1.4. (Other proposals of rational expressions) Kaminski and Tan introduced regular expressions for infinite alphabets by extending classical regular expressions, and proved their equivalence to a subclass of non-deterministic register automata[2]. Later, different extensions of regular expressions suitable for infinite alphabets were studied by process calculus community and automata community. In particular, regular expressions with *name binders* were introduced and shown expressively equivalent to non-deterministic register automata[3], and regular expressions with *finite memory* were developed and shown equivalent to non-deterministic register automata as well[4]. Lately, regular expressions with *atom automorphisms* were proposed and shown equivalent to NRA[5].

2 Rational sets of data vectors

Recall that a data vector is a finite multiset of letters from some orbit-finite alphabet Σ, or a function $\Sigma \to \mathbb{N}$ that maps almost all elements of Σ to 0. Each finite subset of Σ is a special case of a multiset, as hence can be seen as a data vector that maps each its element to 1 and all other elements of Σ to 0. In particular, each singleton $\{\sigma\}$, for $\sigma \in \Sigma$, can be also seen as a data vector.

Recall also that Parikh image of a data word w is a data vector that counts the number of occurrences of each letter in w. For example, the data word $w = a_1a_2a_1$ has Parikh image $\{a_1 \to 2, a_2 \to 1\}$. We can naturally extend this definition to any data language L: Parikh image of L is a set of those data vectors which are Parikh images of words in L.

In the sequel, we consider sets of data vectors over a fixed orbit-finite alphabet Σ. We define a class of sets of data vectors, which we call rational sets, and investigate their basic properties.

Recall that addition of data vectors (seen as functions $\Sigma \to \mathbb{N}$) is pointwise. Let addition of two such sets X, Y of data vectors be defined by *Minkowski sum*

$$X + Y = \{x + y : x \in X, y \in Y\},$$

[2] Kaminski and Tan, 2006.
[3] Kurz, Suzuki, and Tuosto, 2012.
[4] Libkin, Tan, and Vrgoc, 2015.
[5] Bojańczyk, 2020.

and let the Kleene star X^* of X contain all finite sums of elements of X:

$$X^* = \{x_1 + \ldots + x_n \,:\, n \geq 0, x_1, \ldots, x_n \in X\}.$$

In particular, X^* always contains the empty (zero) vector.

Definition 4.2.1: (Rational sets of data vectors)

Consider a fixed orbit-finite alphabet Σ. We define *rational sets of data vectors* over Σ as the smallest class of sets of data vectors that contains all singletons $\{\sigma\}$, for $\sigma \in \Sigma$, and is closed under Minkowski sum, Kleene star, and orbit-finite unions. In particular, the empty set, all finite sets and all orbit-finite sets of data vectors are rational.

By the very definition, Parikh image of a rational language is a rational set of data vectors (cf. Lemma 4.2.4 below). Let us see few examples to get acquainted with rational sets. As in case of rational languages, we rely on the concept of rational expression that denote derivations of rationals sets according to Definition 4.2.1.

Example 4.2.2. (Rational sets of data vectors) Let the alphabet be $\Sigma = \mathbb{A}$. Consider the language in Example 4.1.2 (recall the convention to omit brackets of singleton sets):

$$L = \bigcup_{(a,b,c) \in \mathbb{A}^{(3)}} abc.$$

Parikh image of this language consists of all vectors that contain three different atoms, each occurring exactly once (we write $a \in \Sigma$ to denote a data vector that maps a to 1 and all other alphabet letters to 0, and keep omitting brackets of singleton sets):

$$\textsc{Par}(L) = \bigcup_{(a,b,c) \in \mathbb{A}^{(3)}} a + b + c.$$

Example 4.2.3. (Parikh-equivalence to a rational language) Consider a variant of the language in Example 3.2.7 consisting of those words of even length where each two consecutive letters are different:

$$L_1 = \{a_1 a_2 \ldots a_n \in \Sigma^* : n \text{ even},\ a_1 \neq a_2 \neq \cdots \neq a_n\}.$$

The language is Parikh-equivalent to a larger language L_2, where the non-equality constraint is imposed at every second position only:

$$L_2 \;=\; \{a_1 a_2 \ldots a_n \in \mathbb{A}^* \,:\, n \text{ even},\ a_1 \neq a_2,\ a_3 \neq a_4,\ \ldots\},$$

which is rational, as it is definable by the following rational expression:

$$L_2 \;=\; \Big(\bigcup_{(a,b)\in\mathbb{A}^{(2)}} ab \Big)^*. \tag{4.1}$$

Indeed, every $w \in L_2$ can be transformed, by swapping letters, to a word in L_1. Let $w = a_1 a_2 \dots a_n$. If $a_2 = a_3$ we swap non-equal letters a_3 and a_4 thus achieving $a_1 \neq a_2 \neq a_3 \neq a_4$. Next, if $a_4 = a_5$ we swap analogously a_5 and a_6, and so on. Continuing in this way we finally arrive at a word in L_1.

Parikh image of the rational language L_2 is necessarily rational, and hence so is Parikh image of L_2:

$$\textsc{Par}(L_1) \;=\; \textsc{Par}(L_2) \;=\; \Big(\bigcup_{(a,b)\in\mathbb{A}^{(2)}} a+b \Big)^*.$$

The following two facts describe the relationship between rational data languages and rational set of data vectors.

Lemma 4.2.4: (Rational sets and rational languages)

Rational sets of data vectors are exactly Parikh images of rational data languages.

Proof. In one direction, let L be a rational language, i.e. L is defined by a rational expression. In order to show that Parikh image of L is rational, we construct a rational expression defining $\textsc{Par}(L)$ by replacing each singleton language $\{a\}$, where $a \in \Sigma$, by the corresponding singleton set of data vectors $\{a\}$, and replacing each concatenation $\cdot$ of languages by addition $+$ of data vectors. In the opposite direction, given a rational expression defining a set X of data vectors, we do the reverse replacement, thus obtaining a rational expression defining a language L. Thus L is rational and $\textsc{Par}(L) = X$ as required. (Note that in general, the language L so obtained is not the unique one satisfying $\textsc{Par}(L) = X$.) □

In the sequel we will often refer implicitly to the following corollary of Lemma 4.2.4:

Corollary 4.2.5: (Rationality of Parikh image)

$\textsc{Par}(L)$ is rational if, and only if, L is Parikh-equivalent to a rational data language.

3 Closure properties

Consider a language L over an orbit-finite alphabet Σ and an indexed family of languages $K = (K_\sigma)_{\sigma\in\Sigma}$ over an alphabet Γ, indexed by Σ. We typically use the anonymous function notation

$$\sigma \quad \mapsto \quad K_\sigma.$$

The *substitution* $L(K)$ is the language over Γ containing all words obtained from some word $\sigma_1\sigma_2\ldots\sigma_n \in L$, by replacing every letter σ_i by some word from K_{σ_i}:

$$L(K) \;=\; \bigcup_{\sigma_1\sigma_2\ldots\sigma_n\in L} K_{\sigma_1}K_{\sigma_2}\ldots K_{\sigma_n}.$$

We note that the sum above is not orbit-finite in general.

Example 4.3.1. (Substitution) As usual we use the shorthand $L^+ = L^*L$. Let $\Sigma = \Gamma = \mathbb{A}$, and consider the language L_1 from Example 4.2.2. By the equivariant substitution $K_a = (aa)^+$, or

$$a \;\mapsto\; (aa)^+,$$

we obtain the language $L_1(K) = \left(\bigcup_{a\in\mathbb{A}} aa\right)^+$ containing words, where all maximal constant infixes have even length.

Lemma 4.3.2: (Substitution Lemma)

If $L \subseteq \Sigma^*$ and all languages K_σ, for $\sigma \in \Sigma$, have rational Parikh images (resp. are rational) then the substitution $L(K)$ has also rational Parikh image (resp. is rational).

Proof. Intuitively speaking, it is enough to replace syntactically, in the rational expression defining $\textsc{Par}(L)$ (resp. L), every appearance of a letter σ by an expression defining $\textsc{Par}(K_\sigma)$ (resp. K_σ).

Formally, suppose L and all languages K_σ have rational Parikh images. By Lemma 4.2.4 we assume, w.l.o.g., that languages L and K_σ are rational. We proceed by structural induction on a rational expression defining L and prove that $L(K)$ is rational too. If $L = \{\sigma\}$ is a singleton, for some $\sigma \in \Sigma$, then $L(K) = K_\sigma$ and hence is rational. The cases of $L = L_1 \cdot L_2$, or $L = (L')^*$, are both immediate, as both the operations commute with substitutions:

$$(L_1 \cdot L_2)(K) = L_1(K) \cdot L_2(K) \qquad (L')^*(K) = (L'(K))^*,$$

and preserve rationality, and $L_1(K), L_2(K)$ and $L'(K)$ are rational by induction assumption. Finally, when $L = \bigcup_{x\in X} L_x$, by induction assumption we know rationality of the languages $L_x(K)$ for $x \in X$. As

$$L(K) \;=\; \bigcup_{x\in X} L_x(K)$$

and the mapping $x \mapsto L_x(K)$ is supported by the union of the supports of $x \mapsto L_x$ and $\sigma \mapsto K_\sigma$, we deduce that $L(K)$ is an orbit-finite union of rational languages and hence rational. □

Substitution Lemma is intensively used in the proofs in the following chapters.

The next result shows that rationality is preserved when the alphabet is restricted to its subset:

Lemma 4.3.3: (Closure under restriction)

If a language $L \subseteq \Sigma^*$ has rational Parikh image (resp. is rational) and $\Gamma \subseteq \Sigma$, then the restriction $L \cap \Gamma^*$ has also rational Parikh image (resp. is also rational).

Proof. Intutively speaking, it is enough to syntactically remove, in the rational expression defining Par(L), every appearance of a letter $\sigma \in \Sigma - \Gamma$.

Formally, we proceed by induction on a derivation of L. By Lemma 4.2.4, we assume that the language L is rational. The induction base: when $L = \{\sigma\}$ is singleton, $\sigma \in \Sigma$, then

$$L \cap \Gamma^* = \begin{cases} L & \text{if } \sigma \in \Sigma \\ \emptyset & \text{otherwise} \end{cases}$$

and in each case, $L \cap \Gamma^*$ is rational. The induction step follows immediately, as restriction commutes with all the operations involved:

$$(LK) \cap \Gamma^* = (L \cap \Gamma^*)(K \cap \Gamma^*)$$
$$L^* \cap \Gamma^* = (L \cap \Gamma^*)^*$$
$$\left(\bigcup_{i \in I} L_i\right) \cap \Gamma^* = \left(\bigcup_{i \in I} L_i \cap \Gamma^*\right)$$

□

Summary

In this chapter we proposed a natural extension of rational expressions to orbit-finite alphabets. Then, we introduced rational sets of data vectors and, using a small example, we motivated rational sets as a good candidate to express Parikh images of languages of register automata. Finally, we proved the substitution property of rational sets that will be intensively used in forthcoming chapters. In the following chapters we will work with rational sets of data vectors which can be obtained as Parikh images of languages of non-deterministic one-register automata, and one-register context-free grammars.

Chapter 5

Semilinear sets over orbit-finite alphabets

Objective

Previously, we saw that the natural extension of rational (regular) expressions does not capture languages of register automata, and introduced rational sets of data vectors which seem to be a natural candidate to characterise Parikh images of languages of register automata. This chapter explores semilinear sets, naturally adapted to the orbit-finite setting, and its relationship with register automata and rational sets.

Contents

1 Semilinear sets with orbit-finite unions

We propose the following natural lifting of the notion of semi-linear set[1] to the setting of orbit-finite alphabets. Recall the Kleene star operation on sets of data vectors: given such a set X, the set X^* contains all finite sums of elements of X:

$$X^* = \{x_1 + \ldots + x_n \,:\, n \geq 0, x_1, \ldots, x_n \in X\}.$$

Definition 5.1.1: (Semi-linear set)

Consider data vectors over a fixed orbit-finite alphabet Σ. A *linear set* is then any set of the form

$$N \;=\; \{g\} + P^*,$$

for a data vector g (a *base* vector) and an orbit-finite set P of data vectors (*periods*). We omit brackets and write simply $g + P^*$. A semi-linear set is any orbit-finite union of linear sets:

$$\bigcup_{i \in I} N_i \quad = \quad \bigcup_{i \in I} g_i + P_i^*, \tag{5.1}$$

for a finitely supported mapping $i \mapsto N_i$, where $i \in I$ ranges over an orbit-finite indexing set I.

Example 5.1.2. Parikh image of the language L_1 from Example 3.1.5,

$$L_1 = \{a_1 a_2 \ldots a_n \in \mathbb{A}^* : a_1 = a_i \text{ for some } i \in \{2, \ldots, n\}\},$$

over the input alphabet $\Sigma = \mathbb{A}$, is the set of all data vectors where some atoms appears at least twice, and is hence semi-linear:

$$I \;=\; \bigcup_{a \in \mathbb{A}} a \;=\; \mathbb{A}, \qquad g_a \;=\; a + a, \qquad P_a \;=\; \bigcup_{b \in \mathbb{A}} b \;=\; \mathbb{A}.$$

Above, $a + a$ denotes a data vector that maps a to 2 and all other atoms to 0. As all sets P_i are equal, this set can be also written as:

$$\Big(\bigcup_{a \in \mathbb{A}} a + a \Big) \;+\; \Big(\bigcup_{b \in \mathbb{A}} b \Big)^*.$$

This is an instance of a *hybrid linear*[2] set, i.e., of a set of the form

$$B + P^*,$$

[1] Parikh, 1966, Chistikov and Haase, 2016.
[2] This terminology is used for instance in Chistikov and Haase, 2016.

for orbit-finite sets B and P.

2 Semilinear sets are strictly contained in rational sets

As we demonstrate below, semi-linear sets are exactly as expressive as rational sets of *star-height* 1, where star-height of a set X of data vectors is defined, similarly as for finite alphabets, as the smallest depth of nesting of the Kleene star operation in a rational expression defining X.

Theorem 5.2.1: (Semilinear sets)

Semi-linear sets of data vectors are exactly rational sets of star-height at most 1.

Proof. Every semilinear set is, by definition, a rational set of star-heigth at most 1. For the converse inclusion we use a distributive law of Minkowski sum over orbit finite unions:

Lemma 5.2.2: (Distributive law)

$$\bigcup_{i\in I} L_i \quad + \quad \bigcup_{j\in J} K_j \quad = \quad \bigcup_{\langle i,j\rangle\in I\times J} L_i + K_j.$$

Note that the Cartesian product $I \times J$ of orbit-finite sets I and J is necessarily orbit-finite[3].

Consider a rational set X of data vectors of star-height $h \leq 1$. If $h = 0$, by the repetitive use of the distributive law (from left to right) we deduce that the set X is orbit-finite and hence vacuously semi-linear. If $h = 1$, by the distributive law we similarly deduce that, for every star subexpression Y^*, the set Y is orbit-finite. Moreover, again using the distributive law, we obtain that the set X is an orbit-finite union

$$X \quad = \quad \bigcup_{i\in I} X_i, \tag{5.2}$$

where each X_i is a sum of star subexpressions Y^* and orbit-finite sets. As Minkowski sum is commutative, preserves orbit-finiteness, and admits merging of stars:

$$Y^* \; + \; Z^* \quad = \quad (Y \cup Z)^*,$$

each of sets X_i is of the form

$$Z + Y^*$$

where Y and Z are both obit-finite. Therefore each X_i is hybrid linear, and hence semi-linear. In consequence, the orbit-finite union (5.2) is semi-linear too. □

[3] cf. Bojańczyk, 2019, Sect. 3.

3 Semilinear sets are not sufficient

As the main result of this chapter we prove that semi-linear sets do not capture Parikh images of languages of register automata, which excludes possibility of a straightforward generalisation of Parikh theorem to register automata. The idea of the proof comes from Masters thesis of Marta Juzepczuk[4], where non-semi-linearity has been shown for one-register context-free grammars. We improve the construction and show that Parikh images need not be semi-linear already for (deterministic) one-register automata.

Theorem 5.3.1

Parikh images of languages of 1-NRA are not always semilinear.

This negative result motivates consideration of rational sets, instead of only semi-linear ones, in forthcoming chapters. Recall that semi-linear and rational sets of vectors coincide for finite alphabets[5], and therefore both semi-linear and rational sets are equally good candidates to be considered for generalisation of Parikh theorem to orbit-finite alphabets.

3.1 Proof of Theorem 5.3.1

We demonstrate that Parikh images of 1-NRA languages are not semi-linear in general. As a counterexample we take the following language $L \subseteq \mathbb{A}^*$ over the alphabet $\Sigma = \mathbb{A}$. For $a \in \mathbb{A}$, let

$$K_a \;=\; \bigcup_{b \in \mathbb{A}\setminus\{a\}} b \;=\; \mathbb{A} \setminus \{a\}; \qquad L_a \;=\; aa\,(a\,K_a)^*. \tag{5.3}$$

Let L be the language obtained from

$$L_1 = \{a_1 a_2 \dots a_n \in \Sigma^* : n \geq 1,\ a_1 \neq a_2 \neq \dots \neq a_n\}$$

by the substitution:

$$a \;\mapsto\; L_a. \tag{5.4}$$

The language is clearly rational, and recognised by a (deterministic) one-register automaton depicted below in Figure 5.1. Here the red and green locations are initial and accepting, respectively.

[4] Juzepczuk, 2013.
[5] Eilenberg and Schützenberger, 1969.

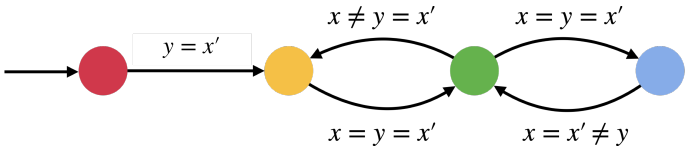


Figure 5.1: A deterministic one-register automaton recognising L.

Remark 5.3.2. (Segments and pairs) Each word $w \in L$ is obtained from some word $a_1 a_2 \dots a_n \in L_1$ by the substitution (5.4), and hence splits into n *segments*, each segment being an infix of w that belongs to L_a for some $a \in \mathbb{A}$. Furthermore, each word in L_a consists of a number of *pairs* of atoms, where the first pair is aa and each consecutive one is of the form ab where $b \neq a$.

As we show, its Parikh image is not semi-linear:

Lemma 5.3.3: (Non-semilinearity)

$\textsc{Par}(L)$ is not semi-linear.

Proof. We start with some preparatory definitions and their properties. For a data vector $p : \mathbb{A} \to \mathbb{N}$, let $\textsc{Sing}(p) = |\{a \in \mathbb{A} : p(a) = 1\}|$ denote the number of atoms appearing exactly once in p (such atoms we call *singular*). We will often rely on the fact that the function $\textsc{Sing}(\_)$ is *sub-additive* with respect to addition of data vectors:

Claim 5.3.4

$\textsc{Sing}(v + w) \leq \textsc{Sing}(v) + \textsc{Sing}(w)$.

Our argument relies on a careful analysis of the limit value of the *singularity ratio* $\frac{\textsc{Sing}(p)}{|p|}$, for $p \in \textsc{Par}(L)$, when $|p|$ tends to infinity. We will exploit the fact that the singularity ratio is *non-increasing* with respect to addition of data vectors:

Claim 5.3.5

$$\frac{\textsc{Sing}(v + w)}{|v + w|} \leq \max\left\{\frac{\textsc{Sing}(v)}{|v|}, \frac{\textsc{Sing}(w)}{|w|}\right\}.$$

For every $a \in \mathbb{A}$ and $p \in \textsc{Par}(L_a)$, we have $\textsc{Sing}(p) < \frac{1}{2}|p|$. Indeed, the first pair aa introduces no singular atom, and each consecutive pair introduces at most one singluar atom. As $\textsc{Sing}(\_)$ is sub-additive with respect to addition of data vectors, we immediately deduce:

Claim 5.3.6

Every $p \in \textsc{Par}(L)$ satisfies $\textsc{Sing}(p) < \frac{1}{2}|p|$.

Using an analogous argument one obtains:

Claim 5.3.7

Every $p \in \textsc{Par}(L)$ satisfies $\textsc{dom}(p) \leq \frac{1}{2}|p|$.

Towards contradiction, suppose $\textsc{Par}(L)$ is semilinear:

$$\textsc{Par}(L) \quad = \quad \bigcup_{i \in I} g_i + P_i^*.$$

For a data vector $p : \mathbb{A} \to \mathbb{N}$ and $S \subseteq \mathbb{A}$, we denote by $p - S$ the data vector obtained from p by removing all occurrences of atoms from S:

$$(p - S)(a) = \begin{cases} p(a) & \text{if } a \notin S \\ 0 & \text{otherwise.} \end{cases}$$

Let $S_i = \textsc{supp}(P_i)$. As a consequence of Claim 5.3.6, we get:

Claim 5.3.8

Every $p \in P_i$ $(i \in I)$ satisfies $\textsc{Sing}(p - S_i) \leq \frac{1}{2}|p|$.

Proof. Towards contradiction, suppose $\textsc{Sing}(p - S_i) > \frac{1}{2}|p|$ for some $p \in P_i$. Let $S' = \textsc{dom}(p - S_i)$. For an arbitrary permutation of atoms $\pi \in \textsc{Perm}$ such that $\pi(a) = a$ for all $a \in S_i$, we have $\pi(p) \in P_i$. Consider such permutations $\pi_1, \ldots, \pi_n \in \textsc{Perm}$, such that $\pi_k(S')$ and $\pi_l(S')$ are disjoint for $k \neq l$. This guarantees that singular atoms in $\pi_1(p), \ldots, \pi_n(p)$ are distinct. As $|g_i|$ is fixed, for sufficiently large n the data vector

$$v = \pi_1(p) + \ldots + \pi_n(p)$$

satisfies $\textsc{Sing}(v) \geq \textsc{Sing}(v - S_i) \geq \frac{1}{2}|v| + \frac{3}{2}|g_i|$, and in consequence

$$\textsc{Sing}(g_i + v) \geq \textsc{Sing}(v) - |g_i| \geq \frac{1}{2}|g_i + v|.$$

Therefore, the data vector

$$g_i + v \ \in \ \textsc{Par}(L)$$

contradicts Claim 5.3.6. This completes the proof. □

We call a data vector $p : \mathbb{A} \to \mathbb{N}$ *non-singular* if $p(a) > 1$ for some $a \in \mathbb{A}$. Claim 5.3.8 can be strengthened as long as non-singular data vectors are considered:

Claim 5.3.9

Every $p \in P_i$ $(i \in I)$ such that $p - S_i$ is non-singular, satisfies $\textsc{Sing}(p - S_i) < \frac{1}{2}|p|$.

Proof. Indeed, suppose $\textsc{Sing}(p - S_i) \geq \frac{1}{2}|p|$ for some $p \in P_i$, and hence $\textsc{dom}(p - S_i) > \frac{1}{2}|p|$ due to non-singularity of $p - S_i$. Considering similar permutations of p as in the argument for Claim 5.3.8, we contradict Claim 5.3.7. □

Let $k = \max\{|S_i| : i \in I\}$ be the maximal size of the support of P_i; note that k is well defined as the family of sets $\{P_i\}_{i \in I}$ is orbit-finite, and the size of the support is invariant inside an orbit. Likewise, let $t = \max\{|g_i| : i \in I\}$ be the maximal size of a base (t is well-defined due to Lemma 3.1.12, as the set of orbits $\{g_i : i \in I\}$ is orbit-finite) and let $s = \max\{|p| : p \in \bigcup_i P_i\}$ be the maximal size of a period (s is likewise well-defined as $\bigcup_i P_i$ is orbit-finite, cf. Lemma 2.3.4).

Let $Z = \{a_0, \ldots, a_k\} \subseteq \mathbb{A}$ be some fixed $k+1$ atoms. A word $v \in L_a$ (cf. (5.3)) we call *varied* if all atoms different than a appear at most once in v. For every $m \in \mathbb{N}$ choose some arbitrary but fixed word $w_m \in L$ of the form:

$$w_m \;=\; v_0\, v_1 \,\ldots\, v_k \;\in\; L_{a_0}\, L_{a_1} \,\ldots\, L_{a_k}, \tag{5.5}$$

where each $v_i \in L_{a_i}$ is a varied word of length $2m$ and no atom appears in two distinct words v_{a_i}, v_{a_j}, for $i \neq j$. Thus each atom $a_i \in Z$ appears $m+1$ times in w_m. Let $q_m = \textsc{Par}(w_m)$. Hence $|w_m| = |q_m| = 2m(k+1)$. As $\textsc{Sing}(q_m) = (m-1)(k+1)$, in the limit we have:

$$\lim_{m \to \infty} \frac{\textsc{Sing}(q_m)}{|q_m|} \;=\; \lim_{m \to \infty} \frac{(m-1)(k+1)}{2m(k+1)} \;=\; \frac{1}{2}, \tag{5.6}$$

irrespectively of the choice of the words w_m. For every $m \in \mathbb{N}$, let $g_{i_m} + {P_{i_m}}^*$ $(i_m \in I)$ be a linear set to which q_m belongs. Thus

$$q_m = g_{i_m} + p_m,$$

for $p_m \in {P_{i_m}}^*$. Recalling (5.5), for every $m \in \mathbb{N}$ choose $a_{j_m} \in Z$ $(j_m \in \{0, \ldots, k\})$ so that $a_{j_m} \notin S_{i_m}$ (such a_{j_m} exists as $|S_{i_m}| \leq k$). We split p_m as follows:

$$q_m \;=\; (g_{i_m} + p_{m,0}) \;+\; p_{m,1} \;+\; p_{m,>1}, \tag{5.7}$$

where $p_{m,1}$ is a sum of vectors from P_{i_m} that contain exactly one appearance of a_{j_m}; $p_{m,>1}$ is a sum of vectors from P_{i_m} that contain more than one appearance of a_{j_m}; and $p_{m,0}$ is a

sum of vectors from P_{i_m} that contain no appearance of a_{j_m} at all. Applying Claim 5.3.6 to $g_{i_m} + p_{m,0} \in \textsc{Par}(L)$, we obtain:

$$\limsup_{m\to\infty} \frac{\textsc{Sing}(g_{i_m} + p_{m,0})}{|g_{i_m} + p_{m,0}|} \leq \frac{1}{2}. \tag{5.8}$$

Observe that the sum of the last two data vectors in (5.7) contains all $m+1$ occurrences of a_{j_m} in q_m except for at most $|g_{i_m}|$ of them possibly appearing in the base g_{i_m}. As

$$m = \frac{1}{2(k+1)}|q_m|,$$

the size of $p_{m,1} + p_{m,>1}$ plus the size of g_{i_m} constitutes at least $\frac{1}{2(k+1)}$ fraction of the whole size $|q_m|$:

$$\frac{1}{2(k+1)}|q_m| - |g_{i_m}| \;<\; |p_{m,1} + p_{m,>1}|. \tag{5.9}$$

We are going to prove the following strict inequality:

$$\limsup_{m\to\infty} \frac{\textsc{Sing}(p_{m,1} + p_{m,>1})}{|p_{m,1} + p_{m,>1}|} < \frac{1}{2}. \tag{5.10}$$

Before providing its proof, we notice that this inequality, together with inequalities (5.8) and (5.9), contradicts the equality (5.6) and thus completes the proof of Lemma 5.3.3. Indeed, since $|p_{m,1} + p_{m,>1}|$ constitutes a constant fraction of $|q_m|$ and the singularity ratio of $p_{m,1} + p_{m,>1}$ is strictly below $\frac{1}{2}$, while the singularity ratio of the remaining part of q_m is at most $\frac{1}{2}$, by sub-additivity of $\textsc{Sing}(\_)$ with respect to addition of data vectors we deduce:

$$\limsup_{m\to\infty} \frac{\textsc{Sing}(q_m)}{|q_m|} < \frac{1}{2}.$$

From now on we concentrate on proving the inequality (5.10). Call $p_{m,1}$ *non-trivial* if it is a sum of at least two vectors from P_m. When $p_{m,1}$ is trivial, $|p_{m,1}| \leq s$ is bounded and hence $p_{m,1}$ can be ignored in (5.10). We split the inequality (5.10) into two separate ones

$$\limsup_{m\to\infty} \frac{\textsc{Sing}(p_{m,1})}{|p_{m,1}|} < \frac{1}{2} \qquad\qquad \limsup_{m\to\infty} \frac{\textsc{Sing}(p_{m,>1})}{|p_{m,>1}|} < \frac{1}{2} \tag{5.11}$$

and prove the first one assuming that $p_{m,1}$ is non-trivial for infinitely many m, and the second one unconditionally. This is enough to derive (5.10), again relying on sub-additivity of $\textsc{Sing}(\_)$ with respect to addition of data vectors.

Concerning the first inequality, we observe that the atom $a_{j_m} \notin S_{i_m}$ is counted in $\textsc{Sing}(v - S_{i_m})$ for every data vector $v \in P_{i_m}$ contributing to the sum $p_{m,1}$, but if there are more than one of these data vectors v, then the atom a_{j_m} is no more counted in $\textsc{Sing}(p_{m,1} - S_{i_m})$. Thus the singularity ratio of $p_{m,1} - S_{i_m}$ loses, intuitively speaking, at least the $\frac{1}{s}$ fraction (recall that s is the maximal size of a period) of the maximal possible value $\frac{1}{2}|p_{m,1}|$ according to Claim 5.3.8.

This allows us to deduce:

$$\frac{\textsc{Sing}(p_{m,1} - S_{i_m})}{|p_{m,1}|} \leq \frac{1}{2} \cdot \left(1 - \frac{1}{s}\right) < \frac{1}{2}$$

which implies, in the limit, the first inequality in (5.11), as $|S_{i_m}|$ is bounded (by k).

Concerning the second inequality, let's put

$$r = \max\{\frac{\textsc{Sing}(v - S_i)}{|v|} \ : \ i \in I, v \in P_i, \ v - S_i \text{ is non-singular}\}.$$

As before, r is well defined. Indeed, the value of

$$\frac{\textsc{Sing}(v - S_i)}{|v|} \tag{5.12}$$

is invariant inside every S_i-orbit included in P_i, and hence each period set P_i admits a bounded number of these values. Furthermore, for two $i, i' \in I$ in the same orbit, the period sets P_i and $P_{i'}$ admit the same values of (5.12), and hence the total number of these values admitted by all the period sets P_i, $i \in I$, is bounded.

Furthermore, $r < \frac{1}{2}$ by Claim 5.3.9. A crucial observation is that by a multiple application of the fact that singularity ratio is non-increasing (cf. Claim 5.3.5) we get:

$$\frac{\textsc{Sing}(p_{m,>1} - S_{i_m})}{|p_{m,>1}|} \leq \frac{\textsc{Sing}(v - S_{i_m})}{|v|}$$

for some $v \in P_{i_m}$ that contributes to the sum $p_{m,>1}$, and hence

$$\frac{\textsc{Sing}(p_{m,>1} - S_{i_m})}{|p_{m,>1}|} \leq r < \frac{1}{2}.$$

As $|S_{i_m}|$ is bounded, the last inequality implies, in the limit, the second inequality in (5.11).

Both the inequalities (5.11) are thus proved. □

Remark 5.3.10. (Semi-linear sets are a strict subclass of rational ones) As a consequence of Theorem 5.3.1 (Parikh images of 1-NRA languages are not always semi-linear), together with the results of the next section, namely Theorem 6.1.1 (Parikh images of 1-NRA languages are always rational), semi-linear sets are strictly contained in rational sets of data vectors. Furthermore, the proof of Lemma 5.3.3 applies also to a larger language

$$L' = \Big(\bigcup_{a \in \mathbb{A}} L_a \Big)^*$$

which shows not only that semi-linear sets are a strict subset of rational sets, but also of rational sets of star-height 2 (as the languages L_a have star-height 1).

Summary

In this chapter, we lifted semi-linear sets to the setting of orbit-finite alphabets, and investigated possibility of likewise lifting of Parikh's theorem. Our findings are negative: we provided a counter-example language, recognized by a deterministic one-register automaton, whose Parikh image is not semi-linear. Therefore semi-linear sets are not sufficient to capture Parikh images of languages of register automata, even of 1-NRA, which sharply distinguishes register automata from finite automata. On the other hand, Parikh image of the counter-example language is a rational set (as defined in the previous chapter). This opens up the question if all languages of register automata have rational Parikh images. We explore, and partially answer this question in the following chapters.

Chapter 6

Parikh images of nondeterministic one-register automata

Objective

In previous chapters we showed that semilinear sets are not expressive enough to capture Parikh images of NRA, and proposed rational sets instead. In this chapter we positively verify this choice for 1-NRA.

Contents

1 Overview

The principal content of this chapter is the proof of the fact that Parikh images of languages of non-deterministic one-register automata are rational:

Theorem 6.1.1: (Rationality)

Parikh images of 1-NRA languages are rational.

This is the most involved proof of this thesis.

For technical convenience, we actually prove a refined version of Theorem 6.1.1 stated in Lemma 6.1.2 below which, due to Definition 3.2.6, implies Theorem 6.1.1. For two configurations $q(a)$ and $q'(a')$ of a automaton $\mathcal{A}$, we define the language $L_{q(a)\,q'(a')}(\mathcal{A})$ as the set of all data words in which $\mathcal{A}$ starts at $q(a)$ and end at $q'(a')$.

Lemma 6.1.2: (Rationality of 1-NRA)

For every 1-NRA $\mathcal{A}$, the languages $L_{q(a)\,q'(a')}(\mathcal{A})$ have rational Parikh images.

After preparatory Sections 2 and 3, we proceed in three major steps. As the first major step, in Sections 4–5 we do the first simplification step due to adapting a classical transition-elimination technique of proving Kleene's theorem to the setting of Parikh images of 1-NRA. As the second major step, in Sections 6–9 we repeatedly simplify the problem further, thus reducing the question of rationality of Parikh images of 1-NRA languages to rationality of certain canonical languages, over extended alphabets. As the last step, in Section 10 we apply some known graph-theoretic tools to achieve our goal.

In a more fine-grained description, the proof of Lemma 6.1.2 proceeds by a sequence of simplifying steps, as stated in consecutive Lemmas 6.4.3, 6.5.2, 6.6.2, 6.7.5, 6.8.2 and 6.9.3 in the forthcoming sections, that ends in the two final lemmas based on graph-theoretic insights, namely Lemmas 6.10.10 and 6.10.14. Instead of only considering Parikh images of input words, in the proof we investigate Parikh images of *runs*, mostly concentrating on *alternations* of register value along a run. This leads us to consider, besides languages over the alphabet $H \times \mathbb{A}$ of an 1-NRA, also languages over richer alphabets:

- languages of *altering paths* over the alphabet $(Q \times \mathbb{A} \times Q) \cup (H \times \mathbb{A})$ in Lemma 6.4.3;
- languages of *altering loops* over the alphabet $\mathbb{A}^2 \times \mathbb{A}$ in Lemma 6.5.2;
- languages of *anti-paths* and *anti-cycles* over $\mathbb{A} \times \mathcal{P}_2(\mathbb{A})$ in Lemmas 6.6.2, 6.7.5, 6.8.2 and 6.9.3.

The intuitive idea underlying the final, most technical steps (Lemmas 6.6.2–6.9.3) is, roughly speaking, that the set of words

$$\langle a_1, b_1\rangle \langle a_2, b_2\rangle \ldots \langle a_n, b_n\rangle$$

over $\mathbb{A}^2$, satisfying $b_i \neq a_{i+1}$ for all $i = 1, \ldots, n-1$, has rational Parikh image. Notably, this is not true for *paths*, where one requires $b_i = a_{i+1}$ instead.

In the proof of the two final lemmas, Lemma 6.10.10 and 6.10.14, we use a known result providing sufficient conditions for Hamiltonian cycles in directed graphs.

2 Normal form

In the sequel we assume that the constraint in every transition rule of 1-NRA defines exactly one orbit in $\mathbb{A}^3$.

Definition 6.2.1: (Normal form of 1-NRA)

A 1-NRA is in *normal form* if the constraint in every transition rule is one of the following five formulas (cf. Example 2.2.18):

(1) $\varphi_1 \equiv x = x' = y$,

(2) $\varphi_2 \equiv x = x' \neq y$,

(3) $\varphi_3 \equiv x \neq y = x'$,

(4) $\varphi_4 \equiv x = y \neq x'$,

(5) $\varphi_5 \equiv x \neq y \neq x' \neq x$.

The constraints (1)-(2) are called *register-preserving* and the constraints (3)-(5) are called *register-updating*.

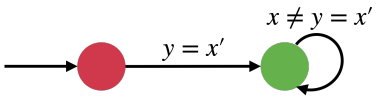
Note that the transition constraint φ_3 updates the register by the input atom, while the transition constraints φ_4 and φ_5 update the register by a fresh value different from the current register value and the input atom. Note also the symmetry between φ_3 and φ_4.

Proposition 6.2.2: (Normal form of 1-NRA)

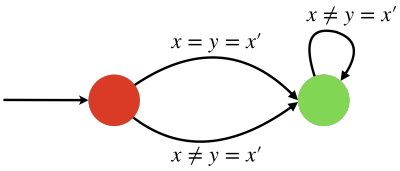
Every 1-NRA can be transformed to an equivalent 1-NRA in normal form.

Indeed, it is enough to split each transition rule into finitely many ones whose constraints are of the form φ_1 - φ_5.

Example 6.2.3. The 1-NRA from Example 3.2.2 in Chapter 3



is transformed into the following one in normal form:



3 Rationality for register-preserving transition rules

Consider a fixed 1-NRA, $\mathcal{A} = \langle H, Q, I, F, \Delta \rangle$. For locations $q, p \in Q$ of $\mathcal{A}$ and $a \in \mathbb{A}$, let L_{qap} be the language of all data words read by a run from configuration $q(a)$ to $p(a)$ that use register-preserving transitions only (thus the register stores a along the whole run). The following lemma shows the rationality of the language L_{qap}. Intuitively, these are essentially languages of finite automata, since the register values are not updated.

Lemma 6.3.1

The languages L_{qap} are rational.

Proof. We only need to consider register-preserving transitions. Define the finite alphabet $\Delta = H \times \{\varphi_1, \varphi_2\}$ and consider every transition rule $(q(x), \langle h, y \rangle, \varphi, q'(x'))$ to be *labeled* by $\langle h, \varphi \rangle \in \Delta$.

Fix q, a and p and let E_{qp} be the classical regular expression over Δ defining the labels of all those runs from q to p that only use transitions of types (1) and (2). Then the language L_{qap} is defined by the expression E_{qap} obtained from E_{qp} by replacing (h, φ_1) with $\langle h, a \rangle$ and replacing $\langle h, \varphi_2 \rangle$ with

$$L_{e,\neq a} \quad = \quad \bigcup_{b \in \mathbb{A} - \{a\}} \langle h, b \rangle.$$

Thus L_{qap} is a rational data language. □

4 Altering paths

In the previous section we dealt with register-preserving transition rules only. We now introduce the language of *altering paths* which is an important tool in dealing with 1-NRA with register-updating transitions.

Definition 6.4.1: (Altering paths)

Define the language P over the alphabet $(Q \times \mathbb{A} \times Q) \cup (H \times \mathbb{A})$ containing words of the form ($n \geq 1$):

$$\langle q_1, a_1, p_1\rangle\langle h_1, b_1\rangle\langle q_2, a_2, p_2\rangle\langle h_2, b_2\rangle \dots \langle q_n, a_n, p_n\rangle \tag{6.1}$$

such that $p_i(a_i) \xrightarrow{\langle h_i, b_i\rangle} q_{i+1}(a_{i+1})$ is a register-updating transition for $i = 1, \dots, n-1$ (in particular $a_i \neq a_{i+1}$ for $i = 1, \dots, n-1$). Words in P we call *altering paths*.

Furthermore, define the subsets $P_{q(a)\,q'(a')} \subseteq P$ of those altering paths as in (6.1) where $q(a) = q_1(a_1)$ and $q'(a') = p_n(a_n)$, for configurations $q(a)$ and $q'(a')$.

Example 6.4.2. (Altering paths) Intuitively, an altering path $p \in P_{q(a)\,q'(a')}$, represents a run of $\mathcal{A}$ from configuration $q(a)$ to $q'(a')$ by emphasizing register-updating transitions. Consider the automaton depicted in Example 6.2.3. An example of an altering path for that automaton is:

$$\langle \bullet, a_1, \bullet\rangle\ \langle a_2\rangle\ \langle \bullet, a_2, \bullet\rangle\ \langle a_3\rangle\ \langle \bullet, a_3, \bullet\rangle$$

where $a_1 \neq a_2$ and $a_2 \neq a_3$. Altering paths are a generalisation of the language L in Example 3.2.7 in Chapter 3:

$$L = \{a_1 a_2 \dots a_n \in \Sigma^* : a_1 \neq a_2 \neq \dots \neq a_n\}.$$

We now state the central lemma that generalises the reasoning given in Example 3.2.7:

Lemma 6.4.3: (Rationality of altering paths)

Altering path languages $P_{q(a)\,q'(a')}$ have rational Parikh images.

Its proof is complicated, and consists of a sequence of steps in the forthcoming sections. Once we have Lemmas 6.3.1 and 6.4.3, we are able to prove the main technical result of this chapter, Lemma 6.1.2:

Proof of Lemma 6.1.2. Indeed, $L_{q(a)\,q'(a')}(\mathcal{A})$ is obtained from the altering path language $P_{q(a)\,q'(a')}$ using the equivariant substitution (q, p range over locations and a, b over $\mathbb{A}$):

$$\langle q, a, p\rangle \quad \mapsto \quad L_{qap} \qquad\qquad \langle h, b\rangle \quad \mapsto \quad \langle h, b\rangle.$$

As a substitution by languages with rational Parikh images preserves rationality of Parikh image, cf. Substitution Lemma (Lemma 4.3.2), by Lemmas 6.3.1 and 6.4.3 we deduce that the languages $L_{q(a)\,q'(a')}(\mathcal{A})$ have rational Parikh images, as required. □

In the rest of this chapter we focus on proving Lemma 6.4.3. We proceed by a sequence of simplification steps in Sections 5–9, and provide the final argument in Section 10. We intensively use Substitution Lemma without explicitly referring to it.

5 Altering loops

As the first step of proving Lemma 6.4.3 that speaks of altering paths, we reduce it to Lemma 6.5.2, stated below, that speaks of *altering loops*.

Definition 6.5.1: (Altering loops)

We define, for a register-updating transition constraint $\varphi \in \{\varphi_4, \varphi_5\}$ and (not necessarily distinct) atoms $a', b, a \in \mathbb{A}$, the language $L_{(a',b)\varphi a}$ over the alphabet $\mathbb{A}^2 \times \mathbb{A}$ as follows: let $L_{(a'_0,b_0)\varphi a_{n+1}}$ contain all (possibly empty) words of the form

$$\langle\langle a_1, a'_1\rangle, b_1\rangle\ \langle\langle a_2, a'_2\rangle, b_2\rangle\ \dots\ \langle\langle a_n, a'_n\rangle, b_n\rangle \tag{6.2}$$

such that $(a'_i, b_i, a_{i+1}) \models \varphi$ for $i = 0, \dots, n$. We omit the case $\varphi = \varphi_3$ as it is can be treated symmetrically to the case $\varphi = \varphi_4$. Words in $L_{(a'_0,b_0)\varphi a_{n+1}}$ we call *altering loops*.

Intuitively, a letter $\langle\langle d, d'\rangle, e\rangle \in \mathbb{A}^2 \times \mathbb{A}$ represents, for some fixed locations p', p and $h \in H$, an altering path from $p(d)$ to $p'(d')$ followed by a register-updating transition that inputs $\langle h, e\rangle$ and returns from location p' to p.

We state the main result about altering loops:

Lemma 6.5.2: (Rationality of altering loops)

Altering loop languages $L_{(a',b)\varphi a}$ have rational Parikh images.

The proof of Lemma 6.5.2 is provided in the next section. In the remaining part of this section we show how Lemma 6.5.2 implies Lemma 6.4.3:

Proof of Lemma 6.4.3. We mimic the standard proof of Kleene's theorem, exploiting altering loops to capture all iterations along loops in $\mathcal{A}$. We proceed by induction on the number of register-updating transition rules in $\mathcal{A}$. If there is no such transition rules, we have trivial (and obviously rational) altering path languages

$$P_{q(a)\,q'(a')} = \begin{cases} \{\langle q, a, q'\rangle\} & \text{if } a = a', \\ \emptyset & \text{otherwise.} \end{cases}$$

Otherwise, remove an arbitrary register-updating transition rule

$$t = (p'(x), \langle h, y\rangle, \varphi, p(x'))$$

from $\mathcal{A}$ (if $\varphi = \varphi_3$ consider the inverse of $\mathcal{A}$ and $\varphi = \varphi_4$ instead), and use the induction assumption for the so obtained automaton $\mathcal{A}'$ to get altering path languages $K_{q(a)\,q'(a')}$ for every locations q, q' and atoms a, a', with rational Parikh images. Let $L_{(c',b)h\varphi c}$ be the language obtained from the altering loop language $L_{(c',b)\varphi c}$ by the equivariant substitution (d, d', e range over $\mathbb{A}$)

$$\langle\langle d, d'\rangle, e\rangle \quad \mapsto \; K_{p(d)\,p'(d')}\, \langle h, e\rangle. \tag{6.3}$$

Rationality of Parikh images of the altering path languages $P_{q(a)\,q'(a')}$ of $\mathcal{A}$ follows by the fact that $P_{q(a)\,q'(a')}$ is equal to the union of $K_{q(a)\,q'(a')}$ and the following set

$$\bigcup_{c',b,c\in\mathbb{A}} K_{q(a)\,p'(c')}\, \langle h, b\rangle\, L_{(c',b)h\varphi c}\, K_{p(c)\,q'(a')}. \tag{6.4}$$

To show the equality, we observe that $K_{q(a)\,a'(q')}$ contains all altering paths in $\mathcal{A}$ that do not use t, and claim that the set (6.4) contains those altering paths in $\mathcal{A}$ that do use t. Specifically, as $\varepsilon \in L_{(c',b)h\varphi c}$, we obtain altering paths using t exactly once, as shown in Figure 6.1 (dotted arrow depict altering paths in $\mathcal{A}'$),

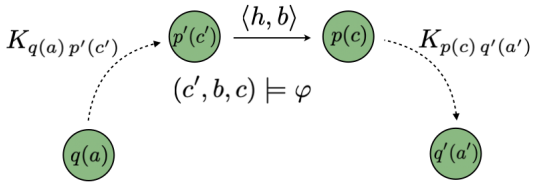


Figure 6.1: Kleene type argument.

or more than once (for instance twice, as shown in Figure 6.2):

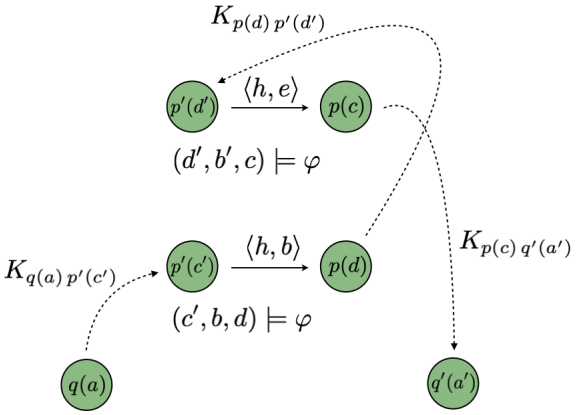


Figure 6.2: Kleene type arguument, cont.

In general, a word in (6.4) factorises into a prefix before the first use of t (an altering path from $q(a)$ to $p'(c')$), the suffix after the last use of t (an altering path from $p(c)$ to $q'(a')$), and the infix leading from $p'(c')$ to $p(c)$. The infix starts with the letter $\langle h, b\rangle$ input by the first traversal of t, and then contains alternately altering paths that do not use t (from $p(d)$ to $p'(d')$, for some $d, d' \in \mathbb{A}$) and traversals of t (a letter $\langle h, e\rangle$ for some $e \in \mathbb{A}$), cf. the substitution (6.3). Therefore, by the definition (6.2) of altering loops $L_{(a',b)\varphi a}$, the set (6.4) contains exactly those altering paths in $\mathcal{A}$ that do use t, as claimed. □

6 Anti-paths

It remains to demonstrate Lemma 6.5.2 only. As the first step, we reduce this lemma to a slightly simplified one, namely Lemma 6.6.2 stated below.

We concentrate on the hardest case $\varphi = \varphi_5$ (all the three atoms involved in φ are pairwise distinct). The remaining case $\varphi = \varphi_4$ is obtained then using the substitution

$$\langle\langle a, a'\rangle, b\rangle \quad \mapsto \quad \langle\langle a, b\rangle, b\rangle.$$

We need to show that the altering loop languages $L_{(c,b)\varphi_5 a}$ have rational Parikh images. Recall that $L_{(c_0,b_0)\varphi_5 a_{n+1}}$ contains all words over $\mathbb{A}^2 \times \mathbb{A}$ of the form

$$\langle\langle a_1, c_1\rangle, b_1\rangle\ \langle\langle a_2, c_2\rangle, b_2\rangle\ \ldots\ \langle\langle a_n, c_n\rangle, b_n\rangle \tag{6.5}$$

such that c_i, b_i, a_{i+1} are pairwise different for $i = 0, \ldots, n$.

Relying on the observation that, in case of $\varphi = \varphi_5$, atoms b_i and c_i play entirely symmetric roles in (6.5) and are forcedly distinct, we rearrange words (6.5) into words over the alphabet $\Gamma = \mathbb{A} \times \mathcal{P}_2(\mathbb{A})$ as follows:

$$\langle a_1, \{b_1, c_1\}\rangle\ \langle a_2, \{b_2, c_2\}\rangle\ \ldots\ \langle a_n, \{b_n, c_n\}\rangle. \tag{6.6}$$

Now we are ready to define *anti-paths*, which are required to show rationality of Parikh images of altering loop languages of Lemma 6.5.2.

Definition 6.6.1: (Anti-paths)

Let $\mathbf{P}_{\{b_0,c_0\}a_{n+1}} \subseteq \Gamma^*$ denote the language of all *nonempty* words of the form (6.6) subject to the same constraints as in (6.5), namely $a_{i+1} \notin \{b_i, c_i\}$ for $i = 0, \ldots, n$; these words we call *anti-paths*.

Note that $a_i \in \{b_i, c_i\}$ is allowed. We state the main result of this section, which is helpful to prove the rationality of the altering loops:

Lemma 6.6.2: (Rationality of anti-paths)

The anti-path languages $\mathbf{P}_{\{b,c\}a} \subseteq \Gamma^*$ have rational Parikh images.

Before proving Lemma 6.6.2 in the next section, we use it now to prove Lemma 6.5.2:

Proof of Lemma 6.5.2. We observe that $L_{(c,b)\varphi_5 a}$ is obtained from $\mathbf{P}_{\{b,c\}a}$ using the equivariant substitution

$$\langle d, \{e, f\}\rangle \mapsto \langle d, e\rangle\, f \quad \cup \quad \langle d, f\rangle\, e,$$

and adding the empty word. Therefore the language $L_{(c,b)\varphi_5 a}$ has rational Parikh image assuming $\mathbf{P}_{\{b,c\}a}$ has so, and Lemma 6.5.2 is implied. □

From now on we focus on proving Lemma 6.6.2.

7 Anti-cycles

In order to show rationality of Parikh images of anti-path languages, we focus on a specific kind anti-path, called *anti-cycles*.

Definition 6.7.1: (Source and target)

Let $\Gamma = \mathbb{A} \times \mathcal{P}_2(\mathbb{A})$. For a letter $\alpha = \langle a, \{b, c\}\rangle \in \Gamma$ we call the atom a its *source*, and the two-element set $\{b, c\}$ its *target*, denoted $a = \textsc{src}(\alpha)$ and $\{b, c\} = \textsc{trg}(\alpha)$, respectively. For a word $w = \alpha_1 \dots \alpha_n \in \Gamma^*$ we denote by $\textsc{src}(w) = \textsc{src}(\alpha_1)$ the first source, and by $\textsc{trg}(w) = \textsc{trg}(\alpha_n)$ the last target.

Definition 6.7.2: (Anti-cycles)

An anti-path w is called an *anti-cycle* if $\textsc{src}(w) \notin \textsc{trg}(w)$ (the first source does not belong to the last target).

Anti-cycles are closed under cyclic shifts, and hence we use the cyclic order when speaking about precedence of letters in anti-cycles. Denote the set of all anti-cycles by $\mathbf{C}$. We build on a simple but crucial observation: anti-paths $\mathbf{P}_{\{b,c\}a}$ are exactly those words $w \in \Gamma^*$ which, prolonged with a single letter $w\,\langle a, \{b, c\}\rangle \in \Gamma$, form an anti-cycle:

Claim 6.7.3

$\mathbf{P}_{\{b,c\}a} = \{w \in \Gamma^* \,:\, w\,\langle a, \{b,c\}\rangle \in \mathbf{C}\}$.

Next we show a technical lemma, that is used later to prove Lemma 6.6.2. Roughly speaking, this lemma says that rationality of a language is preserved if we were to drop a fixed letter that appears at the end of the word.

Lemma 6.7.4

If $L \subseteq \Gamma^*$ is rational and $\alpha \in \Gamma$ then the language $L \triangleleft \alpha = \{w \in \Gamma^* \,:\, w\,\alpha \in L\}$ is rational too.

Proof. We transform a rational expression E defining a language $L \subseteq \Gamma^*$ into a rational expression $\widetilde{E}$ defining $L \triangleleft \alpha$. We proceed by structural induction on E. In case of orbit-finite union the transformation is distributive:

$$\widetilde{\bigcup_{i\in I} E_i} \quad := \quad \bigcup_{i\in I} \widetilde{E_i}.$$

In case of concatenation, the transformation is applied to the second component only:

$$\widetilde{E_1\; E_2} \quad := \quad E_1\; \widetilde{E_2}.$$

In case of iteration, the transformation is applied to the single last iteration (which forces at least one iteration and hence rules out the vacuous generation of the empty data word ε due to 0 iterations):

$$\widetilde{E^*} \quad := \quad E^*\; \widetilde{E}\;.$$

Finally, the induction base, for a singleton $\{\beta\}$, is given by:

$$\widetilde{\beta} \quad := \quad \begin{cases} \mathbf{0} & \text{if } \beta = \alpha \\ \emptyset & \text{otherwise.} \end{cases}$$

□

Lemma 6.7.5: (Rationality of anti-cycles)

$\mathbf{C}$ has rational Parikh image.

As before, we postpone the proof of Lemma 6.7.5 to the next section, but we use it to prove rationality of Parikh images of anti-path languages:

Proof of Lemma 6.6.2. Indeed, let $L \subseteq \Gamma^*$ be rational and Parikh-equivalent to $\mathbf{C}$. By Claim 6.7.3, $\mathbf{P}_{\{b,c\}a}$ is Parikh-equivalent to $L \triangleleft \langle a, \{b, c\}\rangle$, which is rational by Lemma 6.7.4. Thus, in order to complete the whole proof of Lemma 6.1.2 it suffices to prove Lemma 6.7.5. □

8 Anti-cycles over a restricted alphabet

For a word $w = \alpha_1 \dots \alpha_n \in \Gamma^*$ we denote by $\textsc{srces}(w)$ the sequence $\textsc{src}(\alpha_1) \dots \textsc{src}(\alpha_n)$ of sources. For a finite subset $X \subset \mathbb{A}$ and a regular language $K \subseteq X^*$ we define:

$$\mathbf{C}_a^K \;=\; \{w \in \Gamma^* \,:\, \textsc{srces}(w) \in K, a \notin \textsc{trg}(w)\}.$$

Lemma 6.8.1

For every finite set $X \subset \mathbb{A}$ and regular language $K \subseteq X^*$, the languages $\mathbf{C}_a^K$ are rational.

Proof. Consider the finite set $\Delta = (X \cup \{a\})^2$ as an alphabet, and the regular language $P \subseteq \Delta^*$ of all $\{K, a\}$-*paths*, i.e., all nonempty sequences

$$\langle d_1, d_2\rangle \langle d_2, d_3\rangle \dots \langle d_n, d_{n+1}\rangle \;\in\; \Delta^*$$

such that $d_1 d_2 \dots d_n \in K$ and $d_{n+1} = a$. The language $\mathbf{C}_a^X$ is obtained from P by the substitution

$$\langle d, e\rangle \quad\mapsto \bigcup_{\{e', e"\} \in \mathcal{P}_2(\mathbb{A} - \{e\})} \langle d, \{e', e"\}\rangle$$

and is thus rational. □

We mostly focus on a special but central case of Lemma 6.7.5, namely we restrict to the sub-alphabet

$$\Sigma \;=\; \{\alpha \in \Gamma \,:\, \textsc{src}(\alpha) \notin \{b, c\}\} \;\subseteq\; \Gamma.$$

Lemma 6.8.2: (Rationality of anti-cycles over Σ)

The language $\mathbf{D} = \mathbf{C} \cap \Sigma^*$ has rational Parikh image.

As before, we postpone the proof of this lemma to the next section. Before that, we use Lemma 6.8.2 to prove Lemma 6.7.5, by reducing rationality of anti-cycles to rationality of anti-cycles over Σ.

Proof of Lemma 6.7.5. We show that rationality of $\textsc{Par}(\mathbf{D})$ implies rationality of $\textsc{Par}(\mathbf{C})$. To this aim we define, for distinct atoms $b, c \in \mathbb{A}$, the language

$$K_{bc} \;:=\; \mathbf{C}_b^{b\{b,c\}^*}\,\langle b,\{b,c\}\rangle \quad\cup\quad \mathbf{C}_c^{b\{b,c\}^*}\,\langle c,\{b,c\}\rangle$$

of all anti-paths where the last target is $\{b,c\}$, all sources are in $\{b,c\}$, and the first one is b. Languages K_{bc} are rational, due to Lemma 6.8.1. Further, for pairwise distinct atoms a, b, c we define the following rational language

$$K_{a\{b,c\}} \;:=\; \langle a,\{b,c\}\rangle \;\cup\; \bigcup_{b',c'\in\mathbb{A}-\{b\}} \langle a,\{b',c'\}\rangle\, K_{bc}.$$

Note that the source of the first letter in every word in $K_{a\{b,c\}}$ is a, and the target of the last letter is $\{b,c\}$. Lemma 6.7.5 follows once we show the following claim:

Claim 6.8.3

$\textsc{Par}(\mathbf{C})$ is obtained from $\textsc{Par}(\mathbf{D})$ by applying *twice* the substitution

$$\langle d,\{e,f\}\rangle \quad\mapsto\quad K_{d\{e,f\}}.$$

(We consider Parikh images of $\mathbf{C}$ and $\mathbf{D}$, instead of the languages themselves, only because we reason below up to cyclic shifts.) From now on we concentrate on the proof of the claim. Let $\widetilde{\mathbf{D}}$ denote the set of data vectors obtained from $\textsc{Par}(\mathbf{D})$ by applying twice the above-defined substitution. By the very definition, $\widetilde{\mathbf{D}} \subseteq \textsc{Par}(\mathbf{C})$. For the converse inclusion, we prove that every data vector $v \in \textsc{Par}(\mathbf{C})$ belongs to $\widetilde{\mathbf{D}}$.

If v contains no unwanted letters from $\Gamma - \Sigma$ then $v \in \textsc{Par}(\mathbf{D})$, and the claim follows due to $\textsc{Par}(\mathbf{D}) \subseteq \widetilde{\mathbf{D}}$.

Otherwise, choose an anti-cycle $w \in \mathbf{C}$ with $v = \textsc{Par}(w)$ and consider the last appearance of an unwanted letter $(b,\{b,c\}) \in \Gamma - \Sigma$ in w. Applying a cyclic shift ($\rightarrow$) we can assume, w.l.o.g., that the letter is the last one in w. Let u be the maximal suffix of w that belongs to K_{bc} (or, symmetrically, to K_{cb}):

$$w \quad=\quad w'\,u.$$

We observe that $w' \neq \varepsilon$; indeed, as $\textsc{src}(u) = b \in \textsc{trg}(u) = \{b,c\}$, the word u itself is not an anti-cycle.

Let $w' = w\text{"}\langle a,\{b',c'\}\rangle$; since w is an anti-chain we have $b \notin \{b',c'\}$, and by maximality of u we have $a \notin \{b,c\}$. Then $u' = \langle a,\{b',c'\}\rangle\, u \in K_{a,\{b,c\}}$. Replace the suffix u' by $\langle a,\{b,c\}\rangle$, thus obtaining a data word $\widetilde{w} = w\text{"}\langle a,\{b,c\}\rangle$ with smaller number of occurrences of unwanted letters. We continue in the same way with $\widetilde{w}$ untill all occurences of letters from $\Gamma - \Sigma$ are eliminated. A crucial observation is that during elimination of all letters, except for possibly the very last one, the total sum of cyclic shifts ($\rightarrow$) performed does not exceed the full cyclic shift of w. Therefore, Parikh image of the word obtained by elimination of all unwanted letters except for the last one,

belongs to the result of application the substitution once to $\mathbf{D}$. In consequence, the final word belongs to the result of applying the substitution twice, as required. □

9 Anti-cycles of (un)bounded order

In this section we define tools that are needed for proving Lemma 6.8.2, by reducing it to Lemma 6.9.3 stated below.

Definition 6.9.1: (Order of a data vector)

The number of different sources of letters appearing in a data vector $v : \Sigma \to \mathbb{N}$, i.e., the size of the set

$$V_v \;=\; \{\textsc{src}(\alpha) \,:\, \alpha \in \textsc{dom}(v)\}, \tag{6.7}$$

we denote by $\textsc{ord}(v)$ and call *the order* of v (clearly, an atom can be the source of more than one letter in $\textsc{dom}(v)$). The order of a data word $w \in \Sigma^*$ is defined naturally as $\textsc{ord}(w) = \textsc{ord}(\textsc{Par}(w))$.

We write $\mathbf{D}^{<n}$ (resp. $\mathbf{D}^{\geq n}$) for the subsets of $\mathbf{D}$ containing anti-cycles of order smaller than n (resp. at least n). Anti-cycles of bounded order can be easily dealt with separately:

Lemma 6.9.2: (Rationality of anti-cycles over Σ of bounded order)

For every $n \in \mathbb{N}$, the language $\mathbf{D}^{<n}$ is rational.

Proof. For a finite subset $X \subseteq \mathbb{A}$ the language

$$\mathbf{C}_X \;=\; \mathbf{C} \,\cap\, \{\alpha \in \Gamma \,:\, \textsc{src}(\alpha) \in X\}^*.$$

is rational, due to Lemma 6.8.1, as it equals

$$\bigcup_{a \in X} \mathbf{C}_a^{aX^*},$$

and hence so is its restriction $\mathbf{D}_X = \mathbf{C}_X \cap \Sigma^*$. The language $\mathbf{D}^{<n}$, being the union of all the rational languages $\mathbf{D}_X$ for subsets $X \subseteq \mathbb{A}$ of cardinality $< n$, is thus rational as well. □

Therefore, in the rest of the proof we need to concentrate only on anti-cycles or order at least n, for a sufficiently large $n \in \mathbb{N}$.

Lemma 6.9.3: (Rationality of anti-cycles over Σ of unbounded order)

For sufficiently large n, the language $\mathbf{D}^{\geq n}$ is rational.

We prove the lemma in the next section. Here, having Lemmas 6.9.2 and 6.9.3, we easily complete the proof of Lemma 6.8.2:

Proof of Lemma 6.8.2. For $n \in \mathbb{N}$ sufficiently large for Lemma 6.9.3 to hold, we decompose Parikh images of anti-cycles into

$$\textsc{Par}(\mathbf{D}) \;=\; \textsc{Par}(\mathbf{D}^{<n}) \;\cup\; \textsc{Par}(\mathbf{D}^{\geq n}),$$

both of them rational by Lemmas 6.9.2 and 6.9.3, respectively. Lemma 6.8.2 is thus proved. □

10 Anti-cycles of unbounded order and non-degenerate data vectors

This last section is devoted entirely to the proof of Lemma 6.9.3.

For the proof, we take a graph-theoretic view on Parikh images of anti-paths, and build on some known graph-theoretic tools. In the sequel we consider directed graphs without self-loops or parallel edges, but possibly containing tight two-vertex cycles.

Definition 6.10.1: (Source graphs)

Let $v : \Sigma \to \mathbb{N}$ be a fixed data vector. Guided by the crucial property of anti-paths that the source of every letter does not belong to the target of the preceding letter, we define the directed graph $\mathcal{G}_v = (V_v, E_v)$, called *source graph* induced by v: let the vertices V_v of $\mathcal{G}_v$ be the sources of all letters appearing in v, as defined in (6.7), and let $(d, e) \in E_v$ be an edge if, and only if

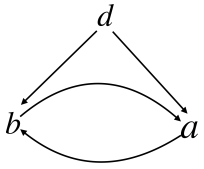
$$\exists \alpha \in \textsc{dom}(v) \colon d = \textsc{src}(\alpha), e \notin \textsc{trg}(\alpha).$$

Whenever $(d, e) \notin E_v$, for distinct atoms $d \neq e$, we say that d *excludes* e (or call (d, e) an *excluded* edge); equivalently, e belongs to the target of every letter in $\textsc{dom}(v)$ with source d:

$$\forall \alpha \in \textsc{dom}(v) \colon\; d = \textsc{src}(\alpha) \implies e \in \textsc{trg}(\alpha).$$

Note that an atom never excludes itself, due to restriction to Σ, and that $\mathcal{G}_v$ depends only on the set $\textsc{dom}(v) \subseteq \Sigma$ of letters appearing in v, and not on cardinalities of letters in v.

Example 6.10.2. (Source graph) Let $v : \Sigma \to \Gamma$ be the Parikh image of the word $\langle a, \{d, c\}\rangle\langle d, \{e, f\}\rangle\langle b, \{d, f\}\rangle \in \Gamma^*$. The following figure depicts the source graph of v:



The nodes $\{a, b, d\}$ are sources of letters in v. Atoms a and b exclude the atom d.

Let $\text{IN}(e) = \{d \in V_v : (d, e) \in E_v\}$ denote the set of in-neighbours of a vertex e, and let $\text{IN-DEG}(e) = |\text{IN}(e)|$ denote the *in-degree* of e. Symmetrically we define out-neighbours $\text{OUT}(e)$ and *out-degree* $\text{OUT-DEG}(e)$. Clearly, an atom may exclude at most two other atoms, and hence (let $n = \text{ORD}(v)$):

Claim 6.10.3

$\text{OUT-DEG}(d) \geq n - 3$ for every vertex $d \in V_v$.

Since every vertex can exclude 2 other vertices in a source graph and there are n vertices, we obtain the following:

Corollary 6.10.4

There are at most $2n$ excluded edges.

In the sequel we rely on Claim 6.10.3 and Corollary 6.10.4 according to which $\mathcal{G}_v$ is not much different from the full directed clique.

Let $v : \Sigma \to \mathbb{N}$ be a data vector. For $A \subseteq \text{DOM}(v)$, let $v_{|A}$ denote the restriction of v to A: $v_{|A}(\alpha) = v(\alpha)$ if $\alpha \in A$, and $v_{|A}(\alpha) = 0$ otherwise.

Lemma 6.10.5

For every simple cycle $\pi = a_1 a_2 \dots a_n$ in $\mathcal{G}_v$ there exists an anti-cycle w with $\text{PAR}(w) = v_{|A}$ where $A = \{\alpha \in \text{DOM}(v) : \text{SRC}(\alpha) \in \{a_1, a_2, \dots, a_n\}\}$.

Proof. We arrange the letters into an anti-path w by taking first all a_1-sourced letters in a consecutive block, then all a_2-sourced ones in a consecutive block, etc. The order of a_i-sourced letters inside a block (including repetitions of equal letters) is irrelevant as long as the last one, say α, satisfies $a_{i+1} \notin \text{TRG}(\alpha)$ (where $n + 1$ is identified cyclically with 1). □

Theorem 6.10.6: (Hamilton cycle)

Let $\mathcal{G}$ be a strongly connected directed graph with n vertices such that for every two vertices d, d', $\text{IN-DEG}(d) + \text{OUT-DEG}(d') \geq n$. Then $\mathcal{G}$ contains a Hamiltonian cycle [a].

[a] Ghouila-Houri, 1960, cf. also Thm. 1 in Kühn and Osthus, 2012.

The tool will be applicable due to the following observation:

Lemma 6.10.7

For sufficiently large n, a directed graph with n vertices such that $\text{IN-DEG}(d) \geq 3$ and $\text{OUT-DEG}(d) \geq n-3$ for every vertex d, is necessarily strongly connected.

Proof. Consider the decomposition of the graph into strongly connected components. As the first step we observe that there may be no singleton components $\{d\}$. Indeed, by the assumption we have $\text{IN-DEG}(d) + \text{OUT-DEG}(d) \geq n$, and hence d forms a tight 2-vertex cycle with some other vertex d'.

In the sequel we use Corollary 6.10.4. As the second step we argue that for sufficiently large n, a component $\{d, e\}$ of size 2 is impossible (and, in consequence, a component of size $n-2$ is impossible too). Towards contradiction, suppose $\{d, e\}$ is a strongly connected component (hence the two vertices form a tight cycle). In consequence, (a) the sets $V_d = \text{OUT}(d) - \{e\}$ and $V_e = \text{IN}(e) - \{d\}$ are disjoint, and (b) there is no edge from V_d to $V_e \cup \{d, e\}$. As $\text{OUT-DEG}(d) \geq 3$ and $\text{IN-DEG}(e) \geq 3$, we have $|V_d| \geq n-4$ and $|V_e| \geq 2$. By (a) we deduce $|V_d| = n-4$ and $|V_e| = 2$. By (b), all $4(n-4)$ edges from V_d to $V_e \cup \{d, e\}$ are excluded. This is impossible as long as $4(n-4) > 2n$.

Likewise one argues that there may be no component of size strictly between 2 and $n-2$. Indeed, supposing there is a component C of size k, for $2 < k < n-2$, no vertex in C may form a tight cycle with other vertex outside of C, and hence at least $k(n-k)$ edges are excluded. This is impossible as long as $k(n-k) > 2n$. As $k(n-k)$ reaches its minimum for $k = 3$ or $k = n-3$, there may be no component of size strictly between 2 and $n-2$ as long as $3(n-3) > 2n$. □

Let $\text{PRE}(d) = \{\alpha \in \text{DOM}(v) : d \neq \text{SRC}(\alpha), d \notin \text{TRG}(\alpha)\}$ denote the set of letters that can precede a d-sourced letter and have themselves source different than d. We now aim at characterising data vectors in the Parikh image of anti-cycles with unbounded order. The following example illustrates our approach.

Example 6.10.8. Consider the following words over Σ:

$$\langle a, \{b, c\}\rangle\ \langle d, \{a, c\}\rangle\ \langle e, \{g, a\}\rangle$$
$$\langle a, \{g, c\}\rangle\ \langle b, \{f, e\}\rangle\ \langle g, \{a, b\}\rangle\ \langle f, \{a, b\}\rangle$$
$$\langle a, \{b, e\}\rangle\ \langle b, \{a, g\}\rangle\ \langle a, \{h, b\}\rangle\ \langle l, \{m, n\}\rangle$$

It can be easily seen that none of the words can be rearranged to an anti-cycle. The first word can not because the node a has no in-neighbours in the source graph of its Parikh image v: $\text{IN}(a) = \emptyset$. The second one can not because two nodes a, b in the source graph of v are mutually their only in-neighbours: $\text{IN}(a) = \{b\}$ and $\text{IN}(b) = \{a\}$, and hence $\text{IN}(a) \cup \text{IN}(b) = \{a, b\}$. The last words can not because its Parikh image v contains only one letter that can precede a or b: $\text{PRE}(a) = \text{PRE}(b) = \langle l, \{m, n\}\rangle$, and $|v_{|\text{PRE}(a)\cup\text{PRE}(b)}| = 1$. Below we argue that these are the only reasons why a data vector is not a Parikh image of an anti-cycle of sufficiently large order.

Definition 6.10.9: (Non-degenerate vectors)

A data vector $v : \Sigma \to \mathbb{N}$ is called *non-degenerate* if its source graph satisfies the following:

(1) $\text{IN}(d) \neq \emptyset$ for every $d \in V_v$,

(2) $\text{IN}(d) \cup \text{IN}(e) \not\subseteq \{d, e\}$ for every non-equal $d, e \in V_v$,

(3) $|v_{|\text{PRE}(d)\cup\text{PRE}(e)}| \geq 2$ for every non-equal $d, e \in V_v$.

(1) excludes vertices of in-degree 0;

(2) excludes pairs of vertices d, e with $\text{IN}(d) = \{e\}$ and $\text{IN}(e) = \{d\}$;

(3) excludes the case when there is only one letter $\alpha \in \text{DOM}(v)$ that can precede d- or e-sourced letters, and moreover $v(\alpha) = 1$.

The following result characterises the anti-cycles with unbounded order using the notion the non-degenerate vectors:

Lemma 6.10.10

For data vectors $v : \Sigma \to \mathbb{N}$ of sufficiently large order, $v \in \text{PAR}(\mathbf{D})$ if, and only if v is non-degenerate.

Proof. Let $\mathcal{G}_v = (V_v, E_v)$ be the source graph and $n = \text{ORD}(v)$.

The 'only if' implication is immediate for data vectors of order at least 3. Indeed, suppose $v = \text{PAR}(w)$ for an anti-cycle $w \in \mathbf{D}$. By the definition of anti-cycles, $\text{IN}(d) \neq \emptyset$ for every $d \in V_v$ and hence (1) forcedly holds. The other two conditions are easily shown by contradiction. Indeed, if (2) fails for some $d, e \in V_v$ then every d- or e-sourced letter would be preceded in w by a d- or e-sourced one, which is impossible as long as $\text{ORD}(v) \geq 3$. Finally, if (3) fails then there are letters a and b which can be only preceded by the same letter α, as a consequence, w fails to be an anti-cycle (see the last word in Example 6.10.8 which fails to satisfy (3)).

For the 'if' implication, we assume that v is non-degenerate ((1)–(3) hold) and prove that $v = \textsc{Par}(w)$ for some $w \in \mathbf{D}$.

Let $k = 9$. Due to Corollary 6.10.4 we can assume n to be large enough so that:

Claim 6.10.11

At most two atoms in V_v have in-degree $< k$.

In other words, this means that there are no 3 atoms excluded by at least $n - k$ vertices. Therefore, relying on Corollary 6.10.4 it is enough to assume $3(n-k) > 2n$, i.e., $n > 3k$.

Let $a_1, a_2 \in V$ be the vertices with the smallest in-degrees. By assumption, $\textsc{in-deg}(a_1) \geq 1, \textsc{in-deg}(a_2) \geq 1$, and by Claim 6.10.11 we have:

Claim 6.10.12

Every $d \in V_v - \{a_1, a_2\}$ satisfies $\textsc{in-deg}(d) \geq k$.

We construct a cycle π in $\mathcal{G}_v$ that satisfies the following condition:

($\circ$) its first vertex d, as well as vertices d not contained in π, satisfy $\textsc{in-deg}(d) \geq k$.

Due to (1), it suffices to consider the following cases:

Case 1. $|\textsc{in}(a_1) \cup \textsc{in}(a_2)| \geq 2$

Relying on (1), choose in $V_v - \{a_1, a_2\}$ two distinct atoms $d \neq d'$ with $d \in \textsc{in}(a_1)$ and $d' \in \textsc{in}(a_2)$. Due to (2) the atoms can be chosen so that $d \neq a_2$ or $d' \neq a_1$. By symmetry we assume w.l.o.g. that $d \neq a_2$. If $d' = a_1$ we take the following simple path π in $\mathcal{G}_v$ satisfying ($\circ$):

$$d \longrightarrow a_1 \longrightarrow a_2$$

Otherwise, suppose $d' \neq a_1$ either. By Claim 6.10.12, $\textsc{in-deg}(d) \geq k$ and $\textsc{in-deg}(d') \geq k$. Choose in $V - \{d, a_1, d', a_2\}$ any atom e with $e \in \textsc{in}(d') \cap \textsc{out}(a_1)$ (since $\textsc{in-deg}(d') \geq k$, such e exists as a_1 excludes at most two atoms, as long as $k \geq 7$). This yields the following simple path π in $\mathcal{G}_v$ satisfying ($\circ$):

$$d \longrightarrow a_1 \longrightarrow e \longrightarrow d' \longrightarrow a_2$$

Case 2. $\textsc{in}(a_1) = \textsc{in}(a_2) = \{d\}$ **for some** $d \in V_v - \{a_1, a_2\}$

Take some two letters α_1, α_2 appearing in v such that $a_1 \notin \textsc{trg}(\alpha_1)$ and $a_2 \notin \textsc{trg}(\alpha_2)$. Due to (3) we can assume that either $\alpha_1 \neq \alpha_2$, or $\alpha_1 = \alpha_2$ but $v(\alpha_1) \geq 2$ (their cardinality in v is at

least 2). Note that $\textsc{src}(\alpha_1) = \textsc{src}(\alpha_2) = d$, and by Claim 6.10.12, $\textsc{in-deg}(d) \geq k$. Choose in $V - \{d, a_1, a_2\}$ any atom e with $e \in \textsc{in}(d) \cap \textsc{out}(a_1)$ (similarly as before, such e exists as long as $k \geq 6$). This yields the non-simple path π in $\mathcal{G}_v$ satisfying ($\circ$):

$$d \longrightarrow a_1 \longrightarrow e \longrightarrow d \longrightarrow a_2$$

We have thus constructed a path π from d to a_2. If $a_2 \notin \textsc{in}(d)$, append at the end of π any vertex c such that $c \in \textsc{out}(a_2) \cap \textsc{in}(d)$. As before, such a vertex exists since a_2 excludes at most 2 atoms and $\textsc{in-deg}(d) \geq k$, as long as $k \geq 8$. Therefore the last vertex c of π satisfies $c \in \textsc{in}(d)$, which means that π is a cycle as required.

In Case 1 we transform π, using Lemma 6.10.5, into an anti-cycle $\bar{w}$. In Case 2 we proceed similarly, except that the vertex d appears twice in π; this exception is treated by splitting all d-sourced letters into two disjoint blocks (cf. the proof of Lemma 6.10.5), containing α_1 and α_2, respectively.

We now remove, intuitively speaking, the anti-cycle $\bar{w}$ from v thus obtaining a smaller data vector v' to which we apply Theorem 6.10.6 and Lemma 6.10.5. We remove from v all letters appearing in $\bar{w}$, and add a single letter $\beta = \langle \textsc{src}(\bar{w}), \textsc{trg}(\bar{w}) \rangle \in \Sigma$. This yields a data vector v'. As the length of π is at most 6, the in-degree of a node e in the graph $\mathcal{G}_{v'}$ may be smaller by at most 6 than in the graph $\mathcal{G}_v$. Thus $\textsc{in-deg}(e) \geq 3$ in $\mathcal{G}_{v'}$ as $k \geq 9$. Moreover $\textsc{out-deg}(e) \geq n' - 3$ in $\mathcal{G}_{v'}$, where n' is the number of nodes of $\mathcal{G}_{v'}$, by Corollary 6.10.4. Therefore the graph $\mathcal{G}_{v'}$, assuming n to be sufficiently large, satisfies assumptions of Lemma 6.10.7, by which $\mathcal{G}_{v'}$ is strongly connected. In consequence, $\mathcal{G}_{v'}$ satisfies assumptions of Theorem 6.10.6, by which we derive a Hamiltonian cycle $\mathcal{C}$ in $\mathcal{G}$. The Hamiltonian cycle is turned, using Lemma 6.10.5, into an anti-cycle in w' with $\textsc{Par}(w') = v'$. Finally, replacing the letter β in w' by $\bar{w}$, yields an anti-cycle w with $\textsc{Par}(w) = v$, as required.

Lemma 6.10.10 is thus proved. □

Let $\mathbf{N}$ denote the set of all non-degenerate data vectors, and $\mathbf{N}^{\geq n} = \{v \in \mathbf{N} : \textsc{ord}(v) \geq n\}$. In these terms, Lemma 6.10.10 claims $\mathbf{N}^{\geq n} = \textsc{Par}(\mathbf{D}^{\geq n})$, for sufficiently large n.

Claim 6.10.13

For sufficiently large n, $\mathbf{N}^{\geq n} = \textsc{Par}(\mathbf{D}^{\geq n})$.

Therefore we will complete the proof of rationality of anti-cycles of bounded order (Lemma 6.9.3) once we prove the following:

Lemma 6.10.14: (Rationality of non-degenerate data vectors)

$\mathbf{N}^{\geq n}$ is rational, for sufficiently large $n \in \mathbb{N}$.

Proof. Fix $n \geq 6$. We define *the kernel* of a data vector $v : \Sigma \to \mathbb{N}$ as the intersection of all targets in v:

$$\textsc{ker}(v) \;=\; \bigcap_{\alpha \in \textsc{dom}(v)} \textsc{trg}(\alpha).$$

The size of the kernel is $0, 1$ or 2. For $X \subseteq \mathbb{A}$ of size at most 2, let

$$\mathbf{N}^{X,\geq n} \;=\; \{v \in \mathbf{N}^{\geq n} \,:\, \textsc{ker}(v) = X\}.$$

As $\mathbf{N}^{\geq n} = \bigcup_X \mathbf{N}^{X,\geq n}$, it is enough to show that the sets $\mathbf{N}^{X,\geq n}$ are rational. This, in turn, is implied by the following decomposition property of sets $\mathbf{N}^{X,\geq n}$:

$$\mathbf{N}^{X,\geq n} \;=\; \mathbf{N}^{X,n} \;+\; \textsc{Par}({\Sigma_X}^*), \tag{6.8}$$

where $\mathbf{N}^{X,n} = \{v \in \mathbf{N} \,:\, \textsc{ord}(v) = n \;\wedge\; \textsc{ker}(v) = X\}$ and $\Sigma_X = \{\alpha \in \Sigma \,:\, X \subseteq \textsc{trg}(\alpha)\}$, as the two sets are rational for every n and X.

Now we prove the decomposition (6.8), and later we argue that the sets appearing on the right-hand side are rational.

Towards showing the decomposition, we prove that kernel-preserving extensions by one letter $\alpha \in \Sigma$ preserve membership in $\mathbf{N}$:

$$v \in \mathbf{N},\; \textsc{ker}(v) = \textsc{ker}(v + \alpha) \implies v + \alpha \in \mathbf{N}; \tag{6.9}$$

and also that there always exists a letter α that one can remove from a vector in $\mathbf{N}^{\geq n+1}$, preserving kernel and membership in $\mathbf{N}$:

$$v \in \mathbf{N}^{\geq n+1} \implies \exists \alpha \in \textsc{dom}(v) \,:\, \textsc{ker}(v) = \textsc{ker}(v - \alpha),\; v - \alpha \in \mathbf{N}.$$

Concerning the first property, suppose $v \in \mathbf{N}$ and $\textsc{ker}(v) = \textsc{ker}(v + \alpha)$. We thus know that v satisfies conditions (1)–(3) and that $d = \textsc{src}(\alpha) \notin \textsc{ker}(v)$ since $d \notin \textsc{trg}(\alpha)$. This implies that $v + \alpha$ satisfies (1). For conditions (2)–(3) we consider two separate cases. If $d \in V_v$ then adding α may only increase in-neighbour sets $\textsc{in}(\_)$ and preceeding-letter sets $\textsc{pre}(\_)$, and hence $v + \alpha$ satisfies (2)–(3). Otherwise, suppose $d \notin V_v$ is a fresh source. We reason by contradiction. If $v + \alpha$ violates (2) for d and some $e \in V_v$, then v necessarily violates (1) due to $\textsc{in}(e) = \emptyset$. If $v + \alpha$ violates (3) for d and some $e \in V_v$, then all $\beta \in \textsc{dom}(v)$, except for exactly one, satisfy $\textsc{trg}(\beta) = \{d, e\}$ and hence forcedly $\textsc{src}(\beta) \neq e$. Therefore there is exactly one e-sourced letter in v and $\textsc{in}(e) = \emptyset$, and hence v violates (1) again.

We now concentrate on the second property. Removal of a letter from v may only increase (inclusion-wise) the kernel, say from X to X', but this only happens if $v(\alpha) = 1$, $X' \not\subseteq \textsc{trg}(\alpha)$, and $X' \subseteq \textsc{trg}(\beta)$ for all $\beta \in \textsc{dom}(v) - \{\alpha\}$. By inspection of possible sizes 1, 2 of X', one deduces that v may contain at most two such kernel-increasing letters. This eliminates at most 2 potential sources $\textsc{src}(\alpha)$.

Non-degeneracy can be only violated by vertices in the source graph of in-degree below 2.

Therefore non-degeneracy of $v - \alpha$ is guaranteed if removal of α does not decrease in-degree of any vertex below 2, i.e., $\textsc{src}(\alpha)$ does not belong to $\textsc{in}(d)$ for $d \in V_v$ of in-degree $\textsc{in-deg}(d) \leq 2$. For sufficiently large n, similarly as in Claim 6.10.12, there are at most 2 such vertices d in V_v. This eliminates at most 4 potential sources $\textsc{src}(\alpha)$.

In total, at most 6 potential sources $\textsc{src}(\alpha)$ are eliminated. Therefore, as long as $\textsc{ord}(v) > 6$, there is $\alpha \in \textsc{dom}(v)$ such that $\textsc{ker}(v) = \textsc{ker}(v - \alpha)$ and $v - \alpha \in \mathbf{N}$.

The decomposition (6.8) is thus proved.

We now argue that the sets appearing on the right-hand side of (6.8) are rational indeed. Fix $X \subseteq \mathbb{A}$ of size at most 2. The set Σ_X is orbit-finite and hence the language ${\Sigma_X}^*$ as well as its Parikh image are necessarily rational. For showing rationality of $\mathbf{N}^{X,n}$ we define

$$\mathbf{N}^{X,\{a_1,\ldots,a_n\}} = \{v \in \mathbf{N}^{X,n} \,:\, V_v = \{a_1, \ldots, a_n\}\}$$

(non-degenerate data vectors where sources are exactly n atoms $\{a_1, \ldots, a_n\}$, and all targets include X) and observe that:

$$\mathbf{N}^{X,n} = \bigcup_{a_1 \ldots a_n \in (\mathbb{A}-X)^{(n)}} \mathbf{N}^{X,\{a_1,\ldots,a_n\}}.$$

It is thus sufficient to show rationality of $\mathbf{N}^{X,\{a_1,\ldots,a_n\}}$ for a fixed set $\{a_1, \ldots, a_n\} \subseteq \mathbb{A}$. Let $\Sigma_X^{\{a_1,\ldots,a_n\}} = \{\alpha \in \Sigma_X \,:\, \textsc{src}(\alpha) \in \{a_1, \ldots, a_n\}\}$. For a set of data vectors S, let $\min(S) \subseteq S$ be the set of vectors that are minimal with respect to the point wise multiset ordering (see Definition 3.1.4). We observe that

$$\mathbf{N}^{X,\{a_1,\ldots,a_n\}} \;=\; \min\left(\mathbf{N}^{X,\{a_1,\ldots,a_n\}}\right) \;+\; \textsc{Par}({\Sigma_X^{\{a_1,\ldots,a_n\}}}^*),$$

i.e., each vector in $\mathbf{N}^{X,\{a_1,\ldots,a_n\}}$ is obtained from a minimal such vector by adding arbitrary letters from $\Sigma_X^{\{a_1,\ldots,a_n\}}$ – this follows by (6.9), since adding letters from $\Sigma_X^{\{a_1,\ldots,a_n\}}$ is kernel-preserving. As $\Sigma_X^{\{a_1,\ldots,a_n\}}$ is orbit-finite, it is enough to argue that the set $\min\left(\mathbf{N}^{X,\{a_1,\ldots,a_n\}}\right)$ is rational. Towards this claim, we calculate a rough upper bound on the size of data vectors belonging to this set. By inspecting the definition of non-degenerate data vectors we observe that the condition (1) requires a witnessing letter for each of n sources $a_1, \ldots, a_n$, condition (2) requires a witnessing letter for each pair or sources, and condition (3) requires two witnessing letters for each pair or sources. In consequence, the size of data vectors in $\min\left(\mathbf{N}^{X,\{a_1,\ldots,a_n\}}\right)$ is bounded by $3n^2 + n$, and therefore the set $\min\left(\mathbf{N}^{X,\{a_1,\ldots,a_n\}}\right)$ is orbit-finite (as n is fixed). The set is thus rational.

This completes the proof of Lemma 6.10.14. □

Claim 6.10.13 together with Lemma 6.10.14 imply Lemma 6.9.3.

Summary

In this chapter, we studied Parikh images of 1-NRA and showed that they are rational sets. Our techniques heavily exploit the assumption that there is only one register, and we don't know if these techniques can be lifted to automata with more registers. On the positive side, we are able to use the above-shown rationality of Parikh images of 1-NRA languages to study Parikh images of one-register context-free languages, and a also languages of a larger non-trivial class of non-deterministic register automata. These results are discussed in the next two chapters.

Chapter 7

Parikh images of hierarchical register automata

Objective

We showed that Parikh images of 1-NRA are rational, but we still do not know if all NRA have rational Parikh images. In this chapter, we slightly generalize the results of Chapter 6 on 1-NRA to a nontrivially larger subclass of NRA which we call *hierarchical register automata*. This model also provides another potential approach to solving the general question, whether all NRA have rational Parikh images(as opposed to generalising techniques from Chapter 4).

Contents

1 Summary of results

In this chapter, we identify a syntactic fragment of NRA which we call *hierarchical register automata* (HRA). Initially, we show it is more expressive than 1-NRA. The main result of this chapter is the following:

Theorem 7.1.1: (Rationality of HRA)

Parikh images of HRA are rational.

Recall that two languages are Parikh-equivalent if they have same Parikh images. As the converse of Theorem 7.1.1 easily holds, namely every rational set of data vectors is Parikh image of some language of a HRA (which is an immediate consequence of Theorem 7.5.1), by the above result we can deduce that the language of a non-deterministic register automaton has rational Parikh image if, and only if the automaton is Parikh-equivalent to some hierarchical register automata.

This chapter is structured in the following way. In the first section, we introduce the model of hierarchical register automaton and provide some examples. In the second section, we show that the class of languages recognized by this model is strictly contained in the NRA languages. The remaining sections are dedicated to proving Theorem 7.1.1.

2 Hierarchical register automata

We define a syntactical subclass of NRA by restricting transition constraints. The idea is to update registers in a hierarchical manner: if a transition rule does not preserve ith register, pre- and post-values of every larger register (jth register, for $j > i$) are unspecified.

Definition 7.2.1: (Hierarchical register automata)

Formally, a HRA is a NRA where each transition constraint φ has the following form:

$$\varphi \quad \equiv \quad \psi(x_1, x_2, \ldots, x_i, y, x'_i) \ \wedge \bigwedge_{1 \leq j < i} x_j = x'_j, \tag{7.1}$$

for some $i \in \{1, \ldots, k\}$.

The sub-formula ψ describes how the post-value of ith register (x'_i) depends on the relation between the input atom (y) and the pre-values of ith register and smaller ones ($x_1, x_2, \ldots, x_i$). Note that all smaller registers are preserved, and larger ones are not mentioned in φ (and hence their pre- and post-values are unspecified, which means that any pre- and post-values are

allowed). Note also that the constraint φ allows for updating ith register (according to the sub-constraint ψ) as well as every larger register (arbitrarily); the former we call *specified* update, and the latter one we call *unspecified* one. The number i we call the *level* of the transition constraint, or of the transition (rule) it appears in. As extreme examples, the following all-registers-preserving constraint

$$\bigwedge_{1\leq j\leq k} x_j = x'_j \neq y, \tag{7.2}$$

as well as the most liberal constraint **true** satisfied by any pre- and post-values of registers and any input atom, both are in the syntactic form (7.1), at level k and 1, respectively.

Example 7.2.2. Let $\Sigma = \mathbb{A}$. Consider the following 2-HRA, with control locations $Q = \{q_1, q_2, q_3\}$, and the single initial and accepting one $I = F = \{q_3\}$. The automaton has the following three transition rules:

$$\begin{aligned}
&(q_3(x_1, x_2),\ y,\ x_1 = x'_1 \neq y \wedge x_2 = x'_2 \neq y,\ q_2(x'_1, x'_2)),\\
&(q_2(x_1, x_2),\ y,\ x_1 = x'_1 \wedge x_2 = x'_2 = y,\ q_1(x'_1, x'_2)),\\
&(q_1(x_1, x_2),\ y,\ x_1 = y,\ q_3(x'_1, x'_2)).
\end{aligned}$$

the first two at level 2 and the last one at level 1. Indeed, the first two transition rules preserve values of both registers; the first one checks that the input is fresh, while the other one checks that that input equals to the second register. The last transition rule requires that the input equals to the first register, and updates both registers arbitrarily. The following figure shows the automaton described above:

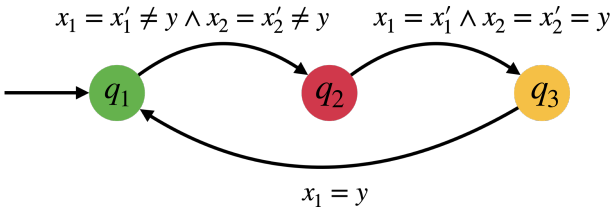


Figure 7.1: A 2-HRA: consecutive blocks of 3 different atoms.

The language accepted by the automaton can be easily described by a rational expression as follows:

$$L' = \left(\bigcup_{a,b,c\in\mathbb{A},\ a\neq b\neq c\neq a} abc \right)^* .$$

Definition 7.2.3: (Runs with fixed register values)

For $1 \leq i \leq k$ and a tuple of atoms $\mathbf{r} \in \mathbb{A}^{(i)}$, we may define a refined semantics of a k-HRA, $\mathcal{A}$ as the language of words accepted by a run where the values of the first (smallest) i registers

are continuously $\mathbf{r}$ and hence never change. We denote such a language by $L_{\mathbf{r}}(\mathcal{A})$.

3 Normal form

As in the previous chapter, we may assume that the constraints in all transition rules of k-HRA are in normal form defined below. Let Φ_i denote the formula

$$\Phi_i \equiv \bigwedge_{1 \leq j < i} x_j = x'_j$$

that says that registers $x_1 \dots x_{i-1}$ are preserved by a transition.

Definition 7.3.1: (Normal form of k-HRA)

A k-HRA is in *normal form* if the constraint in every transition rule is in one of the following five forms, for some $1 \leq j \leq i \leq k$. Below, we write $x_1 \dots x_i \neq y$ instead of $x_1 \neq y \wedge \dots \wedge x_i \neq y$, and likewise $x_1 \dots x_i \neq x'_i$ instead of $x_1 \neq x'_i \wedge \dots \wedge x_i \neq x'_i$.

(1) $\Phi_i \ \wedge \ x_j = y \ \wedge \ x_i = x'_i$,

(2) $\Phi_i \ \wedge \ x_1 \dots x_i \neq y \ \wedge \ x_i = x'_i$,

(3) $\Phi_i \ \wedge \ x_1 \dots x_i \neq y \ \wedge \ y = x'_i$,

(4) $\Phi_i \ \wedge \ x_j = y \ \wedge \ x_1 \dots x_i \neq x'_i$,

(5) $\Phi_i \ \wedge \ x_1 \dots x_i \neq y \ \wedge \ y \neq x'_i \ \wedge \ x_1 \dots x_i \neq x'_i$.

Similarly as before, the constraints (1)-(2) may be called *register-preserving* and the constraints (3)-(5) may be called *register-updating*. The level of constraints (1)-(5) is i.

The five forms of constraints defined above are like in case of 1-NRA, with the difference that registers $x_1 \dots x_{i-1}$ are additionally taken into account when testing the input atom y. All constraints require that registers $x_1 \dots x_{i-1}$ are preserved. The transition constraint (3) updates the register x_i by the input atom, while the transition constraints (4) and (5) update the register x_i by a fresh value different from the current values of registers $x_1 \dots x_i$ and the input atom. The constraints (1) and (4) require that the input atom y is equal to the content of some register x_j, for $j \leq i$, while the other constraints require that the atom y is fresh, i.e, not currently stored in registers $x_1, \dots, x_k$. Finally, all constraints leave completely unspecified contents of registers x_m for $m > i$.

Proposition 7.3.2: (Normal form of HRA)

Every HRA can be transformed to an equivalent HRA in normal form.

4 HRA are strictly between NRA and 1-NRA

We claim that the expressive power of HRA is strictly weaker than that of NRA, but strictly stronger than that of 1-NRA.

Let the alphabet be $\Sigma = \mathbb{A}$. Consider the 2-NRA with four states shown in Figure 7.2.

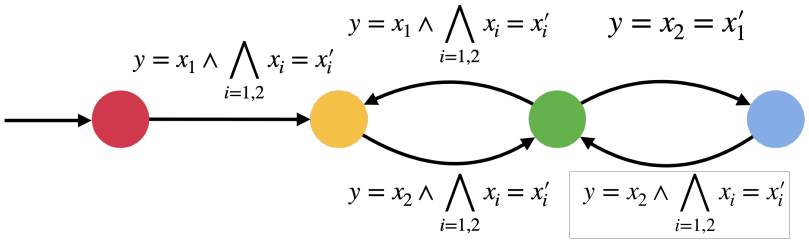


Figure 7.2: A language recognized by a 2-NRA.

Starting from the initial red location, the automaton repeatedly checks that the input letter is equal to the first and the second register, in an alternating manner. After each two successful checks, the automaton reaches the green location: the first two checks go from red to green location, each consecutive two checks execute the yellow-green loop. In addition, in the green location the automaton can, besides continuing the yellow-green loop, execute the green-blue loop which enforces repetition of the last seen atom as the next input letter, shifting of the value of the second register to the first one, a non-deterministic guessing of the new value of the second register, and then checking that this guessed value (which is forcedly different than the new value of the second register, but could be equal to the previous value of the first register) appears as the consecutive input letter. If we choose the green location as the only accepting one, the language L accepted by the automaton consists of all words of the following form:

$$(a_1 b_1)^+ (a_2 b_2)^+ \dots (a_n b_n)^+ \in \mathbb{A}^* \tag{7.3}$$

where $n \geq 1$, $a_i \neq b_i$ for every $i = 1 \dots n$, and $a_{i+1} = b_i$ for every $i = 1 \dots n-1$.

Theorem 7.4.1

The language L defined above is not recognised by any HRA.

Before proving the above fact, we build some preparatory tools that is used in the proof.

Definition 7.4.2: (Seperated sequence)

Consider a finite sequence $h = h_1 h_2 \dots h_l$ of positive integers. We say that h has order $k > 0$ if $h_i \in \{1, 2, \dots, k\}$ for every $i = 1, \dots, l$. We call the sequence *separated* if for every distinct indices $i < j$ satisfying $h_i = h_j$, there is an index m strictly in-between, $i < m < j$, such that $h_m \leq h_i$.

In particular, each two consecutive elements of a separated sequence are forcedly distinct. In the sequel we use the recursive sequence defined by:

$$f_1 = 1 \qquad f_{i+1} = 2f_i + 1.$$

Lemma 7.4.3

The length of a separated sequence of order k is at most f_k.

Proof. We prove this lemma by induction on the order k. For base case $k = 1$, the only separated sequence is 1, whose length is $f_1 = 1$.

For induction step, assume that the length of every separated sequence of order k is at most f_k, and consider an arbitrary separated sequence h of order $k+1$. If we remove all occurrences of $k+1$ from h, we get a sub sequence h' of order k which, by induction assumption, has length $l \leq f_k$. We observe that between every two appeartances of $k+1$ in h, there is at least one element from h'. Therefore the maximum possible number of occurrences of $k+1$ in h is $l+1$. In consequence, the length of h is at most $2l + 1 \leq 2f_k + 1 = f_{k+1}$, as required. □

Proof of Theorem 7.4.1. Suppose, towards contradiction, that a k-HRA $\mathcal{A}$ recognises L. Consider the following word $w \in L$:

$$w = (a_1 a_2)^3 (a_2 a_3)^3 \dots (a_n a_{n+1})^3$$

where $a_i \neq a_j$ for every $i \neq j$, and $n > f_k$. Consider also an arbitrary accepting run π of $\mathcal{A}$ over w.

For every a_i appearing in the word w, we define the *region* of a_i, denoted π_{a_i}, as the set of positions in the run π strictly between the first and the last appearance of a_i in w. We also think of a region as of a sequence of configurations that appear therein. The region of a_i, for $1 < i < n$, consists, as depicted below, of the blue-colored configurations $c_1, c_2, \dots, c_9$ that appear in the run between the two red-colored appearances of a_i:

$$\dots \xrightarrow{a_{i-1}} \xrightarrow{a_i} c_1 \xrightarrow{a_{i-1}} c_2 \xrightarrow{a_i} c_3 \xrightarrow{a_{i-1}} c_4 \xrightarrow{a_i} c_5 \xrightarrow{a_i} c_6 \xrightarrow{a_{i+1}} c_7 \xrightarrow{a_i} c_8 \xrightarrow{a_{i+1}} c_9 \xrightarrow{a_i} \xrightarrow{a_{i+1}} \dots$$

The regions of a_1 and a_n are smaller. For any two consecutive atoms a_i and a_{i+1}, their corresponding regions intersect.

Claim 7.4.4

Let $i \in \{1, \ldots, n\}$. In every configuration in the region π_{a_i}, the atom a_i is stored in some register. Moreover, a_i is stored in the same register in all these configurations.

Proof of Claim 7.4.4. When reading a_i for the first time, the automaton $\mathcal{A}$ has to store a_i in some register, and a_i must be in some register until the end of its region. Indeed, suppose a_i is not stored in any register at some position in the region π_{a_i}. The word w' obtained by replacing all the consecutive appearances of a_i by some fresh atom b is still accepted by $\mathcal{A}$ (using the run π where a_i is renamed to b from the considered position on), while $w' \notin L$ – a contradiction.

Let $h_i \in \{1, \ldots, k\}$ be the index of register in which a_i is stored by the run π after its first appearance. We now argue that a_i is never moved to other register inside the region π_{a_i}. If a_i was moved to a different register by a transition that reads input atom stored currently in some register (transition rule of type (4) in Definition 7.3.1), the new value of all registers updated by this transition are necessarily guessed, and we get a contradiction, similarly as above, by renaming a_i to some fresh b. Likewise, if a_i was moved to some other register by a transition that reads an input atom not currently stored in registers (transition rule of type (3) or (5) in Definition 7.3.1), i.e. a_i is either given as input (3) or guessed (5), we again obtain a contradiction by renaming a_i to some fresh atom b in π from this position on. as a subword, which is a contradiction. □

Inside the whole region π_{a_i}, atom a_i is thus stored in the unique h_i-th register (we also say: register *of index* h_i). We intuitively think of this register as *active*. As consecutive (and only consecutive) regions overlap, there is always (except the first and last configuration of π) one or two active registers.

Claim 7.4.5

Transitions inside the region π_{a_i} do not update register of index h_i or any lower one (do not update jth register, for $j \leq h_i$).

Proof of Claim 7.4.5. By Claim 7.4.4 the atom a_i is in h_i-th register in the configuration before and after the transition. If the level of the transition is $\geq h_i$, the claim is proved. Otherwise, suppose the level of the transition is $< h_i$. This means that the h_i-th register is updated unconditionally by the transition, which allows us to obtain a contradiction similarly as in the proof of Claim 7.4.4, by renaming a to a fresh atom in π from this transition on. □

We have defined a sequence $h = h_1 \dots h_n$ of order k, since $\mathcal{A}$ has k registers, and length n. We observe that for every $i < n$, $h_i \neq h_{i+1}$, as the two simultaneously active registers are necessarily different. We now state the crucial observation: $h_i = h_j$ is only possible if the sequence h falls below h_i on positions between i and j. Formally:

Claim 7.4.6

If $i < j$ and for all m strictly between i and j we have $h_m > h_i$, then $h_i \neq h_j$.

Proof of Claim 7.4.6. Indeed, by the assumptions, inside regions $\pi_{a_i} \dots \pi_{a_{j-1}}$ there is always some active register of index at least h_i. Therefore, by Claim 7.4.5 the register of index h_i is never updated inside regions $\pi_{a_i} \dots \pi_{a_{j-1}}$. Since the two consecutive regions $\pi_{a_{j-1}}$ and π_{a_j} overlap, in the starting configuration of region π_{a_j}, the register of index h_i still contains $a_i \neq a_j$ and hence $h_i \neq h_j$, as required. □

The above claim immediately implies:

Claim 7.4.7

If $i < j$ and $h_i = h_j$ then for some m strictly in between i and j we have $h_m \leq h_i$.

In other words, the sequence h is separated. By Lemma 7.4.3 its length $n \leq f_k$, which is in contradiction with our assumption that $n > f_k$. This completes the proof of Theorem 7.4.1. □

5 HRA recognize all rational languages

Theorem 7.5.1

Rational data languages are recognised by HRA.

Proof. We proceed by induction on rational expression of a rational language. For convenience we assume, w.l.o.g., that each orbit-finite sum is indexed by a subset of $I \subseteq \mathbb{A}^{(n)}$ of non-repeating n-tuples of atoms, for some $n \in \mathbb{N}$. Indeed, every orbit-finite union can be split into a finite union of single-orbit unions, and every single-orbit set J is the image of an equivariant function f from such a set I[1], $J = f(I)$, hence

$$\bigcup_{j \in J} L_j \quad = \quad \bigcup_{i \in I} L_{f(i)} \quad = \quad \bigcup_{i \in I} K_i$$

[1] cf. Bojańczyk, 2019, Sect. 3.2.

where $K_i = L_{f(i)}$. Under this simplifying assumption we prove, by induction on rational expression of a rational language, the following claim (for conciseness, we say that a *tuple* $\mathbf{s} \in \mathbb{A}^{(n)}$ supports x if the set of n atoms appearing in $\mathbf{s}$ does so):

Claim 7.5.2

For every rational language L over an alphabet of the form $\Sigma = H \times \mathbb{A}$, and every tuple $\mathbf{s}$ supporting its rational expression, there is a HRA $\mathcal{A}$ such that $L_{\mathbf{s}}(\mathcal{A}) = L$.

We emphasise that we consider supports of *rational expressions* of rational languages, defined as well-founded trees, instead of supports of languages themselves. Clearly, a tuple supporting a rational expression of a language also support the language itself.

The induction base, for $L = \{\varepsilon\}$ or $L = \{\sigma\}$ where $\sigma \in \Sigma$, is straightforward. The induction step splits into three cases.

Case 1: $L = L_1\, L_2$**.** Let $\mathbf{s}$ be a tuple of atoms supporting the rational expression of L, and hence also the rational expressions of L_1 and L_2. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the HRA which, due to the induction assumption, recognize $L_{\mathbf{s}}(\mathcal{A}_1) = L_1$ and $L_{\mathbf{s}}(\mathcal{A}_2) = L_2$. Let the automaton $\mathcal{A}$ initially run $\mathcal{A}_1$, and from each accepting location of $\mathcal{A}_1$ non-deterministically choose either to continue inside $\mathcal{A}_1$, or to run $\mathcal{A}_2$. We have $L_{\mathbf{s}}(\mathcal{A}) = L$, as required.

Case 2: $L = K^*$**.** This case is dealt with similarly to the previous one.

Case 3: $L = \bigcup_{i \in I} L_i$**.** Let $\mathbf{s}$ be a tuple of atoms supporting the rational expression of L, and hence also the set I and the mapping $i \mapsto L_i$. Thus the concatenated tuple $\mathbf{s}i$ supports L_i (recall that i is assumed for convenience to be a tuple of atoms). For an $\mathbf{s}$-orbit J in I, let

$$L_J \;=\; \bigcup_{j \in J} L_j \;\subseteq\; L.$$

Consider an arbitrary $\mathbf{s}$-orbit J in I (each orbit is treated separately). Fix an arbitrary element $i \in J$ and an automaton $\mathcal{B}$ such that, due to the induction assumption, recognizes $L_{\mathbf{s}i}(\mathcal{B}) = L_i$. Therefore, for every $j = \pi(i) \in J$, where π is an $\mathbf{s}$-automorphism, the same automaton $\mathcal{B}$ recognizes $L_{\mathbf{s}j}(\mathcal{B}) = L_j$. Let the automaton $\mathcal{A}_J$ initially guess $i \in J$ and put it into the smallest registers not occupied by $\mathbf{s}$, and then run $\mathcal{B}$. We have $L_{\mathbf{s}}(\mathcal{A}_J) = L_J$. The language L is the union of finitely many languages L_J, and hence L is recognized by a HRA that initially chooses an $\mathbf{s}$-orbit J in I and then runs $\mathcal{A}_J$.

□

6 Parikh images of HRA are rational

This section contains the proof of Theorem 7.1.1: Parikh images of languages of HRA are rational. The proof proceeds by induction on the number of registers.

Induction base. The induction base, i.e., rationality of Parikh images of 1-HRA languages, follows immediately by Theorem 6.1.1 proved in Chapter 6.

Altering paths. Before proceeding to the induction step we recall the concept of altering paths introduced in Chapter 6, but slightly adapted to the framework of HRA instead of 1-NRA only. Given a k-HRA $\mathcal{A} = \langle H, Q, I, F, \Delta \rangle$, we define the language $P_{\mathcal{A}}$ of *altering paths*, over the alphabet$(Q \times \mathbb{A} \times Q) \cup (H \times \mathbb{A})$ containing words of the form:

$$\langle q_1, a_1, p_1 \rangle \langle h_1, b_1 \rangle \langle q_2, a_2, p_2 \rangle \langle h_2, b_2 \rangle \ \ldots \ \langle q_{n-1}, a_{n-1}, p_{n-1} \rangle \langle h_{n-1}, b_{n-1} \rangle \langle q_n, a_n, p_n \rangle \tag{7.4}$$

$(n \geq 1)$ such that, for $i = 1, \ldots, n-1$, it holds $a_i \neq a_{i+1}$ and

$$p_i(a_i \mathbf{r}) \xrightarrow{\langle h_i, b_i \rangle} q_{i+1}(a_{i+1} \mathbf{r}') \tag{7.5}$$

is a transition of $\mathcal{A}$ at level 1 for some tuples $\mathbf{r}, \mathbf{r}' \in \mathbb{A}^{(k-1)}$, and such that $q_1 \in I$ and $p_n \in F$. The atoms a_i and a_{i+1} are here pre- and post-values of the first register, and $\mathbf{r}, \mathbf{r}'$ are pre- and post-values of the remaining $k-1$ registers. Intuitively, a letter $\langle q, a, p \rangle$ represents a run of $\mathcal{A}$ starting from a configuration $q(a\mathbf{r}')$ and ending in $p(a\mathbf{r})$, for some $\mathbf{r}, \mathbf{r}' \in \mathbb{A}^{(k-1)}$, such that the first register contains a and is preserved along the run until the automaton reaches the configuration $p(a\mathbf{r})$, from which the automaton finally updates the first register. Along this run other registers may be updated.

We observe that the altering path language of a k-HRA $\mathcal{A}$ is the same as the altering path language of a 1-HRA $\mathcal{A}'$ obtained from $\mathcal{A}$ by removing all registers except the first (smallest) one, and all transition rules of level greater than 1. Therefore, as an immediate corollary of the proof of Lemma 6.4.3 in Chapter 6 we get:

Claim 7.6.1

For every $k \geq 1$, the altering path language $P_{\mathcal{A}}$ of k-HRA $\mathcal{A}$ has rational Parikh image.

Induction step. We now proceed to the induction step. To this aim we fix $k > 1$ and assume that languages of HRA with less than k registers have rational Parikh images. We consider a fixed k-HRA $\mathcal{A} = \langle H, Q, I, F, \Delta \rangle$ and aim at showing that Parikh image of $L(A)$ is rational. W.l.o.g. we assume that $\mathcal{A}$ is in normal form (Proposition 7.3.2). Let $\Sigma = H \times \mathbb{A}$ denote the input alphabet.

We construct a k-HRA $\mathcal{A}_{qp}$ by removing from $\mathcal{A}$ all transition rules that update (i.e., do not preserve) the first register, and by taking q as the only initial location and p as the only accepting one. Intuitively speaking, the first register is *frozen* in $\mathcal{A}_{qp}$, in the sense that it is never updated and thus keeps its initial value a along the whole run. For $a \in \mathbb{A}$, we denote by

$$L_a(\mathcal{A}_{qp}) \ = \bigcup_{\mathbf{r}, \mathbf{s} \in \mathbb{A}^{(k-1)}} L_{q(a\mathbf{r})\, p(a\mathbf{s})}(\mathcal{A}_{qp}) \ \subseteq \ L(\mathcal{A}_{qp})$$

the subset of $L(\mathcal{A}_{qp})$ consisting of words accepted by $\mathcal{A}_{qp}$ by a run where the value of the first register is (continuously) a. We need to deduce from the induction assumption the following claim:

Claim 7.6.2

The languages $L_a(\mathcal{A}_{qp})$ have rational Parikh images.

Before proving the lemma we use it to complete the proof Theorem 7.1.1. Consider the language $K = P_{\mathcal{A}(S)}$ obtained by applying the following substitution S to the language $P_{\mathcal{A}}$:

$$\langle q, a, p\rangle \quad \mapsto \quad L_a(\mathcal{A}_{qp}) \qquad\qquad \langle h, b\rangle \quad \mapsto \quad \{\langle h, b\rangle\}.$$

In words, triples $\langle q, a, p\rangle$ are replaced by any word accepted by $\mathcal{A}_{qp}$ by a run where the value of the first register is continuously a, while pairs $\langle h, b\rangle$ are preserved.

Claim 7.6.3

$L(A) = K$.

We argue that both inclusions hold. The inclusion $L(A) \subseteq K$ is shown by factorising each accepting run of $\mathcal{A}$ by transitions that update the first register, of the form (7.5), so that each word $w \in L(\mathcal{A})$ factorizes into:

$$w \;=\; w_1 \langle h_1, b_1\rangle w_2 \langle h_2, b_2\rangle \ldots w_{n-1} \langle h_{n-1}, b_{n-1}\rangle w_n, \tag{7.6}$$

for $w_i \in L_{a_i}(\mathcal{A}_{q_i p_i})$ for some atom a_i and control locations q_i, p_i, and therefore $w \in K$. For the reverse inclusion $K \subseteq L(A)$ consider a word $w \in K$, necessarily of the form (7.6), due to an altering path as in (7.4) and accepting runs π_i of $\mathcal{A}_{q_i p_i}$ over words w_i, where the first register is continuously equal a_i along π_i. By concatenating these runs (considered as sequences of configurations) one gets an accepting run $\pi = \pi_1\pi_2 \ldots \pi_n$ of $\mathcal{A}$ over the word w, as required. The transitions (7.5) confirm that π is a run since $\mathcal{A}$ is hierarchical: all these transitions are all at level 1 and may perform (unspecified) updates of all other registers.

Having Claims 7.6.1, 7.6.2, and 7.6.3, one easily completes the proof of Theorem 7.1.1. Indeed, Parikh image of $K = P_{\mathcal{A}}(S)$ is rational due to Lemma 4.3.2, as Parikh images of $P_{\mathcal{A}}$ and all languages $L_a(\mathcal{A}_{qp})$ are so due to Claims 7.6.1 and 7.6.2, respectively, and therefore the same holds for $L(A)$, due to Claim 7.6.3.

Proof of Claim 7.6.2. For every $q, p \in Q$ we define a new $(k-1)$-HRA $\mathcal{A}'_{qp}$ that behaves exactly as $\mathcal{A}_{qp}$ except that the first register is removed. The removal of the register is compensated by an additional bit in the finite component of the alphabet of $\mathcal{A}'_{qp}$ that informs the automaton whether the input atom is equal to the (removed) first register or not.

Formally, the new automaton is $\mathcal{A}'_{qp} = \langle \{=, \neq\} \times H, Q, \{q\}, \{p\}, \Delta' \rangle$, where the transition rules Δ' are defined as follows. Due to the assumption that $\mathcal{A}$ is in normal form (and hence so are all automata $\mathcal{A}_{qp}$), its every transition constraint (7.1) at level i, say, either entails the equality $y = x_1$, or the inequality $y \neq x_1$. The transition rules Δ' are obtained from the transition rules of $\mathcal{A}_{qp}$ (i.e., from transition rules of $\mathcal{A}$ at level greater than 1) by transforming each transition rule

$$(q(x_1, x_2 \dots x_k), \langle h, y \rangle, \varphi, q'(x'_1, x'_2 \dots x'_k))$$

of $\mathcal{A}_{qp}$ to the following one:

$$(q(x_1, x_2 \dots x_k), \langle (\sim, h), y \rangle, \varphi', q'(x'_1, x'_2 \dots x'_k))$$

where $\sim \in \{=, \neq\}$ is chosen so that φ entails $y \sim x_1$, and φ' is obtained from φ by removing all (in)equalities referring to the first register.

By induction assumption we know that Parikh image of $\mathcal{A}'_{qp}$ is rational, for every $q, p \in Q$. For $a \in \mathbb{A}$, consider the following sub-alphabet (that fixes, intuitively, the value of the first register to be a):

$$\Sigma_a \;=\; \{\langle (=, h), a \rangle \;:\; h \in H\} \;\cup\; \{\langle (\neq, h), b \rangle \;:\; h \in H,\; b \in \mathbb{A} - \{a\}\} \;\subseteq\; \Sigma,$$

and define the languages L_{qap} as the restriction of $L(\mathcal{A}'_{qp})$ to the sub-alphabet Σ_a:

$$L_{qap} \;:=\; L(\mathcal{A}'_{qp}) \cap (\Sigma_a)^*.$$

By Lemma 4.3.3 from chapter 4 we have:

Claim 7.6.4

Parikh images of the languages L_{qap} are rational.

Finally, we observe that $L_a(\mathcal{A}_{qp})$ is obtained from L_{qap} by applying the substitution (actually, the projection):

$$\langle (\sim, h), b \rangle \quad \mapsto \quad \{\langle h, b \rangle\}$$

and therefore also has rational Parikh image, as required. This completes the proof of Lemma 7.6.2, and hence also the proof of Theorem 7.1.1. □

As a consequence of Theorems 7.1.1 and 7.5.1, we get the following:

Corollary 7.6.5: (Parikh equivalence of HRA and rational languages)

Parikh images of HRA languages are exactlty rational sets of data vectors.

The above result also provides another potential approach to the quest of rationality of Parikh images of general NRA:

Corollary 7.6.6: (Rationality of NRA/CFG)

The language of a NRA (CFG) has rational Parikh image if, and only if, it is Parikh-equivalent to some HRA.

Summary

In this chapter, we introduced a syntactic subclass of register automata, called hierarchical register automata. We showed that they are strictly more expressive than one-register automata, strictly less expressive than unrestricted non-deterministic register automata, recognize all rational languages, and their languages are Parikh-equivalent to rational languages. In consequence, the language of a non-deterministic register automaton has rational Parikh image if, and only if the language is Parikh-equivalent to the language of some hierarchical register automaton. It remains open if this characterisation may be helpful for proving rationality of Parikh images of languages of unrestricted non-deterministic register automata.

Chapter 8

Parikh images of one-register context-free languages

Objective

In the previous chapters we showed that Parikh images of 1-NRA languages are rational, and then slightly generalized this result to HRA. In this chapter we investigate further into one-register context-free grammars, and extend the results of Chapter 6 to this model.

Contents

1 Summary of results

The main objective of this section is to prove following result:

Theorem 8.1.1: (Rationality)

Parikh images of one-register context-free languages are rational.

As a corollary, we deduce that the language of every 1-CFG is Parikh-equivalent to the language of an NRA. The number of registers of the NRA may be larger than one.

We proceed in four steps. In Section 2, by a Ramsey type argument, we prove that a sufficiently large set of productions contains a *compatible* pair (Lemma 8.2.3). Then in Section 3, we define *traversals* and *side-effects* which will be useful in further sections. In Section 4, we define *width* of derivation trees. Then, in Section 5 we show that for a sufficiently large constant $n \in \mathbb{N}$, every derivation tree can be transformed into a tree of width at most n while preserving the Parikh image of its yield (Lemma 8.5.2). The cut-and-paste transformation relies on compatibility of productions in a tree. Finally, also in Section 5, we argue that Parikh image of the set of words generated by derivation trees of width bounded by n is rational, for every fixed $n \in \mathbb{N}$ (Lemma 8.5.7). Lemmas 8.5.2 and 8.5.7 imply Theorem 8.1.1.

2 Compatibility

The equality type of a tuple $\langle a_1, \ldots, a_k \rangle \in \mathbb{A}^k$ is defined as the set $\{\langle i, j \rangle : 1 \leq i < j \leq k,\ a_i = a_j\}$. Intuitively speaking, tuples of the same equality type admit the same equalities between their coordinates.

Definition 8.2.1: (Compatible tuples)

Two tuples $\alpha = \langle a_1, \ldots, a_k \rangle$ and $\beta = \langle b_1, \ldots, b_k \rangle$ we call *compatible* if they have the same equality type, and for every coordinate $i \in \{1, \ldots, k\}$ one of two conditions holds: either (1) $a_i = b_i$; or (2) $a_i \neq b_i$ and both a_i and b_i do not appear in the other tuple: $a_i \notin \{b_1, \ldots, b_k\}$, $b_i \notin \{a_1, \ldots, a_k\}$.

In particular, two equal k-tuples are always compatible, but also two disjoint tuples are always compatible.

Let us see few examples to get acquainted with the definition of compatibility.

Example 8.2.2. Consider the two 4-tuples in Figure 8.1. In the first component of both vectors we have $a \neq c$, and a is not present in the second vector and c is not present in the first vector. This condition holds also for both third and fourth component. Both vectors have d in the second component. This fulfills the conditions of compatibility. Therefore these two tuples are compatible.

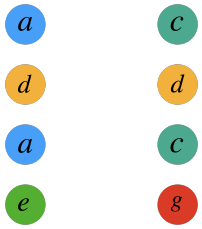


Figure 8.1: Two compatible tuples.

As another example, consider the two 4-tuples in Figure 8.2. These two tuples are of the same equality type, but are not compatible. The first component of both vectors are not the same, but the atom c is equal to fourth component of the first vector, which violates the defining condition of compatibility.

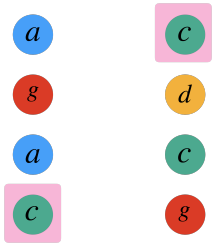


Figure 8.2: Two non-compatible tuples.

Lemma 8.2.3: (Compatible pairs)

For every $k \in \mathbb{N}$ there is some $l = f(k) \in \mathbb{N}$ such that every finite multiset of k-tuples of atoms $A : \mathbb{A}^k \to \mathbb{N}$ of size at least l contains two compatible k-tuples.

Proof. Let $k \in \mathbb{N}$ be fixed. If A contains two equal tuples, they are compatible. Thus we can assume A to be a set. We take $l = f(k)$ large enough to satisfy the constraint (8.1) below.

The number of different equality types E_k is finite and equal to the number of partitions of the coordinates set $\{1, \ldots, k\}$ (the kth Bell number). By the pigeonhole principle, for $l = |A|$ large enough, there is a subset $A' \subseteq A$ of size $l' = |A'| = \frac{l}{E_k}$ whose elements have all the same equality type.

We now consider an undirected clique of size l' with vertices A', where the edge between vertices $\alpha = \langle a_1, \ldots, a_k \rangle$ and $\beta = \langle b_1, \ldots, b_k \rangle$ is labeled (coloured) by the set $D_{\alpha\beta} = \{i \in \{1, \ldots, k\} : a_i \neq b_i\}$. Intuitively, the colour describes the coordinates on which α and β disagree. The number of colours is at most 2^k. By Ramsey's theorem, for l' large enough the

graph contains a monochromatic clique A" of size $l" = k^2 + 1$; indeed, it suffices to take

$$l' \geq R(\underbrace{l", l", \ldots, l"}_{2^k}). \tag{8.1}$$

Thus every two elements of A" disagree on the same coordinates $D \subseteq \{1, \ldots, k\}$, and hence also agree on the same coordinates $\{1, \ldots, k\} - D$.

Take any $\alpha = \langle a_1, \ldots, a_k \rangle \in A$". For every coordinate $i \in D$, all tuples $\beta \in A$" are pairwise different on that coordinate. Therefore, at most k tuples $\beta = \langle b_1, \ldots, b_k \rangle \in A$" may satisfy

$$b_i \in \{a_1, \ldots, a_k\}, \tag{8.2}$$

i.e., b_i appears in α. As $|D| \leq k$, at most k^2 tuples (including α itself) may satisfy the condition (8.2) for some coordinate $i \in D$. Therefore taking any of the remaining tuples, say β, we obtain a compatible pair α, β. □

3 Traversals and side-effects

Recall the definition of 1-CFG in Chapter 3. Its configurations are of the form $q(a) \in Q \times \mathbb{A}$, and its input alphabet is $\Sigma = H \times \mathbb{A}$.

Like in Proposition 6.2.2 in Chapter 6, we assume in the sequel that each production rule of 1-CFG defines one orbit in $\mathbb{A}^3$. We naturally adopt the definition of normal form of 1-NRA (cf. Definition 6.2.1) to 1-CFG:

Proposition 8.3.1: (Normal form of 1-CFG)

Similarly as in the case of 1-NRA, we may assume that every constraint in production rules of 1-CFG describes one orbit in $\mathbb{A}^3$, i.e., is one of φ_1 - φ_5 from Definition 6.2.1.

Definition 8.3.2: (Arity, tree order and paths)

The number of children of a node x in a derivation tree $\mathcal{T}$ we call *arity* of x (leaves are nodes of arity 0, other nodes have arity 2). Let $\preceq$ denote the partial tree order ($x \preceq y$ if x is an ancestor of y). A path from a node x to a node y, assuming $x \preceq y$, is the set $\{z \in \mathcal{T} : x \preceq z \preceq y\}$ of all nodes z appearing between the nodes x and y, including x and y.

Consider an arbitrary derivation tree $\mathcal{T}$ of $\mathcal{G}$, and recall from Chapter 3 that Π_2 denotes the set of productions derived from Δ_2 which have arity 2. We distinguish two ways of traversing a

production $\bullet(a) \to \bullet(b)\ \bullet(b') \in \Pi_2$ appearing in $\mathcal{T}$ by a path, namely left and right traversal as shown in Figure 8.3:

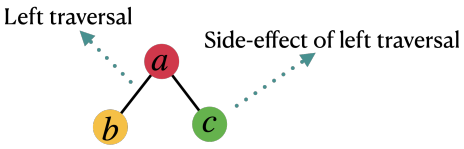
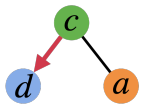


Figure 8.3: Left and right traversal.

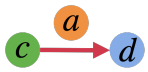
Definition 8.3.3: (Side-effects)

Once left or right traversal is chosen, say the right one, a production $q(a) \to p(b)\ p'(b') \in \Pi_2$ resembles a transition of 1-NRA (over the extended input alphabet $\Gamma = (Q \cup H) \times \mathbb{A}$) from $q(a)$ to $p'(b')$ which inputs the label of the remaining node, namely $p(b)$. We call the pair $p(b) \in \Gamma$ the *side-effect* of the right traversal; symmetrically we call $p'(b')$ the side-effect of the left traversal.

For example, the following left-traversal:

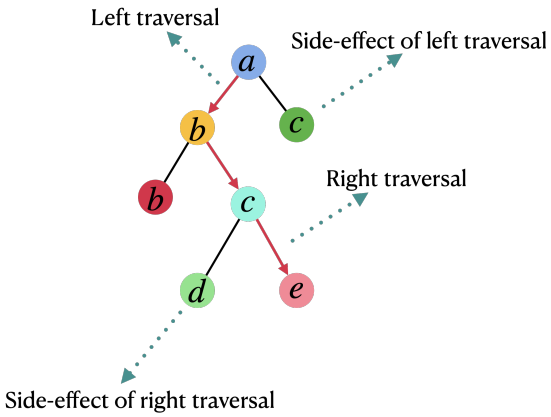


can be interpreted as the transition of an 1-NRA:



For two configurations $q(a)$ and $p(b)$ of $\mathcal{G}$, we denote by $S_{q(a)\,p(b)} \subseteq \Gamma^*$ the set of all sequences of side-effects that may appear along a path from a node labeled by $q(a)$ to a node labeled by $p(b)$ in a derivation tree of $\mathcal{G}$.

Example 8.3.4. The figure below shows side-effects of left and right traversals:



The path in the above derivation tree

has the following side-effect sequence:

As a corollary of Lemma 6.1.2 we get:

Lemma 8.3.5: (Rationality of side-effects)

Languages $S_{q(a)\,p(b)}$ have rational Parikh images.

Proof. Indeed, the claim follows immediately by Lemma 6.1.2, if production traversals are considered as transitions of a 1-NRA over the input alphabet $\Gamma = (Q \cup H) \times \mathbb{A}$, and the side-effect of a traversal is considered as input of a transition. □

4 Height, width, and rank

In this section we provide some technical definitions regarding derivation trees that will be useful in the next section.

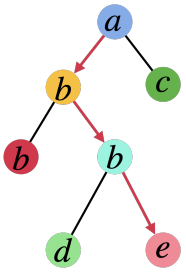
Definition 8.4.1: (Traversals)

Similarly as for 1-NRA, the right traversal of a production $q(a) \to p(b)\ p'(b') \in \Pi_2$ is called *register-preserving* if $a = b'$, and *register-updating* if $a \neq b'$; likewise for the left traversal.

Definition 8.4.2: (Length of a path)

We define the *length* of a path in a derivation tree $\mathcal{T}$ as the number of register-updating production traversals along the path, and the *height* of a node x in $\mathcal{T}$ as the maximal length of a path from x to a leaf. The height of a tree is the height of its root.

Example 8.4.3. In the following derivation tree, consider the path from root to the node labelled by atom e:



The register is updated twice along this path, once from a to b and once from b to e, therefore the length of this path is 2. Consider the yellow node labelled by atom b. There are three paths starting in this node and ending in some leaf. Each of the paths has length 1, therefore the height of the node is 1. The height of the tree is 2.

Definition 8.4.4: (Cut)

A *cut* in $\mathcal{T}$ is a set of nodes which are pairwise incomparable with respect to the tree ordering. A cut is called *n-cut* if its size is at least n and the height of every node in the cut is at least n.

Definition 8.4.5: (Width)

The *width* of a derivation tree $\mathcal{T}$ is the maximal n for which $\mathcal{T}$ contains some n-cut.

Definition 8.4.6: (Rank of a derivation tree)

The *rank* of a derivation tree is defined as the multiset of lengths of all paths from the root to some leaf.

Example 8.4.7. Consider the derivation tree in Figure 8.4 below. As every other derivation tree, it admits a 0-cut consisting of all leaves (highlighted by grey boxes). The cut $\{d, e\}$ is a 1-cut, and the width of the tree is 1, as it admits no 2-cuts. The rank of the tree is $\{1, 2, 3, 3, 3, 1\}$, the 6-element multiset containing lengths of all the 6 paths from root to leaves.

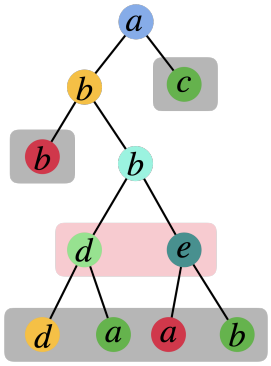


Figure 8.4: Cuts in a derivation tree.

For a finite multiset $r : \mathbb{N} \to \mathbb{N}$ of natural numbers, let the *diagram of* r be the unique non-increasing sequence $w \in \mathbb{N}^*$ such that $\textsc{Par}(w) = r$. We define the order on ranks as follows: $r \leq r'$ if the diagram of r is lexicographically smaller than the diagram of r'. For instance, $\{7, 5, 2, 2\} < \{7, 7, 3\}$.

5 Bounded width

Definition 8.5.1: (Parikh-equivalent derivation trees)

We call two derivation trees $\mathcal{T}, \mathcal{T}'$ Parikh-equivalent if

$$\textsc{Par}(\textsc{yield}(\mathcal{T})) = \textsc{Par}(\textsc{yield}(\mathcal{T}')).$$

In this last section we formulate and prove main ingredients necessary for the proof of Theorem 8.1.1: Lemma 8.5.2 states that derivation trees of a 1-CFG are Parikh-equivalent to derivation trees of bounded width, and Lemma 8.5.7 says that the languages generated by derivation trees of bounded width have always rational Parikh images. These two lemmas immediately imply Theorem 8.1.1.

Lemma 8.5.2: (Small width trees)

Consider a fixed 1-CFG. For a sufficiently large constant n, every derivation tree is Parikh-equivalent to a derivation tree of width smaller than n.

Proof. Let $m = |\Delta_2|$ be the number of production rules of arity 2. Fix an arbitrary $n \geq f(6) \cdot 2m$, for f given by Lemma 8.2.3. We show:

Claim 8.5.3

For every derivation tree $\mathcal{T}$ of $\mathcal{G}$ of width $\geq n$ there exists a Parikh-equivalent derivation tree $\mathcal{T}'$ of rank strictly larger than $\mathcal{T}$, but of the same size (= the number of nodes) as $\mathcal{T}$.

The claim is sufficient for proving Lemma 8.5.2. Indeed, as the transformation preserves the size, the rank can increase only finitely many times. Therefore, by iterating the transformation we ultimately arrive at a derivation tree $\mathcal{T}'$ whose rank can not be further increased. By Claim 8.5.3, the width of $\mathcal{T}'$ is forcedly smaller than n, as required in Lemma 8.5.2.

From now on we concentrate on proving Claim 8.5.3. Let $\mathcal{T}$ be a derivation tree of width $\geq n$. Consider some fixed n-cut $\{x_1, \ldots, x_n\}$ and disjoint paths $\pi_1, \ldots, \pi_n$ in $\mathcal{T}$ of length $\geq n$, each path π_i going from x_i to some leaf.

Consider a fixed path π_i. It contains $\geq n$ register-updating production traversals, and therefore by the pigeonhole principle the same production rule $q \xrightarrow{\varphi} p\ p' \in \Delta_2$ and the same (say left) register-updating traversal repeats at least $n' = \frac{n}{2m}$ times along π_i. We apply Lemma 8.2.3 for $k = 3$ to deduce that, as $n' \geq f(6) \geq f(3)$, some two of these traversals, as depicted in the figure below,

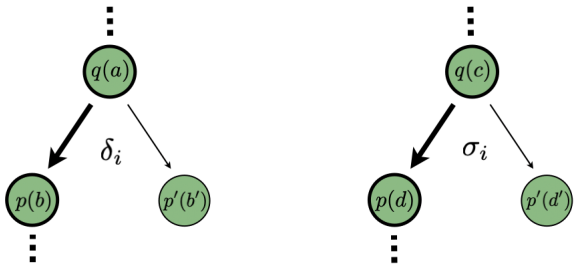


Figure 8.5: Traversals in a derivation tree.

are compatible, by which we mean that their underlying 3-tuples $\langle a, b, b' \rangle$ and $\langle c, d, d' \rangle$ are so. Thus each path π_i traverses a pair of compatible productions δ_i, σ_i which agree on the production rule and (left or right) traversal.

We now repeat a similar argument for paths. As before, by the pigeonhole principle in at least n' paths π_i, the same production rule and the same traversal was used in productions δ_i and σ_i derived in the above reasoning. We now apply Lemma 8.2.3 for $k = 6$ to deal with pairs $\langle \delta_i, \sigma_i \rangle$ of productions, where a pair $\langle \delta_i, \sigma_i \rangle$ induces a 6-tuple obtained by concatenating two underlying 3-tuples of δ_i and σ_i. Since $n' \geq f(6)$, according to the lemma some two of these pairs, say $\langle \delta_i, \sigma_i \rangle$ and $\langle \delta_j, \sigma_j \rangle$, are compatible (by which we mean that the two induced 6-tuples are so, similarly as above).

We have thus four productions $\delta, \sigma, \bar{\delta}, \bar{\sigma}$, traversed by two disjoint paths in $\mathcal{T}$ (we do not depict non-terminals from now on, as all the four productions are induced by the same rule) as shown in Figure 8.6.

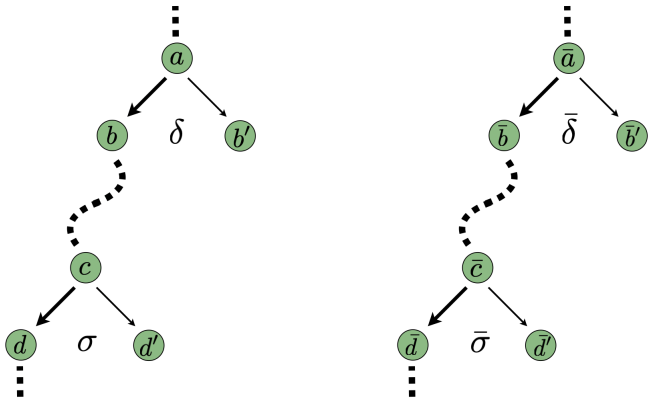


Figure 8.6: Repeating derivations.

Claim 8.5.4

The four underlying triples $\langle a, b, b' \rangle$, $\langle c, d, d' \rangle$, $\langle \bar{a}, \bar{b}, \bar{b}' \rangle$ and $\langle \bar{c}, \bar{d}, \bar{d}' \rangle$ are pairwise compatible.

Proof of Claim 8.5.4. By the construction we have compatibility of $\langle a, b, b' \rangle$ and $\langle c, d, d' \rangle$, $\langle a, b, b' \rangle$ and $\langle \bar{a}, \bar{b}, \bar{b}' \rangle$, and of 6-tuples $\langle a, b, b', c, d, d' \rangle$ and $\langle \bar{a}, \bar{b}, \bar{b}', \bar{c}, \bar{d}, \bar{d}' \rangle$. It only remains to prove compatibility of $\langle a, b, b' \rangle$ and $\langle \bar{c}, \bar{d}, \bar{d}' \rangle$, and of $\langle c, d, d' \rangle$ and $\langle \bar{a}, \bar{b}, \bar{b}' \rangle$. We concentrate of the former pair, as the other one is dealt with similarly.

The equality types of triples $\langle a, b, b' \rangle$ and $\langle \bar{c}, \bar{d}, \bar{d}' \rangle$ are the same, since so are the equality types of $\langle a, b, b' \rangle$ and $\langle c, d, d' \rangle$, and of $\langle c, d, d' \rangle$ and $\langle \bar{c}, \bar{d}, \bar{d}' \rangle$. We concern the first coordinate of the triples. If $a = \bar{c}$, compatibility condition is satisfied on this coordinate. Otherwise, supposing $a \neq \bar{c}$, we derive $a \notin \{\bar{c}, \bar{d}, \bar{d}'\}$: if $a \neq \bar{a}$ then this follows due to compatibility of the two 6-tuples, and if $a = \bar{a}$ then this follows due to compatibility of $\langle \bar{a}, \bar{b}, \bar{b}' \rangle$ and $\langle \bar{c}, \bar{d}, \bar{d}' \rangle$; symmetrically we derive $\bar{c} \notin \{a, b, b'\}$. The two remaining coordinates are dealt with similarly. □

We are now prepared to cutting and pasting in $\mathcal{T}$. For convenience we use below atoms a, b, etc. to identify respective nodes (keeping in mind potential equalities between these atoms).

Definition 8.5.5: (Relevance)

Define the *relevance* of a node x in $\mathcal{T}$ as the maximal length of a path from the root of $\mathcal{T}$ to a leaf that traverses x.

Recall that all the four traversals of Claim 8.5.4 are register-updating, and hence $a \neq b$, and likewise for other tuples. We distinguish three cases, depending on the relation of b' to a and b.

Case 1 $b' = a$**.** By symmetry assume, w.l.o.g., that the relevance $\bar{r}$ of the node $\bar{b}$ is larger or equal to the relevance r of the node b. We cut the segment of $\mathcal{T}$ starting from the edge $a \to b$ and ending with the edge $c \to d$, and paste this segment between the nodes $\bar{a}$ and $\bar{b}$ as depicted in the figure 8.7:

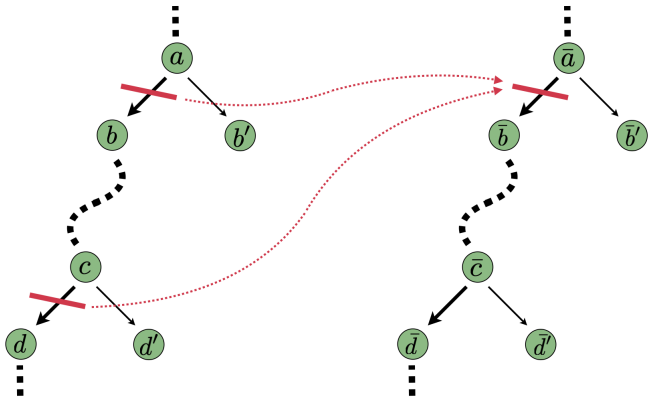


Figure 8.7: Case 1: Before cutting and pasting.

By Claim 8.5.4 the tree $\mathcal{T}'$ so obtained is still a derivation tree:

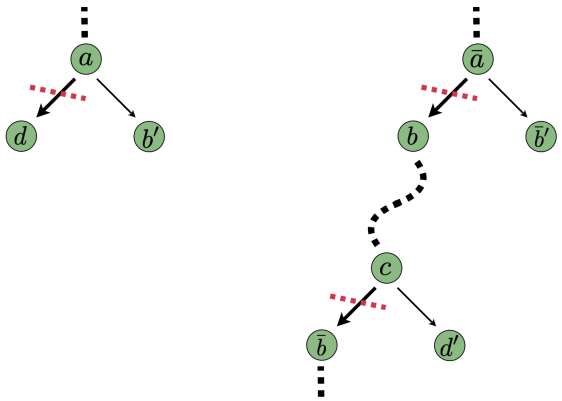


Figure 8.8: Case 1: After cutting and pasting.

Indeed, $d \neq a$ (because either $d = b$ or d does not appear elsewhere) and hence $q(a) \to p(d)\ p'(b') \in \Pi_2$ is a production; likewise for the two remaining productions above.

Furthermore, we claim that rank of $\mathcal{T}'$ is strictly larger than rank of $\mathcal{T}$. To this aim we analyse the effect of cut and paste on the lengths of the paths from the root to a leaf in $\mathcal{T}$. First of all, paths not traversing b or $\bar{b}$ remain untouched. Concerning the cut, the lengths of all paths from the root to a leaf in $\mathcal{T}$ traversing the nodes b and d decrease, and the lengths of paths traversing b but not d change arbitrarily. Concerning the paste, the lengths of all paths traversing $\bar{b}$ strictly increase. Thus some paths of length $\bar{r}$ in $\mathcal{T}$ get prolonged, and all other affected paths in $\mathcal{T}$ have lengths at most $r \leq \bar{r}$. Therefore the rank of $\mathcal{T}'$ is strictly larger than the rank of $\mathcal{T}$.

Case 2 $b' = b$**.** By symmetry assume, w.l.o.g., that the relevance $\bar{r}$ of the node $\bar{a}$ is larger or equal to the relevance r of the node a. We cut the segment of $\mathcal{T}$ starting from the edge $a \to b$ and ending with edges $c \to d$ and $c \to d'$, and paste this segment between the node $\bar{a}$ and the nodes $\bar{b}, \bar{b}'$, and moreover cut the subtree rooted in b' and paste it in place of the subtree rooted in $\bar{b}'$, as depicted in Figure 8.9.

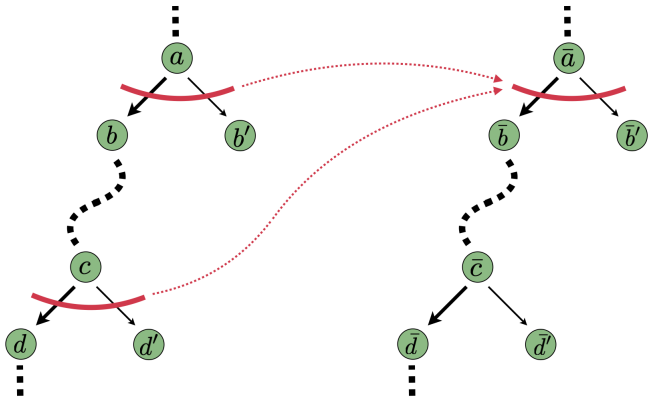


Figure 8.9: Case 2: Before cutting and pasting.

By Claim 8.5.4 the tree $\mathcal{T}'$ obtained is a derivation tree, as shown in Figure 8.10.

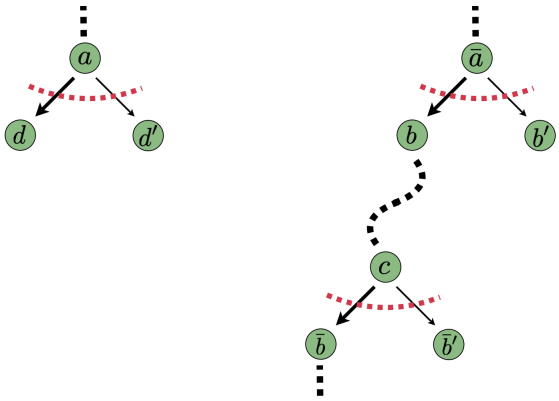


Figure 8.10: Case 2: After cutting and pasting.

Similarly as before, we claim that the rank of $\mathcal{T}'$ is strictly larger than the rank of $\mathcal{T}$. First of all, paths from the root to a leaf in $\mathcal{T}$ not traversing a or $\bar{a}$ remain untouched. Furthermore, the lengths of all paths from the root to a leaf that traverse $\bar{a}$ strictly increase, and the lengths of all paths traversing a either decrease or change arbitrarily. Thus some paths of length $\bar{r}$ in $\mathcal{T}$ get prolonged, and all other affected paths in $\mathcal{T}$ have lengths at most $r \leq \bar{r}$. Therefore the rank strictly increases.

Case 3 $b' \notin \{a, b\}$**.** In this case one can use any of the two cut-and-paste schemes described above.

The proof of Claim 8.5.3 is thus completed, and hence so is the proof of Lemma 8.5.2. □

Given a 1-CFG $\mathcal{G}$, recall that $L_{q(a)}(\mathcal{G}) \subseteq \Sigma^*$ stands for the language of yields of all complete derivation trees $\mathcal{T}$ with root labeled by $q(a)$. We denote by $H_{q(a),n} \subseteq L_{q(a)}(\mathcal{G})$ the subset of words generated by a derivation tree of height at most n, and by $W_n \subseteq L(\mathcal{G})$ the subset of words generated by a derivation tree of width at most n. We now prove, for every $n \in \mathbb{N}$, rationality of the languages $H_{q(a),n}$, and then use it to derive rationality of the Parikh image of the language W_n.

Lemma 8.5.6: (Bounded height trees)

Consider a fixed 1-CFG. For every $n \in \mathbb{N}$, the languages $H_{q(a),n}$ have rational Parikh images.

Proof. We use Substitution Lemma (Lemma 4.3.2 in Chapter 4) several times.

For a nonterminal $q \in Q$ and an atom $a \in \mathbb{A}$, consider the set of derivation trees of $\mathcal{G}$ with root labeled by $q(a)$, which use only productions with the left-hand side in $Q \times \{a\}$ (thus every non-leaf in such a tree belongs to $Q \times \{a\}$), and where every leaf belongs either to $H \times \mathbb{A}$ or to $Q \times (\mathbb{A} - \{a\})$. Intuitively, we stop derivation at a terminal, or at a configuration with register value different than a (i.e., at first register update along every path). The language $K_{q(a)}$ generated by such trees is obtained by applying a substitution to a classical context-free language (with the finite set $Q \times \{a\}$ of nonterminals), and thus has rational Parikh image.

The proof of Lemma 8.5.6 is by induction on n. In case $n = 0$, we observe that $H_{q(a),0}$ is the restriction of $K_{q(a)}$ to terminals $H \times \{a\}$:

$$H_{q(a),0} \quad = \quad L_{q(a)} \cap (H \times \{a\})^*$$

and thus is itself a classical context-free language (with the finite set $Q \times \{a\}$ of nonterminals and the finite set $H \times \{a\}$ of terminals); in consequence, it has rational Parikh image.

For the induction step we assume rationality of languages $H_{q(a),n}$, and observe that $H_{q(a),n+1}$ is obtained by applying to the language $K_{q(a)}$ the substitution:

$$p(b) \quad \mapsto \quad H_{p(b),n} \qquad \langle h, b\rangle \quad \mapsto \quad \langle h, b\rangle,$$

where $p \in Q$, $h \in H$, and $b \in \mathbb{A}$. Indeed, intuitively speaking, $K_{q(a)}$ allows for exactly one register update, while $H_{p(b),n}$ allows for $\leq n$ additional register updates along every path. Therefore $H_{q(a),n+1}$ has rational Parikh image, as required.

□

Lemma 8.5.7: (Bounded width tree)

Consider a fixed 1-CFG. For every $n \in \mathbb{N}$, the language W_n has rational Parikh image.

Proof. For a fixed $n \in \mathbb{N}$, consider an arbitrary fixed derivation tree $\mathcal{T}$ of width at most n, and the subset $\mathcal{H} \subseteq \mathcal{T}$ of those nodes which have height at least $n+1$. The set $\mathcal{H}$ is closed under ancestors and is thus itself a tree; contrarily to $\mathcal{T}$ whose non-leaf nodes have arity 2, the tree $\mathcal{H}$ may contain nodes of arity 1. Notably, as a special case $\mathcal{H}$ may be empty.

By assumption, width of $\mathcal{T}$ is at most n, and hence it may contain n-cuts but no $(n+1)$-cuts. This implies that the largest cut in $\mathcal{H}$ has size n. In consequence:

Claim 8.5.8

$\mathcal{H}$ has at most n leaves, and hence at most $n-1$ nodes of arity 2.

Let $\mathcal{L}$ denote the finite multiset (of size at most n) of configurations $q(a)$ labelling leaves of $\mathcal{H}$.

Any maximal path in $\mathcal{H}$ consisting of nodes of arity 1 we call a *segment*. Thus $\mathcal{H}$ decomposes uniquely into leaves, nodes of arity 2, and segments. An example tree shown in Figure 8.11 has $n = 4$ leaves, 3 nodes of arity 2 and 4 segments (depicted by blue areas) of size 3, 2, 2 and 1, respectively.

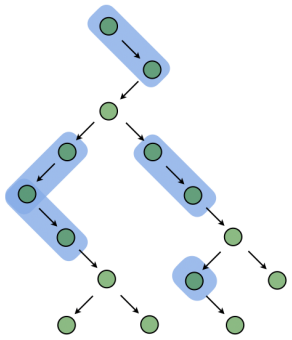


Figure 8.11: Segments.

Using Claim 8.5.8 we deduce:

Claim 8.5.9

$\mathcal{H}$ contains at most $2n-1$ segments.

We again rely below on Substitution Lemma (Lemma 4.3.2 in Chapter 4).

Let $\mathcal{S}$ denote the finite multiset (of size at most $2n-1$) of pairs of configurations $\langle q(a), p(b)\rangle$ labelling ends of segments. Let $\widetilde{S}_{q(a)\,p(b)}$ be obtained from the side-effect language $S_{q(a)\,p(b)}$ by the equivariant substitution (for $q' \in Q$):

$$q'(c) \quad \mapsto \quad H_{q'(c),n};$$

by Lemmas 8.5.6 and 8.3.5 languages $\widetilde{S}_{q(a)\,p(b)}$ have thus rational Parikh images. Let's define ($\prod$ denotes concatenation)

$$L_{\mathcal{L},\mathcal{S}} \quad = \quad \Big(\prod_{q(a)\in\mathcal{L}} H_{q(a),n} \Big) \quad \Big(\prod_{\langle q(a),p(b)\rangle\in\mathcal{S}} \widetilde{S}_{q(a)\,p(b)} \Big)$$

as the concatenation of two concatenations, one of them ranging over $\mathcal{L}$ and the other one over $\mathcal{S}$. By the very definition of the language $L_{\mathcal{L},\mathcal{S}}$ we have

Claim 8.5.10

$\textsc{Par}(\textsc{yield}(\mathcal{T})) \in \textsc{Par}(L_{\mathcal{L},\mathcal{S}})$.

Claim 8.5.11

The languages W_n and $K \;=\; \bigcup_{\mathcal{L},\mathcal{S}} L_{\mathcal{L},\mathcal{S}}$ are Parikh-equivalent, where $\mathcal{L},\mathcal{S}$ range over all possible sets arising from all derivation trees $\mathcal{T}$ of $\mathcal{G}$ of width at most n.

Proof of Claim 8.5.11. The inclusion $\textsc{Par}(W_n) \subseteq \textsc{Par}(K)$ we deduce by Claim 8.5.10. For the converse inclusion $\textsc{Par}(K) \subseteq \textsc{Par}(W_n)$ we should prove: for every $\mathcal{L},\mathcal{S}$ arising from some derivation tree $\mathcal{T}$ of width at most n, the language $L_{\mathcal{L},\mathcal{S}}$ is included in W_n. Indeed, given $\mathcal{T}$ and $\mathcal{H}$ used to derive sets $\mathcal{L},\mathcal{S}$, we observe that every word $w \in L_{\mathcal{L},\mathcal{S}}$ is Parikh-equivalent to the yield of a derivation tree $\mathcal{T}'$ of width at most n, obtained from $\mathcal{H}$ by replacing each leaf labelled by $q(a)$ with a tree of height $\leq n$ with root labeled by $q(a)$, and replacing each segment with a sequence of productions, where every side-effect $q(a)$ is replaced by a tree of height $\leq n$ with root labeled by $q(a)$. Thus $\textsc{Par}(w) \in \textsc{Par}(W_n)$. □

Finally, we derive rationality of $\textsc{Par}(K)$. By Lemmas 8.5.6 and 8.3.5 the languages $L_{\mathcal{L},\mathcal{S}}$ have rational Parikh images. Due to the bounds on the size of $\mathcal{L}$ and $\mathcal{S}$ (cf. Claim 8.5.8 and Claim 8.5.9), by Lemma 3.1.12 the set of all possible pairs $\mathcal{L},\mathcal{S}$ is orbit-finite. Therefore K, as an orbit-finite union of languages with rational Parikh images, has a rational Parikh image too.

The proof of Lemma 8.5.7 is thus complete. □

Remark 8.5.12. (Possible generalisations) We suppose that the method developed in this section can be extended to prove rationality of Parikh images of languages of one-register context-free grammars of any arity. Furthermore, we suppose also that the method can be extended to show that every context-free grammar (with arbitrarily many registers) is Parikh-equivalent to a non-deterministic register automaton (with possibly more registers).

6 HRA and 1-CFG

By Theorem 8.1.1, languages of 1-CFG have rational Parikh images, and by Corollary 7.6.5, Parikh images of HRA are exactly all rational sets of data vectors. Therefore we can conclude the following:

Theorem 8.6.1: (Parikh's equivalence of 1-CFG and HRA)

Every one-register context-free language is Parikh-equivalent to the language of some hierarchical register automaton.

Summary

In this chapter, we applied rationality of Parikh images of non-deterministic one-register automata to reason about Parikh images of one-register context-free languages. We designed a transformation of derivation trees that decreases width, and used it to show that one-register context-free languages have rational Parikh images.

Chapter 9

Conclusions

Objective

In this final chapter we give a brief outline on the state of the art of our knowledge on Parikh images of register automata and context-free grammars, and discuss open problems, conjectures, and possible fuerther research directions.

Contents

1 Summary

In Chapter 1 we posed the following four questions:

(1) Are Parikh images of languages of non-deterministic register automata always semi-linear?

(2) Are Parikh images of languages of non-deterministic register automata always rational sets?

(3) Does (2) hold for register context-free languages?

(4) Are Parikh images of register automata and register context-free languages the same?

We provided the negative answer to question (1) in Chapter 5: we proved that languages of a deterministic one-register automata need not be semi-linear, under a suitable extension of classical semi-linear sets allowing orbit-infinite unions instead of finite ones. However, Parikh image of this language is *rational*, again under a likewise extesions of classical rational sets. We introduced concepts of orbit-finite rational languages and rational sets in Chapter 4, and discussed some of the properties like *Substitution Lemma* – one of the core tools in the subsequent proofs.

We provided a partial answer to question (2) in Chapter 6 for nondeterministic one-register automata. The crucial parts of the proof resorts to a Kleene type technique of transition elimination, and to a necessary condition for a Hamiltonian cycles in directed graphs. Using the rationality of Parikh images of nondeterministic one-register automata, we were able to show the main result of Chapter 8, namely Parikh images of one-register context-free languages are rational. This provides a partial answer to question (3), but also to question (4): one-register context-free grammars are Parikh-equivalent to register automata (with possibly more than one register).

In attempt to fully answer (2), we identified a new strict subclass of register automata (in terms of languages recognized), which we call *hierarchical register automata*, in Chapter 7. This model is strictly more expressive than one-register automata in terms of both languages recognized, as well as in terms of Parikh images thereof. In the same chapter, we proved that hierarchical automata recognize all rational languages, and their Parikh images are rational sets of data vectors. As an outcome of these results, in order to show that some language of register automaton or register grammar has rational Parikh image, it is enough to provide a hierarchical register automaton with the same Parikh image. In particular, by the results of Chapter 6, one-register context-free grammars are Parikh-equivalent to hierarchical register automata (with possibly more than one register).

2 Open questions

Our main techniques of Chapter 6 heavily exploit the assumption that an automaton (or grammar) has only one register, and we don't know if these techniques can be lifted to automata (or grammars) with more registers. But we have strong evidences to believe that the whole class of

languages of register automata as well as register context-free grammars have rational Parikh images. However, the long sequence of proofs in Chapter 6 demonstrates that proving rationality of Parikh image may be surprisingly difficult, even for languages recognized by simple NRA.

3 Conjectures

We end up the thesis with the hypotheses we came up with during our investigations. The proofs of rationality of Parikh images of 1-NRA languages, as well as the likewise proof for the hard language from Chapter 7, make us believe that Theorems 6.1.1 and 8.1.1 can be extended to higher numbers of registers:

Conjecture 9.3.1: (Parikh images of NRA and CFG)

Parikh images of languages of all nondeterministic register automata and all register context-free grammars are rational.

Furthermore, we believe that rational sets of data vectors, as introduced in this thesis, can be efficiently manipulated. We tend to believe that the most fundamental algorithmic problems about these sets are decidable. For instance, we believe that one can decide non-emptiness of intersection of these sets, or inclusion of these sets. We are thus tempted by formulating the following general conjecture:

Conjecture 9.3.2: (Algorithmic problems on rational sets)

Basic algorithmic problems are decidable for rational sets of data vectors.

Bibliography

Barloy, Corentin and Lorenzo Clemente (2021). "Bidimensional Linear Recursive Sequences and Universality of Unambiguous Register Automata". *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*. Ed. by Markus Bläser and Benjamin Monmege. Vol. 187. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:15 (cit. on p. 38).

Bojańczyk, Mikołaj (2011). "Data Monoids". *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10-12, 2011, Dortmund, Germany*. Ed. by Thomas Schwentick and Christoph Dürr. Vol. 9. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 105–116 (cit. on pp. 10, 39).

— (2019). "Slightly Infinite Sets" (cit. on pp. 9, 18, 22, 24, 26, 32, 34, 38–40, 54, 91).

— (2020). *Regular expressions for data words*. Personal communication (cit. on pp. 10, 45, 47).

Bojańczyk, Mikołaj, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin (2011). "Two-variable logic on data words". *ACM Trans. Comput. Log.* 12.4, 27:1–27:26 (cit. on p. 40).

Bojańczyk, Mikołaj, Bartek Klin, and Sławomir Lasota (2011). "Automata with Group Actions". *Proc. LICS 2011*, pp. 355–364 (cit. on pp. 18, 20).

— (2014). "Automata theory in nominal sets". *Log. Methods Comput. Sci.* 10.3 (cit. on pp. 10, 18, 20, 38, 42).

Bojańczyk, Mikołaj, Bartek Klin, Sławomir Lasota, and Szymon Toruńczyk (2013). "Turing Machines with Atoms". *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*. IEEE Computer Society, pp. 183–192 (cit. on p. 42).

Bojańczyk, Mikołaj, Bartek Klin, and Joshua Moerman (2021). "Orbit-Finite-Dimensional Vector Spaces and Weighted Register Automata". *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, pp. 1–13 (cit. on pp. 18, 38).

Bojańczyk, Mikołaj and Sławomir Lasota (2010). "An Extension of Data Automata that Captures XPath". *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*. IEEE Computer Society, pp. 243–252 (cit. on p. 40).

Bojańczyk, Mikołaj and Rafal Stefański (2020). "Single-Use Automata and Transducers for Infinite Alphabets". *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*. Ed. by Artur Czumaj, Anuj Dawar, and Emanuela Merelli. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 113:1–113:14 (cit. on p. 39).

Cheng, Edward Y. C. and Michael Kaminski (1998). "Context-Free Languages over Infinite Alphabets". *Acta Informatica* 35.3, pp. 245–267 (cit. on pp. 10, 42).

Chistikov, Dmitry and Christoph Haase (2016). "The Taming of the Semi-Linear Set". *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*. Ed. by Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi. Vol. 55. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 128:1–128:13 (cit. on p. 53).

Clemente, Lorenzo and Sławomir Lasota (2015a). "Reachability analysis of first-order definable pushdown systems". *Proc. CSL 2015*. Ed. by Stephan Kreutzer. Vol. 41. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 244–259 (cit. on pp. 10, 18, 42).

— (2015b). "Timed Pushdown Automata Revisited". *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*. IEEE Computer Society, pp. 738–749 (cit. on p. 42).

— (2018). "Binary Reachability of Timed Pushdown Automata via Quantifier Elimination and Cyclic Order Atoms". *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. Ed. by Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 118:1–118:14 (cit. on p. 42).

— (2021). "Reachability relations of timed pushdown automata". *J. Comput. Syst. Sci.* 117, pp. 202–241 (cit. on p. 42).

Clemente, Lorenzo, Sławomir Lasota, Ranko Lazic, and Filip Mazowiecki (2017). "Timed pushdown automata and branching vector addition systems". *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, pp. 1–12 (cit. on p. 42).

— (2019). "Binary Reachability of Timed-register Pushdown Automata and Branching Vector Addition Systems". *ACM Trans. Comput. Log.* 20.3, 14:1–14:31 (cit. on p. 42).

Colcombet, Thomas, Clemens Ley, and Gabriele Puppis (2015). "Logics with rigidly guarded data tests". *Log. Methods Comput. Sci.* 11.3 (cit. on p. 39).

Demri, Stéphane and Ranko Lazic (2009). "LTL with the freeze quantifier and register automata". *ACM Trans. Comput. Log.* 10.3, 16:1–16:30 (cit. on pp. 36–38).

Eilenberg, Samuel and M.P Schützenberger (1969). "Rational sets in commutative monoids". *Journal of Algebra* 13.2, pp. 173–191 (cit. on pp. 13, 55).

Francez, Nissim and Michael Kaminski (1994). "Finite-Memory Automata". *Theor. Comput. Sci.* 134.2, pp. 329–363 (cit. on pp. 9, 30, 35, 36).

Gabbay, Murdoch and Andrew M. Pitts (2002). "A New Approach to Abstract Syntax with Variable Binding". *Formal Aspects Comput.* 13.3-5, pp. 341–363 (cit. on p. 20).

Ghouila-Houri, A. (1960). "Une condition suffisante d'existence d'un circuit hamiltonien". *C. R. Acad. Sci. Paris* 25, pp. 495–497 (cit. on p. 77).

Grigore, Radu and Nikos Tzevelekos (2016). "History-Register Automata". *Log. Methods Comput. Sci.* 12.1 (cit. on p. 40).

Juzepczuk, Marta (2013). "Zbiory semiliniowe nad nieskończonym alfabetem (in Polish)". MA thesis. University of Warsaw (cit. on pp. 12, 55).

Kaminski, Michael and Tony Tan (2006). "Regular Expressions for Languages over Infinite Alphabets". *Fundam. Informaticae* 69.3, pp. 301–318 (cit. on pp. 10, 45, 47).

Klin, Bartek, Sławomir Lasota, and Szymon Toruńczyk (2021). "Nondeterministic and co-Nondeterministic Implies Deterministic, for Data Languages". *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*. Ed. by Stefan Kiefer and Christine Tasson. Vol. 12650. Lecture Notes in Computer Science. Springer, pp. 365–384 (cit. on pp. 18, 36).

Kühn, Daniela and Deryk Osthus (2012). "A survey on Hamilton cycles in directed graphs". *European Journal of Combinatorics* 33.5, pp. 750–766 (cit. on p. 77).

Kurz, Alexander, Tomoyuki Suzuki, and Emilio Tuosto (2012). "On Nominal Regular Languages with Binders". *Proc. FOSSACS 2012*. Ed. by Lars Birkedal. Vol. 7213. Lecture Notes in Computer Science. Springer, pp. 255–269 (cit. on pp. 10, 45, 47).

Libkin, Leonid, Tony Tan, and Domagoj Vrgoc (2015). "Regular expressions for data words". *J. Comput. Syst. Sci.* 81.7, pp. 1278–1297 (cit. on pp. 10, 45, 47).

Montanari, Ugo and Marco Pistore (1999). "Finite State Verification for the Asynchronous pi-Calculus". *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*. Ed. by Rance Cleaveland. Vol. 1579. Lecture Notes in Computer Science. Springer, pp. 255–269 (cit. on p. 20).

Mottet, Antoine and Karin Quaas (2019). "On the Containment Problem for Unambiguous Single-Register Automata with Guessing". *CoRR* abs/1905.12445. arXiv: 1905.12445 (cit. on p. 38).

Murawski, Andrzej S., Steven J. Ramsay, and Nikos Tzevelekos (2017). "Reachability in pushdown register automata". *J. Comput. Syst. Sci.* 87, pp. 58–83 (cit. on p. 42).

Neven, Frank, Thomas Schwentick, and Victor Vianu (2004). "Finite state machines for strings over infinite alphabets". *ACM Trans. Comput. Log.* 5.3, pp. 403–435 (cit. on pp. 9, 34, 37, 39).

Parikh, Rohit (1966). "On Context-Free Languages". *J. ACM* 13.4, pp. 570–581 (cit. on pp. 29, 53).

Pitts, Andrew M. (2013). *Nominal Sets: Names and Symmetry in Computer Science*. Vol. 57. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (cit. on p. 20).

Sakamoto, Hiroshi and Daisuke Ikeda (2000). "Intractability of decision problems for finite-memory automata". *Theor. Comput. Sci.* 231, pp. 297–308 (cit. on p. 37).

Segoufin, Luc (2006). "Automata and Logics for Words and Trees over an Infinite Alphabet". *Proc. CSL 2006*. Vol. 4207. Lecture Notes in Computer Science. Springer, pp. 41–57 (cit. on pp. 9, 34).

Tzevelekos, Nikos (2011). "Fresh-register automata". *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*. Ed. by Thomas Ball and Mooly Sagiv. ACM, pp. 295–306 (cit. on p. 40).