On the complexity of regular languages

Corentin Barloy

Université de Lille

Doctoral School **MADIS**

University Department **CRIStAL**

Thesis defended by **Corentin Barloy**

Defended on **July 5, 2024**

In order to become Doctor from Université de Lille

Academic Field **Computer Science**

Speciality **Theoretical Computer Science**

On the complexity of regular languages.

Thesis supervised by

Sylvain Salvati	Supervisor
Michaël Cadilhac	Co-Monitor
Charles Paperman	Co-Monitor

Committee members

Referees	Arnaud Durand	Professor at Université Paris-Diderot	
	Howard Straubing	Professor at Boston College	
Examiners	Damien Pous	Senior Researcher at ENS Lyon	Committee President
	Cristina Sirangelo	Professor at Université Paris Cité	
	Sophie Tison	Professor Emeritus at Université de Lille	
Supervisors	Sylvain Salvati	Professor at Université de Lille	
	Michaël Cadilhac	Associate Professor at DePaul University	
	Charles Paperman	Associate Professor at Université de Lille	

Colophon

Doctoral dissertation entitled “On the complexity of regular languages.”, written by Corentin Barloy, completed on September 2, 2024, typeset with the document preparation system LaTeX and the yathesis class dedicated to theses prepared in France.

Université de Lille

Doctoral School **MADIS**

University Department **CRIStAL**

Thesis defended by **Corentin Barloy**

Defended on **July 5, 2024**

In order to become Doctor from Université de Lille

Academic Field **Computer Science**

Speciality **Theoretical Computer Science**

On the complexity of regular languages.

Thesis supervised by

Sylvain Salvati	Supervisor
Michaël Cadilhac	Co-Monitor
Charles Paperman	Co-Monitor

Committee members

Referees	Arnaud Durand	Professor at Université Paris-Diderot	
	Howard Straubing	Professor at Boston College	
Examiners	Damien Pous	Senior Researcher at ENS Lyon	Committee President
	Cristina Sirangelo	Professor at Université Paris Cité	
	Sophie Tison	Professor Emeritus at Université de Lille	
Supervisors	Sylvain Salvati	Professor at Université de Lille	
	Michaël Cadilhac	Associate Professor at DePaul University	
	Charles Paperman	Associate Professor at Université de Lille	

Université de Lille

Université de Lille

École doctorale **MADIS**

Unité de recherche **CRIStAL**

Thèse présentée par **Corentin Barloy**

Soutenue le **5 juillet 2024**

En vue de l'obtention du grade de docteur de l'Université de Lille

Discipline **Informatique**

Spécialité **Informatique Théorique**

Sur la complexité des langages réguliers.

Thèse dirigée par	Sylvain Salvati	directeur
	Michaël Cadilhac	co-encadrant
	Charles Paperman	co-encadrant

Composition du jury

Rapporteurs	Arnaud Durand	professeur à l'Université Paris-Diderot	
	Howard Straubing	professeur au Boston College	
Examinateurs	Damien Pous	directeur de recherche à l'ENS Lyon	président du jury
	Cristina Sirangelo	professeure à l'Université Paris Cité	
	Sophie Tison	professeure émerite à l'Université de Lille	
Directeurs de thèse	Sylvain Salvati	professeur à l'Université de Lille	
	Michaël Cadilhac	mcf à DePaul University	
	Charles Paperman	mcf à l'Université de Lille	

On the complexity of regular languages.

Abstract

Regular languages, languages computed by finite automata, are among the simplest objects in theoretical computer science. This thesis explores several computation models: parallel computing with Boolean circuits, processing of structured documents in streaming, and information maintenance on a structure subject to incremental updates. For the latter, auxiliary structures are either stored in RAM or represented by databases updated by logical formulae.
This thesis investigates the resources required to compute classes of regular languages in each of these models. The methods employed rely on the interaction between algebra, logic, and combinatorics, notably exploiting the theory of finite semigroups. This approach of complexity has proven extremely fruitful, particularly in the context of Boolean circuits, where regular languages play a central role. This research angle was crystallised by Howard Straubing in his book "Finite Automata, Formal Logic, and Circuit Complexity", where he conjectured that any regular language definable by an arbitrary formula from a logic fragment can be rewritten to use only simple, regular predicates.
The first objective of this manuscript is to prove this conjecture in the case of the Σ_2 fragment of first-order logic with a single alternation of quantification. A second result provides a description of space complexity, in the streaming model, for verifying regular properties on trees. Special attention is given to properties verifiable in constant and logarithmic space. A third objective is to describe all regular tree languages that can be incrementally maintained in constant time in RAM. Finally, a last part focuses on the development of efficient logical formulae for maintaining all regular languages in the relational model.

Keywords: regular languages, circuit complexity, finite semigroups, formal logic, tree languages, dynamic problems.

Sur la complexité des langages réguliers.

Résumé

Les langages réguliers, langages calculés par automates finis, sont parmi les objets les plus simples de l'informatique théorique. Cette thèse étudie plusieurs modèles de calculs : le calcul parallèle avec les circuits booléens, le traitement en flot de documents structurés, et la maintenance d'information sur une structure soumise à des mises à jour incrémentales. Pour ce dernier modèle, les structures auxiliaires sont soit stockées en RAM, soit représentées par des bases de données mises à jour par des formules logiques.
Cette thèse étudie les ressources nécessaires pour calculer des classes de langages réguliers dans chacun de ces modèles. Les méthodes employées exploitent l'interaction entre algèbre, logique et combinatoire, en mettant notamment à profit la théorie des semigroupes finis. Cette approche de la complexité s'est notamment montrée extrêmement fructueuse dans le cadre des circuits booléens, où les langages réguliers jouent un rôle central. Cette angle de recherche a été cristallisé par Howard Straubing dans son livre "Finite Automata, Formal Logic, and Circuit Complexity", où il émet la conjecture que tout langage régulier définissable par une formule arbitraire d'un fragment de logique peut être réécrite en utilisant uniquement des prédicats simples, c'est-à-dire réguliers.
Le premier but de ce manuscrit est de prouver cette conjecture dans le cas du fragment Σ_2 de la logique du premier-ordre avec une seule alternance de quantification. Un deuxième résultat propose une description de la complexité en espace, dans le modèle de flot, pour vérifier des propriétés régulières sur des arbres. Une attention particulière est portée aux propriétés vérifiables en espace constant et logarithmique. Un troisième objectif est de décrire tous les langages réguliers d'arbres pouvant être maintenus incrémentalement en temps constant en RAM. Enfin, une dernière partie porte sur le développement de formules logiques efficaces pour maintenir tous les langages réguliers dans le modèle relationnel.

Mots clés : languages réguliers, complexité de circuits, semigroupes finis, logique formelle, languages d'arbres, problèmes dynamiques.

CRIStAL
Université de Lille - Campus scientifique – Bâtiment ESPRIT – Avenue Henri Poincaré – 59655 Villeneuve d'Ascq

Remerciements

Tout d'abord, je souhaite remercier mes trois encadrants pour leur aide et leur soutien, sans lesquels cette thèse n'aurait pas pu voir le jour. Ce furent 4 années merveilleuses en votre compagnie, puisse la *dot-depth* être bientôt résolue!

Merci Charles pour ta bonne humeur de chaques instants, même dans le manque de sommeil. Le chaos ambiant qui t'entoure n'a d'égal que la profondeur de tes connaissances en théorie algébrique des automates.

Merci Michaël pour ton enthousiasme sans bornes. Tu m'as également apporté énormément scientifiquement, depuis que tu m'as conseillé la lecture du "Straubing". Les bières sont plus savoureuses à tes côtés.

Merci Sylvain pour avoir partagé tes connaissances inepuissables avec l'humble thésard que je suis, malgré toutes tes responsabilités. Lourde est la tête qui porte la couronne!

Je remercie également Arnaud Durand et Howard Straubing pour avoir pris le temps de rapporter cette thèse, et Damien Pous, Cristina Sirangelo et Sophie Tison d'avoir acceptés de former ce jury si prestigieux.

Je suis également redevable envers Nath, Filip, Nathan, Shaull, Michaël et Lorenzo pour avoir guidés mes premiers pas dans le monde vaste et incroyable de la recherche. Cette gratitude s'étend à tous les enseignants qui ont un jour pris de leur temps pour me transmetre de leur savoir, de la maternelle au MPRI, en passant par Saint-Louis et l'ENS Ulm. Bien entendu, je ne parle pas uniquement des savoirs scientifiques, mais aussi de tout ce qui a aidé à faconner ma vision du monde.

L'équipe LINKS a été un lieu fantastique où j'ai pu m'épanouir en faisant ce que j'aime. Merci à ses membres: Iovka, Florent, Aurélien, Joachim, Mikael, Charles, Sylvain et Sophie. J'ai promis à Sophie de m'excuser pour l'avoir appélée "*Dinosaurus Emeritus*" tout ce temps: désolé! L'animation dans le bureau B213 a beaucoup joué dans la réussite de cette thèse, désolé également de la gène occasionnée. Merci aux thésards du passé (Claire et Nico), du présent (Antonio et Oliver) et du futur (Bastien et Mathias) de m'avoir supporté.

Faire de la recherche seul dans son coin est peu attrayant, et je dois énormément à toutes les personnes avec qui j'ai un jour travaillé, ainsi que celles rencontrés en conférence et écoles d'été (vive les centres de vacances CNRS). En particulier, j'ai hâte de démarer mon postdoc à Bochum avec Thomas et toute son équipe (Nils, Felix, Marco, Fabian et Florian).

Pour en finir avec la thématique "recherche", le soutien des assistants à la recherche est inestimable. Merci en particulier à Nathalie et Nicole, sans qui la partie administrative aurait été beaucoup plus lourde.

Je remercie toutes les personnes avec qui je me suis lié d'amitiés durant ce long parcours et dont la liste serait trop longue à écrire. Merci à Oliver d'avoir été le compagnon de bureau idéal, surtout quand il fallait se moquer des permanents du labo. Tu vas me manquer.

Ma grande famille a aussi été un soutien indéfectible. Encore une fois, une liste exhaustive serait trop longue mais merci à vous tous. Merci à mes parents et à ma soeur de m'avoir soutenu depuis toutes ces années.

Merci à Riemann d'être très mignon.

Enfin, merci à Célestine d'être la personne la plus incroyable sur Terre. Tu illumines ma vie à chaque instant.

Contents

Introduction

Models of computation are at the heart of theoretical computer science. They are formalisms that describe how an algorithm is executed abstractly. For instance, Turing machines model the mathematical essence of computations, whereas the von Neumann architecture models executions on modern computers, along with the memory management. The usage of ressources of all kinds has been considered extensively, with for instance the complexity classes P, NP and PSPACE that account for the execution time and size of used memory of Turing machines, and the measurement of the number of communications between different memories in the von Neumann model. However, these kinds of models are very expressive, and studying their complexity is a challenging task. Moreover, many semantic properties on them are undecidable, starting from deciding if an algorithm accepts every input, which makes a formal language described by a Turing machine hard to comprehend. There exist restricted models of computation with limited expressive power, but easier to handle and understand, that fix some of these issues. These models are usually amenable to a study of their semantic properties. Some of them are even designed to be compiled efficiently to programs that are optimised for some well-chosen parameters. The best example of a limited yet interesting model is finite automata: their input is received sequentially and only a bounded information can be stored in memory. Moreover, finite automata are designed to be compiled such that their execution is fast and has a bounded memory usage. They also enjoy numerous extensions with various computing mechanisms, that are studied to solve practical problems, like automated verification of softwares.

Automata have been studied for a long time, starting with Kleene's theorem [69] that gives a simple programming language, regular expressions, to specify all languages expressible by a finite automaton. Nowadays, regular expressions have found their place in many fields of computer science, and their optimisation is leading to significant speed-up of the execution of other algorithms. Regular expressions are used in the field of linguistics to describe patterns in natural languages. Linguists often use regular expressions to analyse and describe linguistic phenomena such as morphology and syntax. Regular expressions are commonly used in text processing tasks: search engines use regular expressions to match and retrieve relevant information from vast amounts of text data, and programming language compilers use regular expressions to parse source codes. Regular expressions are utilised in bioinformatics for DNA sequence analysis and pattern matching. Regular expressions play a crucial role in network security applications for tasks like intrusion detection and filtering. Regular expressions are employed in data extraction tasks, where structured data needs to be extracted from unstructured text sources like web pages or documents. They help identify and extract specific patterns or information from large volumes of textual data efficiently. Regular expressions are also integrated into text editors to facilitate powerful search and replace operations.

Computation and logic were deeply interwoven from the start of computer science. Logic is a successful tool to design programming languages, ranging from logic programming languages like Prolog and Datalog, to databases query languages like SQL. More recently, a whole subfield

of complexity theory, descriptive complexity, has been devoted to finding logical description of complexity classes. The goal is to find purely syntactic representation of ressource consumption, which is a deeply semantic property. One the gemstones, and starting points, of descriptive complexity is Fagin's Theorem [39] that shows that the languages in NP are exactly those definable with a formula of existential second-order logic. Finding a logic for the class P is an involved challenge still open in its full generality. This logical appraoch also works for finite automata: Büchi [21] showed their expressive power is the same as monadic second-order formulae. In this case, we also have tools from algebra and topology to study automata. Schützenberger [112] showed how to canonically associate a finite algebraic structure, the syntactic semigroup, to any regular language. Many properties of the language are reflected on the obtained semigroup. For instance, McNaughton and Papert [79] described all the languages computed by a first-order formula. The property they found can be read directly on the syntactic semigroup by Schützenberger theorem [113], yielding a decision algorithm to decide if a language is computed by a first-order formula. A lot of work has been dedicated since to understand classes of regular languages. When the class of regular languages has some nice closure properties, the work of Eilenberg [37] shows that membership to the class can be decided by looking only at the syntactic monoid. In addition, thanks to Reiterman [109], regular languages can be studied through the prism of the so-called profinite topology, a generalisation of the fruitful p-adic topologies in arithmetic. It gives that the classes of languages that fall under the scope of the work of Eilenberg can be described by a set of topological equations. All these techniques are powerful tools for the study of regular languages.

There are many different models of computation, built on different paradigms: what kind of operations can the model perform?, is it deterministic?, is it probabilistic?, are there several parallel computation devices?, what does the input represent and how is it received?, can we reuse previous computations?, … Even when the model is fixed, efficiency can be measured in many ways. Typically, the time of execution and the size of the memory needed are two investigated parameters, as studied in the classical complexity classes P, NP and PSPACE. One could think of many more, like power consumption or ecological impact. We describe all the variants considered in this manuscrit.

- *Finite automata.* They were already introduced: they receive the input sequentially and can only use a memory of bounded size.
- *Boolean circuits.* They are close to hardware and to the electronical circuits we can find in a microprocessor. They consist in logical AND, OR and NOT gates wired together. They correspond to a model of parallel computation, where the gates can be evaluated independently of the others. We consider parameters such as the size of the representation of a circuit, and the time needed for a circuit to stabilise, ie. every gate has reached its final value.
- *Stackless automata.* This variant considers that the input represents a serialised tree (like an XML document), and is received sequentially. This new model can use a single counter to store the depth of the tree in addition of the finite memory of finite automata.
- *Incremental problems.* The memory is not erased between two runs of the algorithm. Therefore, previous results of intermediate computations can be reused to speed up following computations.

In this thesis, we investigate the relation between regular languages and those models, with the help of logic, algebra and topology. Looking at these simple languages inside potentially considerably bigger classes can shed some light on the latter classes. Indeed, this point of view was first used in circuit complexity, and was dramatically successful. The descriptive complexity of circuit classes is known to involve logic formulae that have arbitrary numerical

predicates. Moreover, it is interesting to see how regular languages, witnessing completely sequential computations, help understand the parallel computation of circuits. It started with the first major inexpressibility results for a Boolean circuit for the PARITY language, which is regular. This is the language of words over $\{0,1\}$ that have an even number of 1s, and it was shown by Furst, Saxe and Sipser [41] that it requires either exponentially many gates or a superconstant evaluation time. Later, the study of small-depth circuits has shown that many different classes can be separated with regular languages. Moreover, there are regular languages that are complete, for a very strong notion of reduction, for certain classes of circuits. For instance, there is a language that is complete for the class of circuits with polynomially many gates and logarithmic depth, by Barrington's theorem [13]. This motivates the systematic study of regular languages inside circuits classes. A global picture was obtained by Barrington, Compton, Straubing and Thérien [14], in particular with results on circuits that have modulo counting gates. Howard Straubing [126], in his book, proposed a conjecture to identify all regular languages in circuits classes, based on logic. It states that the regular languages of a logic that can use arbitrary predicates are exactly the languages of the same logic that can only use regular numerical predicates. Thanks to the toolbox of semigroup theory, the latter logic class, with regular numerical predicates, is often decidable and well understood. The conjecture is known to hold for some fragments of logic, but a few counter-examples are known.

In this manuscript, we build on the success of this study, and transfer it to other notions of complexity. For instance, the study of regular languages inside incremental complexity classes has already started. Skovjberg Frandsen, Miltersen and Skyum [122] and Amarilli, Jachiet and Paperman [7] give a trichotomy of regular languages parametrised by their complexity regarding incremental complexity. Parallels are known between incremental classes and circuit classes [28, 81, 33], making it even more tempting to pursue this line of work. For stackless automata, their use is primarly for the validation and querying of XML documents. Querying regular properties is a well identified problem, and specific query languages, like XPath, have been designed. In all those topics, the methodology built for the study of regular languages, ie semigroup theory, inside circuit classes is very fruitful for the other notions of complexity as well.

This manuscript is divided into two parts. The first part introduces the main notions and the previous work. It is itself divided into four chapters.

- **Chapter 1.** We formally introduce the concept of finite automata, both for inputs that are words or trees. We define monadic second-order logic as well, and explicit the many ties it has with regular languages. We see in particular that the expressivity of a formula depends both on syntactic restrictions and on the available numerical predicates.
- **Chapter 2.** Finite semigroups are presented as computation devices that are expressively equivalent with regular languages. It allows to use the algebraic machinery to study regular languages. We give the framework of varieties that is a bridge between classes of languages and classes of semigroups, and present the links with topology and the equational theory of semigroups. Regular numerical predicates in logical formulae will play an important role, and we already show a theory to algebraically take into account the addition of regular predicates into a logic. The chapter is concluded with an equivalent framework for regular tree languages.
- **Chapter 3.** We introduce Boolean circuits, and the classes of small-depth circuits that we consider in this thesis. As for finite automata, we present their descriptive complexity. This time, the formulae under consideration can use arbitrary numerical predicates. We detail the main lower bounds techniques in the field that are used to show inexpressibility results.
- **Chapter 4.** As mentioned previously, we motivate our study of the complexity of regular languages. We especially focus on circuit complexity, and review the major results on

the subject. As mentionned, a fragment of logic has the Straubing property [126] if every formula using arbitrary numerical predicates that defines a regular language can be rewritten using only simpler, regular numerical predicates. Proving that a fragment possesses such a property is very enlightening on the properties of that fragment. We review the known fragment possessing the Straubing property, and we exhibit two fragments that falsify it.

The second part gives our new results for regular languages that belong into several models of computation. It is itself divided into three chapters.

- **Chapter 5.** We characterise the regular languages expressible in Σ_2, the fragment of first-order formulae with one alternation of quantifiers. It is achieved by showing that Σ_2 has the Straubing property. This is done for regular languages with a neutral letter first, by showing lower bounds against depth-3 Boolean circuits with bounded top fan-in. The heart of the combinatorial argument resides in studying how positions within a language are determined from one another. We then extend the result to any regular language by using algebraic techniques coming from finite category theory.
- **Chapter 6.** We study the processing of regular properties of streamed trees, encoded as words. In this work, we work within the weak validation setting of Segoufin and Vianu [116]: the input work has to be the valid encoding of some tree. In this setting, it is hard to characterise all regular languages that can be weakly validated with a finite automaton. Hence we consider special regular languages, named RPQs, whose definitions only depend on the paths within a tree. First, we characterise all RPQ that are weakly validatable with a finite automaton. This shows that finite automata cannot capture many regular properties in this framework. However, using a stack is costly in memory. Thus, we propose an intermediate stackless model based on register automata equipped with a single counter, used to maintain the current depth in the tree. Our main result is an effective charactersation of RPQs that can be weakly validated with a stackless automaton. We conclude the chapter with an algebraic study of the classes appearing.
- **Chapter 7.** We tackle the incremental maintenance of regular languages: a same algorithm has to be run several times in a row, and the intermediate results can be stored to be reused. So, there is a word that is subject to updates and we have to be able to answer efficiently at any moment whether the word belongs to a given language or not. We consider first that we can maintain data structures in the RAM model, following Skovbjerg Frandsen, Miltersen, and Skyum and later Amarilli, Jachiet and Paperman [7]. We transfer their result to the case of tree languages and characterise all regular tree languages that can be maintained with a constant time per update and query in the RAM model. In a second time, we consider the complexity class Dyn-FO, that stems from database theory. The auxiliary data are stored in datebases that can be updated with first-order formulae. In this setting, we study the fine-grained complexity of the formulae needed to maintain regular languages.

Bibliography of the current chapter

[7] Antoine Amarilli, Louis Jachiet, and Charles Paperman. "Dynamic Membership for Regular Languages". In: *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Ed. by Nikhil Bansal, Emanuela Merelli, and James Worrell. Vol. 198. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. DOI: 10.4230/LIPIcs.ICALP.2021.116.

[13] David A. Barrington. “Bounded-width polynomial-size branching programs recognize exactly those languages in NC1”. In: *Journal of Computer and System Sciences* 38 (1989). DOI: 10.1016/0022-0000(89)90037-8.

[14] David A. Barrington, Kevin Compton, Howard Straubing, and Denis Thérien. “Regular languages in NC_1”. In: *Journal of Computer and System Sciences* 44.3 (1992). DOI: 10.1016/0022-0000(92)90014-A.

[21] J. Büchi. “Weak Second-Order Arithmetic and Finite Automata”. In: *Mathematical Logic Quarterly* 6 (1960). DOI: 10.1007/978-1-4613-8928-6_22.

[28] R. F. Cohen and R. Tamassia. “Dynamic expression trees”. In: *Algorithmica* 13 (1995). DOI: 10.1007/BF01190506.

[33] Guozhu Dong and Jianwen Su. “Arity Bounds in First-Order Incremental Evaluation and Definition of Polynomial Time Database Queries”. In: *Journal of Computer and System Sciences* 57.3 (1998). DOI: 10.1006/jcss.1998.1565.

[37] Samuel Eilenberg. “Automata, Languages and Machines, Vol. B”. In: Verlag: Academic Press Inc, 1976.

[39] Ronald Fagin. “Generalized first-order spectra, and polynomial time recognizable sets”. In: *SIAM-AMS Proc.* 7 (Jan. 1974).

[41] Merrick Furst, James B Saxe, and Michael Sipser. “Parity, circuits, and the polynomial-time hierarchy”. en. In: (1984). DOI: 10.1007/BF01744431.

[69] SC Kleene. “Representation of events in nerve nets and finite automata”. In: *Automata Studies: Annals of Mathematics Studies. Number 34* 34 (1956).

[79] Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. research monograph no. 65)*. The MIT Press, 1971.

[81] Peter Bro Miltersen, Sairam Subramanian, Jeffrey Scott Vitter, and Roberto Tamassia. “Complexity models for incremental computation”. In: *Theoretical Computer Science* 130.1 (1994). DOI: 10.1016/0304-3975(94)90159-7.

[109] Jan Reiterman. “The Birkhoff theorem for finite algebras”. In: *algebra universalis* 14 (1982). DOI: 10.1007/BF02483902.

[112] M. P. Schützenberger. “Une théorie algébrique du codage”. fre. In: *Séminaire Dubreil. Algèbre et théorie des nombres* 9 (1955).

[113] M.P. Schützenberger. “On finite monoids having only trivial subgroups”. In: *Information and Control* 8.2 (1965). DOI: 10.1016/S0019-9958(65)90108-7.

[116] Luc Segoufin and Victor Vianu. “Validating Streaming XML Documents”. In: *Proc. PODS 2002*. ACM, 2002. DOI: 10.1145/543613.543622.

[122] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. “Dynamic word problems”. In: *J. ACM* 44.2 (1997). DOI: 10.1145/256303.256309.

[126] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. en. Boston, MA: Birkhäuser, 1994. DOI: 10.1007/978-1-4612-0289-9.

Part I

Preliminaries

Notations

We give here several notations for very classical mathematical notions.

- We use $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ to respectively denote the sets of natural (resp. relative, real) numbers.
- For $x \in \mathbb{R}$, its floor and ceiling are denoted by $\lfloor x \rfloor$ and $\lceil x \rceil$.
- For $n \in \mathbb{N}$, the factorial of n is the number $n! = 1 \times 2 \times \cdots \times n$.
- For A a set, A^c is its complement. For A and B two sets, their set difference is denoted by $A - B$.
- A relation on A is a subset $A \times A$. It is an order if it is reflexive, transitive and antisymmetric. It is an equivalence if it is reflexive, transitive and symmetric. An equivalence class is a set of elements that are all in relation.
- A partition of A is a set of subsets $A_1, \ldots, A_n$ such that every element of A belongs to exactly one of the A_i. The set of equivalence classes of an equivalence relation forms a partition.
- A function $f : A \to B$ is injective if for every $x, y \in A$, $f(x) = f(y)$ implies $x = y$. It is surjective if for every $x \in B$, there exists $y \in A$ such that $f(y) = x$. It is bijective if it is both injective and surjective. The inverse image of a subset $C \subseteq B$ is the set

$$f^{-1}(C) = \{x \in A \mid f(x) \in C\}.$$

- A graph consists in a set of vertices V and a set of edges $E \subseteq V \times V$. A path in a graph is a sequence of vertices such that two consecutive vertices are in relation in E. A cycle is a path such that the first and last vertices are the same. A directed acyclic graph (DAG) is a directed graph that has no cycle.
- We use the Landau notation for asymptotic comparisons of functions. Let $f, g : \mathbb{N} \to \mathbb{N}$ be two functions. We say that $f = o(g)$ if $f(n)/g(n)$ tends to 0 when n tends to infinity. We say that $f = O(g)$ if there is a constant C such that $f(n) \leq C \cdot g(n)$ for every n. We say $f = \theta(g)$ if both $f = O(g)$ and $g = O(f)$ stand.

Chapter 1

Automata and Logic

Outline of the current chapter

Finite automata give a handy abstraction for finite systems arising in computer science. They model systems whose behaviour only depends on a finite information gathered on the previous inputs. Usually, they are machines that take words as input, that is to say that we assume that the input is discrete and fed to the system one at a time. They form a deeply restricted version of Turing machines that has no tape to work on. For instance, it is customary that text editors and lexical analysers are created as finite automata. Indeed, scanning through the code of a program to obtain a syntax tree exclusively needs to remember a finite chunk of the processed string. This theory of finite automata is particularly rich and mathematically elegant, and has been investigated for a long time. Numerous enhancements and generalisations of the power of finite automata have been considered, and led to many results. See [62] for an introduction on automata theory and [94] for a broad coverage on the topic.

One way of expanding the capacity of automata is by making them process tree-structured data instead of words. In this case, we talk about tree automata. See [29] for an introduction on tree automata.

There exists a profound connection between automata theory and formal logic, dating back to Büchi [21] in the sixties who demonstrated that finite automata and monadic second-order logic can exactly express the same languages. This connection has since evolved and found application in various fields such as automated software verification and program synthesis. The scope of formal logic even covers broader complexity classes like P, NP or PSPACE. For instance, Fagin [39] uncovered the equivalence between NP and existential monadic second-order logic. See [130] for an introduction on the links between logic and finite automata.

1.1 Finite automata on finite words

Let A be a finite set, that will be called the *alphabet*. Its elements will be called *letters*. A *word* over A is a finite sequence $w = a_1 \cdots a_n$ of letters of A. The word without any letters is called the *empty word* and is denoted by ε. The *length* of w, denoted by $|w|$, is the integer n. For a a letter, $|w|_a$ is the number of a's in the sequence. The *concatenation* of two words $a_1 \cdots a_n$ and $b_1 \cdots b_m$ is the word $a_1 \cdots a_n b_1 \cdots b_m$. The set of words over A is denoted by A^*. A language $\mathcal{L}$ is a subset of A^*. A language $\mathcal{L}$ has a *neutral letter* if there is a letter a that can be removed or added without affecting membership: for every word u, v, $uv \in \mathcal{L}$ if and only if $uav \in \mathcal{L}$. We call Neut the class of languages with a neutral letter. We will often omit quantifications over the alphabet, as they will be most of the time obvious.

We can now define the most basic recognition device for languages of finite words.

Definition 1.1.
A (non-deterministic) *finite automaton* is a tuple $\mathcal{A} = (Q, A, \delta, I, F)$ where:

- Q is a finite set called the set of states,
- A is an alphabet,
- $\delta \subseteq Q \times A \times Q$ is called the transition relation,
- I and F are both subsets of Q and are respectively called the initial and final sets.

A *run* in $\mathcal{A}$ is an alternating sequence of states and letters $(q_0, a_1, q_1, a_2, \ldots, q_{n-1}, a_n, q_n)$ such that for all $0 \leq i < n$, $(q_i, a_{i+1}, q_{i+1}) \in \delta$. It is graphically denoted by

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots q_{n-1} \xrightarrow{a_n} q_n.$$

Its *label* is the word $a_1 \cdots a_n$. It is *accepting* if $q_0 \in I$ and $q_n \in F$, in which case we say that $a_1 \cdots a_n$ is accepted by the automaton. The language of the automaton, denoted by $\mathcal{L}(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$. A language is *recognisable* if it is the language of some automaton.

Regular expressions. Let $\mathcal{L}$ and $\mathcal{M}$ be two languages. We can define three natural operations on languages:

- Union: $\mathcal{L} \cup \mathcal{M} = \{w \in A^* \mid w \in \mathcal{L} \text{ or } w \in \mathcal{M}\}$,
- Concatenation: $\mathcal{L} \cdot \mathcal{M} = \{uv \mid u \in \mathcal{L} \text{ and } v \in \mathcal{M}\}$,
- Kleene star: $\mathcal{L}^* = \bigcup_{n \in \mathbb{N}} \mathcal{L}^n$, where $\mathcal{L}^n$ denotes the concatenation of $\mathcal{L}$ by itself n^{th} times.

Definition 1.2.
We inductively define the set of regular expressions. It is the smallest set such that

- both $\varnothing$ and ε are regular expressions,
- for every letter a in the alphabet, a is a regular expression,
- for every regular expressions e and f, the expressions $e + f$, $e \cdot f$, e^* are regular as well.

We can associate a language to any regular expression. We define $\mathcal{L}(\varnothing)$ to be the empty language and $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. If $a \in A$, then $\mathcal{L}(a)$ is the singleton $\{a\}$. If e and f are regular expressions, then $\mathcal{L}(e + f) = \mathcal{L}(e) \cup \mathcal{L}(f)$, $\mathcal{L}(e \cdot f) = \mathcal{L}(e) \cdot \mathcal{L}(f)$ and $\mathcal{L}(e^*) = \mathcal{L}(e)^*$. A language is *regular* if it is the language of some regular expression.

The following is the starting point of automata theory.

Theorem 1.3 (Kleene [69])**.**
The set of recognisable languages and the set of regular languages are exactly the same.

We will prefer the adjective "regular" to refer to these languages. We denote by Reg the set of all regular languages.

Minimal automaton. An automaton is said to be *deterministic* if, for every state $q \in Q$ and letter $a \in A$, there is at most one state q' such that $(q, a, q') \in \delta$. In this case, for each letter a, we set δ_a to be the function that associates a state q to this q', and we denote $\delta_a(q)$ by $q \cdot a$. For $u \in A^*$, we extend this definition and notation to the function δ_u. It is well known that every regular language can be recognised by a deterministic automaton that is *minimal* with regard to the number of states. This minimal automaton is unique up to isomorphism.

We can explicitly describe it. Let $\mathcal{L}$ be a language. For $u \in A^*$, the left (resp. right) *quotient* of $\mathcal{L}$ by u is the language $u^{-1}\mathcal{L} = \{v \mid uv \in \mathcal{L}\}$ (resp. $\mathcal{L}u^{-1} = \{v \mid vu \in \mathcal{L}\}$). The Nerode automaton for $\mathcal{L}$ is the automaton $\mathcal{A}_\mathcal{L} = (Q, A, \delta, I, F)$ with:

- Q the set of left quotients,
- δ the set of triplets $(u^{-1}\mathcal{L}, a, (ua)^{-1}\mathcal{L})$ for $u \in A^*$, $a \in A$,
- $I = \mathcal{L}$,
- F the set of left quotients of $\mathcal{L}$ by a word in $\mathcal{L}$.

Fact 1.4 (Nerode [83])**.**
The language $\mathcal{L}$ is regular if and only if it has only a finite number of left quotients. In this case, its Nerode automaton $\mathcal{A}_\mathcal{L}$ is the unique minimal automaton for $\mathcal{L}$.

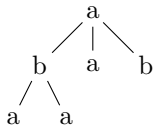
1.2 Finite automata on finite trees

We still have a finite alphabet A. A *tree* and a *forest* over A are mutually defined inductively:

- the empty forest ε is a forest,
- for f a forest and $a \in A$, $a(f)$ is a tree,
- for $t_1, \ldots, t_n$ a sequence of trees, $t_1 + \cdots + t_n$ is a forest.

We will usually remove ε from the expressions, that is to say that we will write a instead of $a(\varepsilon)$.

Example 1.5.
The tree $a(b(a + a) + a + b)$ is graphically represented as



We can define several classical notions. The *root* of a tree $a(f)$ is the letter a. A *subtree* of a tree $a(f)$ where f is defined by the sequence $t_1, \dots, t_n$ is any of the trees t_i, or recursively any subtree of t_i. We can view a tree as a connected directed acyclic graph whose nodes are labelled by elements in A, with a distinguished node that is the root. In this case, a *leaf* is a node of fan-out 0. The other nodes are called *internal*. A *path* in a tree is a path in the underlying graph. A *descendant* of an internal node n is a node m such that there is a path from n to m. If the path is of size 1, we say that m is a child of n. The *relabelling* of a tree is another tree with the same underlying graph and different labels (possibly from another alphabet). It this case, we say that the trees have the same *shape*.

A language of trees is a set of trees.

Note that we defined what is known in the literature as *unranked* trees, that is to say that there are no restrictions on the number of children of a node.

Trees as words. For some applications, such as processing of trees in streaming, trees are seen as words. The *markup encoding* translates a tree over A into a finite word over $A \cup \overline{A}$, where $\overline{A}$ has fresh symbols $\{\overline{a} \mid a \in A\}$. The alphabet A is the set of opening tags and $\overline{A}$ is the set of closing tags. In this context, the encoding is the following:

- $\langle \varepsilon \rangle = \varepsilon$,
- for f a forest and $a \in A$, $\langle a(f) \rangle = a \langle f \rangle \overline{a}$,
- for $t_1, \dots, t_n$ some trees, $\langle t_1 + \cdots + t_n \rangle = \langle t_1 \rangle \cdots \langle t_n \rangle$.

For a tree language $\mathcal{L}$, its markup language is the set $\langle \mathcal{L} \rangle$ of encodings of words of $\mathcal{L}$.

There exists a second possible encoding, which does not recall the symbol that is being closed. Let $\triangleleft$ be a universal closing symbol. The *term encoding* of a tree t is denoted as $[t]$ and is defined as the markup encoding but for f a forest and $a \in A$:

$$[a(f)] = a\,[f]\,\triangleleft\,.$$

Tree automata. There is an analogue to finite automata for trees. The model described here is known as hedge automaton, but we will simply call it a tree automaton for the sake of simplicity.

Definition 1.6.
A (non-deterministic) *finite tree automaton* is a tuple $\mathcal{A} = (Q, A, \delta, F)$ where:

- Q is a finite set called the set of states,
- A is an alphabet,
- δ is a finite set of transitions of the form $a(\mathcal{L}) \to q$ where $a \in A$, $q \in Q$ and $\mathcal{L}$ is a regular language over Q,
- F is a subset of Q and is called the set of final states.

A *run* in $\mathcal{A}$ is a tree over $A \times Q$ such that for every node (a, q) with children $(a_1, q_1), \dots, (a_n, q_n)$, there is a transition $a(\mathcal{L}) \to q$ with $q_1 \cdots q_n \in \mathcal{L}$. Its label is the relabelled tree over A where a node (a, q) becomes a. It is accepting if the root is labelled by a state in F. The language of the automaton, denoted by $\mathcal{L}(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$. A tree language is recognisable (or regular) if it is the language of some tree automaton.

Document Type Definitions. Another way to define a regular tree language is by the Document Type Definition (DTD) syntax. It is closer to real life specification of trees. While the last

approach was bottom-up, going from the leaves to the root, this one is top-down. This formalism has been studied in [86].

A DTD over A is a finite set of rules $a \to \mathcal{L}$ where a is a letter and $\mathcal{L}$ a regular language. A tree can be derived by a DTD d if for every node labelled by a with children labelled by $a_1, \dots, a_n$, there is in d a rule $a \to \mathcal{L}$ with $a_1 \cdots a_n \in \mathcal{L}$.

Definition 1.7.
A specialised DTD over A is a triple (A', d, μ) where:

- A' is a finite alphabet,
- d is a DTD over A',
- μ is a function from A' to A.

A tree t can be derived by a specialised DTD if there is a tree t' over A' that can be derived from d and such that $\mu(t') = t$, where the mapping is applied on each node. It is known that the trees derivable by a specialised DTD are precisely the regular tree languages.

1.3 Monadic second-order logic

1.3.1 Formalism

Syntax. We start by giving the syntactic definition of monadic second-order logical formulae (MSO formulae for short). After laying down all the building blocks, we will explain how to interpret them.

We assume given two infinite sets of *variables* $\mathbb{V}_1$ and $\mathbb{V}_2$. They are respectively called the sets of *first-order* and *second-order* variables. We will usually use the end of the roman alphabet to denote their elements: $x, y, z, x_1, x_2, \dots$ for $\mathbb{V}_1$, and $X, Y, Z, X_1, X_2, \dots$ for $\mathbb{V}_2$.

A *letter predicate* is one of the symbols $\mathbf{a}$ for $a \in A$. A *signature* σ is a set of symbols, the *numerical predicates*, each of them coming with an integer called the *arity* of the predicate.

An *atomic formula* is one of the following expressions:

- $\mathbf{a}(x)$ for $a \in A$ and $x \in \mathbb{V}_1$,
- $R(x_1, \dots, x_k)$ for $R \in \sigma$ of arity n and $x_1, \dots, x_k \in \mathbb{V}_1$,
- $x \in X$ for $x \in \mathbb{V}_1$ and $X \in \mathbb{V}_2$.

Definition 1.8.
We define by induction the set of (MSO) formulae. Let φ and ψ be two formulae.

- An atomic formula is a formula,
- $\varphi \wedge \psi$ is a formula,
- $\neg\varphi$ is a formula,
- $\exists x \varphi$ is a formula for $x \in \mathbb{V}_1$,
- $\exists X \varphi$ is a formula for $X \in \mathbb{V}_2$.

A formula constructed without the last two rules is called *quantifier-free*. A variable x (or X) is *free* in a formula φ if it is not quantified, that is to say that x appears in a subformula on which $\exists x$ (or $\exists X$) is never applied. Formally, we define the set of free first-order (resp. second-order) variables $\mathbb{F}_1(\varphi)$ (resp. $\mathbb{F}_2(\varphi)$) with the same notation as in the preceding definition:

- $\mathbb{F}_1(\mathbf{a}(x)) = \{x\}$,
- $\mathbb{F}_1(R(x_1,\dots,x_k)) = \{x_1,\dots,x_k\}$,
- $\mathbb{F}_1(x \in X) = \{x\}$,
- $\mathbb{F}_1(\varphi \wedge \psi) = \mathbb{F}_1(\varphi) \cup \mathbb{F}_1(\psi)$,
- $\mathbb{F}_1(\neg\varphi) = \mathbb{F}_1(\varphi)$,
- $\mathbb{F}_1(\exists x\varphi) = \mathbb{F}_1(\varphi) - \{x\}$,
- $\mathbb{F}_1(\exists X\varphi) = \mathbb{F}_1(\varphi)$,

- $\mathbb{F}_2(\mathbf{a}(x)) = \emptyset$,
- $\mathbb{F}_2(R(x_1,\dots,x_k)) = \emptyset$,
- $\mathbb{F}_2(x \in X) = \{X\}$,
- $\mathbb{F}_2(\varphi \wedge \psi) = \mathbb{F}_2(\varphi) \cup \mathbb{F}_2(\psi)$,
- $\mathbb{F}_2(\neg\varphi) = \mathbb{F}_2(\varphi)$,
- $\mathbb{F}_2(\exists x\varphi) = \mathbb{F}_2(\varphi)$,
- $\mathbb{F}_2(\exists X\varphi) = \mathbb{F}_2(\varphi) - \{X\}$.

A formula without free variables is called a *sentence*.

Semantic. We can now add some semantics. First of all, we need to give a meaning to the symbols in the signature σ. A *numerical relation* of arity k is a collection for every integer n of a set of k-tuples of $\{1,\cdots,n\}$. An *interpretation* $\mathcal{I}$ of the signature is a function that maps every numerical predicate of arity k to a numerical relation of arity k. Most of the time, it is enough to specify a numerical relation as a set of k-tuples of $\mathbb{N}$. In this case, it is understood that when restricted to an integer n we only keep the tuples containing integers smaller than n.

Example 1.9.
The following are the most frequent numerical relations used as interpretations of numerical predicates.

- Equality predicate $x = y$: all the tuples (i,i) with $i \in \mathbb{N}$,
- Order predicate $x < y$: all the tuples (i,j) with $i < j$,
- Successor predicate $x = y + 1$,: all the tuples $(i, i+1)$ for $i \in \mathbb{N}$,
- Modular predicate $x \equiv_q r$, parametrised by two integers q and r: all the tuples (i) with i congruent to r modulo q.
- Maximal predicate $x = \max$: it associates to $n \in \mathbb{N}$ the singleton $\{(n)\}$.
- Minimal predicate $x = 1$: it consists solely of the singleton $\{(1)\}$.
- For k an integer, we define analogously $x = y + k$, $x = \max - k$ and $x = k$.

An attentive reader would notice that we used the word "predicate" instead of "interpretation" in the examples. Indeed, we will blur the lines between the signature and the interpretation, and often stop making a distinction. For instance, we will allow ourselves to write

$$\exists x \exists y,\ x < y \wedge \mathbf{a}(x) \wedge (y \equiv_3 2)$$

for a formula over a signature with two symbols of arity 1 and 2 and interpreted as the modular interpretation with parameters $(3,2)$ and the order.

We are interested in expressing properties of finite words, hence we want to give a sense to

$$w \models_{\mathcal{I}} \varphi$$

where w is a word, φ a formula and $\mathcal{I}$ an interpretation of the signature. It will mean that w satisfies φ under $\mathcal{I}$. However, while it is intuitive to do so for sentences, it is not straightforward for formulae with free variables. Given the inductive construction, it is crucial to generalise $\models_{\mathcal{I}}$ to formulae with free variables. To do so, we introduce $(\mathcal{V}_1,\mathcal{V}_2)$-*structures*. Roughly speaking, they will allow to store the assignation of free variables. Formally, let $\mathcal{V}_1$ (resp. $\mathcal{V}_2$) be a finite set

of first-order (resp. second-order) variables. A $(\mathcal{V}_1, \mathcal{V}_2)$-structure is a word

$$(a_1, A_1, B_1) \cdots (a_n, A_n, B_n)$$

over the alphabet $A \times 2^{\mathcal{V}_1} \times 2^{\mathcal{V}_2}$ such that $A_1, \ldots, A_n$ partitions $\mathcal{V}_1$. There are no restrictions on the second-order variables. With this definition, we can define the satisfaction relation $w \models_{\mathcal{I}} \varphi$. It will always be the case that w is a $(\mathbb{F}_1(\varphi), \mathbb{F}_2(\varphi))$-structure. For the case of a sentence, it will be satisfied by $(\emptyset, \emptyset)$-structures or, in other terms, words. Note that all variables appearing in atomic formulae are necessarily free.

Definition 1.10.
Let w be a $(\mathcal{V}_1, \mathcal{V}_2)$-structure, φ a MSO formula and $\mathcal{I}$ an interpretation. We characterise $w \models_{\mathcal{I}} \varphi$ by induction:

- $w \models_{\mathcal{I}} \mathbf{a}(x)$ whenever w has a letter (a, A, B) with $x \in A$,
- $w \models_{\mathcal{I}} R(x_1, \ldots, x_k)$ whenever $(i_1, \ldots, i_k)$ is in the numerical relation given by $\mathcal{I}(R)$, where i_j is the position in w in which the variable x_j appears,
- $w \models_{\mathcal{I}} x \in X$ whenever w has a letter (a, A, B) with $x \in A$ and $X \in B$,
- $w \models_{\mathcal{I}} \varphi \wedge \psi$ whenever both $w \models_{\mathcal{I}} \varphi$ and $w \models_{\mathcal{I}} \psi$ hold,
- $w \models_{\mathcal{I}} \neg\varphi$ whenever $w \models_{\mathcal{I}} \varphi$ does not hold,
- $w \models_{\mathcal{I}} \exists x \varphi$ whenever $(a_1, A_1, B_1) \cdots (a_i, A_i \cup \{x\}, B_i) \cdots (a_n, A_n, B_n) \models_{\mathcal{I}} \varphi$ holds for some i,
- $w \models_{\mathcal{I}} \exists X \varphi$ whenever $(a_1, A_1, B_1) \cdots (a_{i_1}, A_{i_1}, B_{i_1} \cup \{X\}) \cdots (a_{i_j}, A_{i_j}, B_{i_j} \cup \{X\}) \cdots (a_n, A_n, B_n) \models_{\mathcal{I}} \varphi$ holds for some subset of positions $i_1, \ldots, i_j$.

The language of a sentence φ under an interpretation $\mathcal{I}$, designated by $\mathcal{L}(\varphi, \mathcal{I})$, is the set of words that satisfy φ under $\mathcal{I}$. We say that two formulae are equivalent (under a given $\mathcal{I}$) whenever they define the same languages.

A prominent logic is the set of first order formulae, abridged FO. It is constructed as MSO, but without the second-order constructions $x \in X$ and $\exists X \varphi$. Its semantic is the same as MSO on the remaining construction rules.

In the formula employed in the everyday life of a scientist, there are also formulae constructed with disjunctions, implications and with universal quantifiers. We chose not to include them in the syntax for brevity, but we provide syntactic sugar for these formulae:

- $\varphi \vee \psi$ is defined as $\neg(\neg\varphi \wedge \neg\psi)$,
- $\varphi \Rightarrow \psi$ is defined as $\psi \vee \neg\varphi$,
- $\varphi \Leftrightarrow \psi$ is defined as $\varphi \Rightarrow \psi \wedge \psi \Rightarrow \varphi$,
- $\forall x \varphi$ is defined as $\neg\exists x \neg\varphi$,
- $\forall X \varphi$ is defined as $\neg\exists X \neg\varphi$.

We will drop the subscript $\mathcal{I}$ in the relation $\models$ whenever the signature is clear.

1.3.2 Links with regular languages.

As mentioned in the introduction, the interplay between logical formalisms and regular languages has been first exhibited by a theorem of Büchi [21].

Theorem 1.11 (Büchi [21])**.**

The regular languages are exactly the languages that are definable by a formula of MSO[<].

The proof goes by directly translating automata into formulae and vice versa. With this equivalence in mind, a valid inquiry arises: is it essential to employ second-order quantifications to fully encompass all regular languages, and characterise the regular languages in $FO[<]$ if it is the case. The answer has been given by McNaughton and Papert [79]: $FO[<]$ is strictly weaker than $MSO[<]$, and they described its languages. Incidentally, it is a striking demonstration of the algebraic method, that we will develop in the next chapter.

Star-free languages. A natural question is to understand which regular languages can be defined with a first-order formula. It can been shown with an elementary proof that the language $(aa)^*$ need second-order quantifiers to be defined by a formula. This motivates the study of regular expressions without the Kleene star. However, with only union and concatenation left, it is impossible to define infinite languages anymore. A way to circumvent this issue is to allow the complementation operation instead of the Kleene star. Indeed, we can define set complementation for a language $\mathcal{L}$ over an alphabet A:

$$\mathcal{L}^{\mathsf{c}} = A^* - \mathcal{L}.$$

Regular expressions are closed under complementation thanks to Kleene theorem and the fact that automata can be complemented.

Definition 1.12.
We inductively define the set of star-free expressions. It is the smallest set such that

- for every letter a in the alphabet, a is a star-free expression,
- for every star-free expressions e and f, the expressions $e + f$, $e \cdot f$, e^{c} are star-free as well.

Note that the regular expressions $\varnothing$ and ε can be defined with star-free expressions and are not required in the definition anymore. The language of a star-free expression is defined as for regular languages with the additional $\mathcal{L}(e^{\mathsf{c}}) = \mathcal{L}(e)^{\mathsf{c}}$. We call a language *star-free* whenever it is the language of some star-free expression.

Theorem 1.13 (McNaughton, Papert [79])**.**
The star-free languages are exactly the languages that are definable by a formula of $FO[<]$.

1.3.3 Defining languages

A *fragment* of MSO is any subset of the whole set of MSO formulae, over a signature with infinitely many symbols of every arity. We will also deal with different classes of predicates (or properly: classes of interpretation of predicates). The chosen predicates will restrict the symbols of the infinite signature that can be used. For F a fragment and $\mathcal{P}$ a class of predicates, we symbolise the class of languages of sentences in F under an interpretation in $\mathcal{P}$ by F[$\mathcal{P}$]. We are deeply concerned with the expressive power of classes of languages of this form. First and foremost, we have to expose the main examples of fragments and predicates.

Predicates. The most basic class of predicates is the class consisting only of the order relation $<$. It has been extensively studied for its tight relationship with regular languages. The class of all successor predicates $x = y + k$, with the maximal $x = \max - k$ and minimal $x = k$ predicates is denoted by loc. There is also mod, the class of all modular predicates for every parameters q and r.

We will write arb for the set of all the possible numerical predicates. Notice that it is an extremely large class, that even encompasses undecidable behaviours.

Example 1.14.
Assume we have an enumeration of all the Turing machines. The predicate $\mathrm{Halt}(x)$ of arity one, which is the set of all integers n such that the n^{th} Turing machine halts, belongs to the class arb. With this very predicate, we can express the (not so natural) language of words with an a at a position that corresponds to a halting Turing machine with the formula:

$$\exists x,\ \mathbf{a}(x) \wedge \mathrm{Halt}(x).$$

One last essential class is the one of regular predicates reg. A predicate is regular if it is definable by a finite automaton. Explicitly, a predicate P of arity k is regular if there exists an automaton $\mathcal{A}$ over $\{a\} \times 2^{\{x_1,\dots,x_k\}}$ such that $(i_1,\dots,i_k)$ is in the n^{th} numerical relation of P if and only if the word

$$(a, A_1)\cdots(a, A_n) \text{ with } x_1 \in A_{i_1}, \dots, x_k \in A_{i_k}$$

is accepted by $\mathcal{A}$. Thanks to the tight links between MSO and regular languages, Straubing [124] and Péladeau [91] have shown that regular predicates can be expressed with very simple formulae. Observe that a formula with k free first-order variables, no free second-order variables and no letter predicates can be used to define a numerical relation of arity k.

Theorem 1.15 (Straubing [124], Péladeau [91])**.**
A predicate is regular if and only if it is definable by a quantifier-free formula of MSO[$<$, loc, mod] without any letter predicate.

This implies that we can simplify the signature of fragments that satisfy a slight property. Namely, take F a fragment and φ a formula of F. Assume replacing any atomic subformula in φ by a Boolean combination of atomic formulae gives a formula that remains in F. Then we have that:

$$\mathrm{F}[\mathrm{reg}] = \mathrm{F}[<, \mathrm{loc}, \mathrm{mod}].$$

Fragments. We have already seen the fragments MSO itself and FO. For the purpose of this document, we will only define fragments that are restrictions or extensions of first-order logic, but always without second-order quantifications.

It is possible to restrict the number of variables that a formula can access, that is to say that we restrict $\mathbb{V}_1$ to be of bounded size. Note that it is not a bound on the number of quantifications, as a same variable name can be reused several times in different scopes. Actually, it is equivalent to assert that every subformula has a bounded number of free variables. To illustrate this, the formula

$$\exists x,\ \mathbf{a}(x) \wedge (\exists y,\ y < x \wedge (\exists x,\ x < y))$$

has three quantifications but only uses two variables (the first two occurrences of x are bound to the first quantifier while the last occurence is bound to the second quantifier). It expresses the fact that there is a letter a that is not in the first two positions of a word.

For k an integer, we denote by FO^k the set of FO formulae that use at most k variables. The most interesting among these fragments is FO^2. Indeed, FO^1 is very weak and cannot express many languages. In particular, it can only use unary numerical predicates, and impose conditions about the letters and their positions but nothing about the global structure of the word. Another reason is that three variables are often enough to express every language.

Fact 1.16 (Kamp [68])**.**
The following equality is true:

$$\mathrm{FO}^3[<] = \mathrm{FO}[<].$$

In another direction, one could want to design fragments that reflect the complexity of a formula. One could think about limiting the number of quantifications in a formula. This approach does not define meaningful fragments. Undoubtedly, in natural language, a formula with a thousand existential quantifiers will be easier to understand that a formula with a single existential quantifier nested with a universal quantifier. This is why it is the number of alternations between the two types of quantifiers that is measured.

A formula is in *prenex normal form* if it is constructed with the application of all the quantification rules last. It looks like:

$$\exists x_1 x_2 x_3 \cdots \forall y_1 y_2 y_3 \cdots \cdots \exists z_1 z_2 z_3 \cdots \varphi$$

where φ is quantifier-free. It is well known that any formula can be put in prenex normal form with some very simple syntactic manipulations. The number of alternations of such a formula is the number of times an existential quantifier is followed by a universal one (and vice versa).

Definition 1.17.
For k an integer, we define the following fragments:

- Σ_k is the set of formulae that can be put in prenex normal form that starts with an existential quantifier and only have $k-1$ quantifier alternations.
- Π_k is the set of formulae that can be put in prenex normal form that starts with a universal quantifier and only have $k-1$ quantifier alternations.
- $\mathcal{B}\Sigma_k$ is the set of Boolean combinations of Σ_k formulae.

Both Σ_0 and Π_0 are defined as the fragment of quantifier-free formulae. For every i we have the inclusions

$$\Sigma_i \subseteq \mathcal{B}\Sigma_i \subseteq \Sigma_{i+1}.$$

For every integer i, we furthermore define the fragment Δ_i as the sentences that can be written both as a Σ_i sentence and a Π_i sentence. This is not purely syntactic, so for $\mathcal{P}$ a class of numerical predicates, we define

$$\Delta_i[\mathcal{P}] = \Sigma_i[\mathcal{P}] \cap \Pi_i[\mathcal{P}].$$

We can interweave several already defined fragments to form new ones. The composition of

two fragments F_1 and F_2 is defined as the set of sentences of F_1 that only uses formulae from F_2 as atomic formulae. It is denoted as $F_1 \circ F_2$.

More quantifiers. One last approach to modify the expressivity of a logic is to enhance its syntax with new quantifiers. We will exclusively consider such quantificators over first-order variables. For instance, we can add a *modular quantifier* $\exists^{q,r}$ for every integers q and r. We call MSO + MOD the set of formulae that can be obtained with the inductive procedure of Definition 1.8 and the additional rule

$$\exists^{q,r} x\varphi \text{ is a formula for } x \in \mathbb{V}_1 \text{ and } \varphi \text{ a formula.}$$

We extend the semantic of Definition 1.10 with:

$w \models_{\mathcal{I}} \exists^{q,r} \varphi$ whenever the number of i such that $(a_1, A_1, B_1)\cdots(a_i, A_i \cup \{x\}, B_i)\cdots(a_n, A_n, B_n) \models_{\mathcal{I}} \varphi$ holds is congruent to r modulo q.

If we restrict the modular quantifiers used to only check constraints modulo integers in a set X, we call the logic MSO + MOD$[X]$.

Example 1.18.
In (MSO + MOD)[<], the formula

$$\exists x,\ \exists^{2,0} y,\ x < y \wedge \mathbf{a}(y) \wedge \mathbf{b}(x)$$

expresses the regular language of words with a b followed by an even number of a's.

We can take a step further by introducing a highly general class of quantifiers. These are called *regular Lindström quantifiers*, in honor of Lindström [76] who studied them in the first place. We will restrict our attention to such quantifiers that entail regular properties. That is, we have a new quantifier for every regular language $\mathcal{L}$ over the alphabet $\{0,1\}$. We can add to the syntax the rule:

$$\exists^{\mathcal{L}} x\varphi \text{ is a formula for } x \in \mathbb{V}_1 \text{ and } \varphi \text{ a formula.}$$

Let $w = (a_1, A_1, B_1)\cdots(a_n, A_n, B_n)$ be a $(\mathcal{V}_1, \mathcal{V}_2)$-structure, $x \in \mathbb{V}_1$ and φ a formula. Let u be the word of size n over $\{0,1\}$ that records in which position x can be assigned so that φ holds, that is for $1 \leq i \leq n$

$$u_i = 1 \text{ if and only if } (a_1, A_1, B_1)\cdots(a_i, A_i \cup \{x\}, B_i)\cdots(a_n, A_n, B_n) \models_{\mathcal{I}} \varphi.$$

With that, we can complete the satisfaction rules with

$$w \models_{\mathcal{I}} \exists^{\mathcal{L}} \varphi \text{ whenever } u \in \mathcal{L}.$$

Example 1.19.
It is possible to catch all of the already defined quantifiers in the scope of regular Lindström quantifiers. The following table gives the equivalence between the standard quantifiers and the languages to see them as regular Lindström quantifiers.

Quantifier	Lindström for $\mathcal{L}=$
$\exists$	$(0+1)^*1(0+1)^*$
$\forall$	1^*
$\exists^{q,r}$	$((0^*10^*)^q)^*(0^*10^*)^r$

Let $\mathcal{C}$ be a class of regular languages. We denote by $\mathrm{MSO}+\mathcal{C}$ the set of formulae that can be obtained with the addition of regular Lindström quantifiers associated with a regular language in $\mathcal{C}$.

Property 1.20.
Let $\mathcal{L}$ be a regular language and φ be a formula of MSO[<]. Then $\exists^{\mathcal{L}} x\varphi$ is equivalent to a formula of MSO[<].

Proof. Let $\psi_{\mathcal{L}}$ be the MSO[<] sentence that exists by Theorem 1.11 such that, for every word w, $w \models \psi_{\mathcal{L}}$ if and only if $w \in \mathcal{L}$. We want to build a formula with two free second-order variables $\psi'(X_0, X_1)$ such that a $(\mathcal{V}_1, \mathcal{V}_2)$-structure

$$(a_1, A_1, B_1)\cdots(a_n, A_n, B_n) \text{ such that exactly one of } X_0 \text{ or } X_1 \text{ belongs to every } B_i$$

satisfies ψ' if and only if the word u that has a 1 exactly in the positions i with $X_1 \in B_i$ belongs to $\mathcal{L}$. To define this formula, we syntactically replace every occurrence of the letter predicates $\mathbf{0}(x)$ (resp. $\mathbf{1}(x)$) in $\psi_{\mathcal{L}}$ by $x \in X_0$ (resp. $x \in X_1$). With this definition, the formula equivalent to $\exists^{\mathcal{L}} x\varphi$ is

$$\exists X_0, X_1,\ (\forall x,\ \varphi(x) \Rightarrow x \in X_1 \wedge \neg\varphi(x) \Rightarrow x \in X_0) \wedge \psi'_{\mathcal{L}}(X_0, X_1).$$

The first part of the formula gives that X_0 and X_1 have to partition the positions in a word, and that X_1 (resp. X_0) contains all the positions such that φ (resp. $\neg\varphi$) holds. We can conclude thanks to the definition of regular Lindström quantifier. □

This entails that when a class of predicates $\mathcal{P}$ contains <, adding regular Lindström quantifiers to MSO is useless:

$$(\mathrm{MSO} + \mathrm{Reg})[\mathcal{P}] = \mathrm{MSO}[\mathcal{P}].$$

However, it might be interesting to add extra quantifiers to smaller fragment of MSO, like FO. This is why we denote by $\mathrm{Lin}(\mathcal{C})$ the set of sentences that uses only Lindström quantifications in $\mathcal{C}$. We also denote by $\mathrm{FO}+\mathcal{C}$ the first-order formulae that can use Lindström quantifications in $\mathcal{C}$. It is equal to $\mathrm{Lin}(\mathcal{C}')$ where $\mathcal{C}'$ is $\mathcal{C}$ enhanced with 1^* and 0^*10^*.

Bibliography of the current chapter

[21] J. Büchi. "Weak Second-Order Arithmetic and Finite Automata". In: *Mathematical Logic Quarterly* 6 (1960). DOI: 10.1007/978-1-4613-8928-6_22.

[29] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. 2008. DOI: https://inria.hal.science/hal-03367725.

[39] Ronald Fagin. "Generalized first-order spectra, and polynomial time recognizable sets". In: *SIAM-AMS Proc.* 7 (Jan. 1974).

[62] John Hopcroft and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[68] Hans Kamp. "Tense Logic and the Theory of Linear Order". PhD thesis. Ucla, 1968.

[69] SC Kleene. "Representation of events in nerve nets and finite automata". In: *Automata Studies: Annals of Mathematics Studies. Number 34* 34 (1956).

[76] Per Lindström. "First Order Predicate Logic with Generalized Quantifiers". In: *Theoria* 32.3 (1966). DOI: 10.1111/j.1755-2567.1966.tb00600.x.

[79] Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. research monograph no. 65)*. The MIT Press, 1971.

[83] Anil Nerode. "Linear automaton transformations". In: *Proceedings of the American Mathematical Society* 9.4 (1958).

[86] Yannis Papakonstantinou and Victor Vianu. "DTD inference for views of XML data". In: *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. PODS '00. New York, NY, USA: Association for Computing Machinery, May 2000. DOI: 10.1145/335168.335173.

[91] Pierre Péladeau. "Logically defined subsets of $\mathbb{N}^k$". In: *Theoretical Computer Science* 93.2 (1992). DOI: 10.1016/0304-3975(92)90328-D.

[94] Jean-Eric Pin. *Handbook of Automata Theory*. EMS Press, 2021. DOI: 10.4171/automata.

[124] Howard Straubing. "Constant-depth periodic circuits". In: *International Journal of Algebra and Computation* 01.01 (1991). DOI: 10.1142/S0218196791000043.

[130] Wolfgang Thomas. "Languages, Automata, and Logic". In: *Handbook of Formal Languages: Volume 3 Beyond Words*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997. DOI: 10.1007/978-3-642-59126-6_7.

Chapter 2

Algebra and Topology

Outline of the current chapter

An automaton, when the information about which states are initial and final is dropped, only consists of several functions $Q \to Q$. There is one such function for every word, and together they form an algebraic structure called a *monoid*. These monoids unravel the combinatorial structure of the automaton and expose many of its properties. Hence, it alleviates the cost of looking at combinatorial properties hidden in an automaton by only looking at a mere algebraic object. Moreover, monoids are objects already subject to a rich literature that is therefore available for the study of regular languages. In particular, *Green's relations* [49] play a significant role. In this spirit, Schützenberger [112] showed how to canonically associate a monoid, to any regular language, that reflects properties of the language. He used these techniques to show that star-free languages can be described thanks to a simple property on their syntactic monoids [113]. It opened the door to a general study of equivalences between classes of languages and classes of monoids. For instance, Simon [119] exhibited a class of monoids that corresponds to the class of piecewise-testable languages, that is to say languages defined by the subwords that can appear in a word. Since then, a line of research is devoted to find classes of monoids that corresponds to

given languages. Later, Eilenberg [37] introduced a general correspondence between classes of languages and classes of monoids that both enjoy several closure properties. Topology plays as well an important role in automata theory. Profinites spaces are topological spaces that contain all the information of finite monoids. They can be constructed through the general prism of Stone duality [42]. It creates a robust equational theory that provides a set of equations on a profinite space to describe any variety of monoids. See [95] for an introduction by Pin to algebraic automata theory and [97] for a survey by Pin on topological methods in automata theory.

A corresponding theory exists for tree automata, although it is generally considered less extensive compared to its counterpart for word automata. See [18] for a review by Bojańczyk on algebras for trees.

2.1 Finite monoids

We start by introducing our first algebraic object.

Definition 2.1.
A *monoid* is a tuple $(M, \cdot)$ where M is a set and $\cdot : M \times M \to M$ is a binary operation on M. It has to satisfy:

- $\cdot$ is associative: for every $x, y, z \in M$, we have $(x \cdot y) \cdot z = x \cdot (y \cdot z)$,
- M has an *identity*: there is an element $1 \in M$ such that for every $x \in M, 1 \cdot x = x \cdot 1 = x$.

We will almost never refer explicitly to the operation and denote a monoid by its set M. The identity can also be named the neutral element. A *semigroup* is a monoid without the requirement about the existence of an identity. A *zero* in a monoid M is an element 0 such that for every $x \in M, 0 \cdot x = x \cdot 0 = 0$. An element x is *invertible* if there is an element y such that $x \cdot y = y \cdot x = 1$. A monoid in which every element is invertible is called a *group*. For an element x and an integer k, x^k is the element $x \cdot \cdots \cdot x$ where the operation is applied k times. We say that x^k is a power of x. An *idempotent* is an element $e \in M$ such that $e^2 = e$.

Fact 2.2.
Let M be a monoid and $x \in M$. There exists a unique idempotent that is a power of x. It is denoted by x^ω.

Let M and N be two monoids. A *morphism* from M to N is a function μ from M to N that is preserving in the sense that:

$$\forall x, y \in M,\ \mu(x \cdot y) = \mu(x) \cdot \mu(y).$$

We say that N is a *submonoid* of M if there exists an injective morphism from N to M. In this case, N is naturally identified with a subset of M. We say that N is a *quotient* of M if there exists a surjective morphism from M to N. We say that N *divides* M if N is the quotient of a submonoid of M. If X is a subset of M, the submonoid of X *generated* by X is the smallest submonoid of M that contains X.

Example 2.3.
Let A be a finite set. The set of words A^* endowed with the concatenation operation is a monoid. Its neutral element is ε, the empty word.

This very important example is called the *free monoid* over A.

Except for free monoids, all the monoids we consider are finite. We will now see how monoids relate to regular languages. To this extent, we need to introduce three notions linked to languages.

Recognisability. A language $\mathcal{L}$ over A is said to be *recognised* by a monoid M if there exists a morphism μ from A^* to M and a subset P of M such that

$$\mathcal{L} = \mu^{-1}(P).$$

Transition monoids. Let $\mathcal{A} = (Q, A, \delta, I, F)$ be a deterministic finite automaton. We can see that the monoid $\mathcal{F}(Q)$ of all functions from Q to Q with the composition of functions as operation is a monoid. The *transition monoid* of $\mathcal{A}$ is the submonoid of $\mathcal{F}(Q)$ generated by the functions δ_a for every letter a.

Syntactic monoids. A congruence on a monoid M is an equivalence relation $\sim$ such that for every $x, y, z, t \in M$,

$$x \sim y \text{ and } z \sim t \text{ implies } x \cdot z \sim y \cdot t \text{ and } z \cdot x \sim t \cdot y.$$

With such an object, we can build a new monoid $M/\sim$ whose base set is the set of equivalence classes of $\sim$. Its operation between two equivalence classes is the class of the product of any two chosen representatives. Let $\mathcal{L}$ be a language. Its *syntactic congruence* is the relation $\sim_{\mathcal{L}}$ on A^* defined by:

$$u \sim_{\mathcal{L}} v \text{ if } \forall x, y \in A^*,\ xuy \in \mathcal{L} \Leftrightarrow xvy \in \mathcal{L}.$$

Its *syntactic monoid* is $M_{\mathcal{L}} = A^*/\sim_{\mathcal{L}}$. The *syntactic morphism* $\mu_{\mathcal{L}}$ is the morphism that maps a word in A^* to its equivalence class in $M_{\mathcal{L}}$.

There are very strong links between regular languages and finite monoids.

Theorem 2.4.
Let $\mathcal{L}$ be a language. Then $\mathcal{L}$ being regular is equivalent to:

- $\mathcal{L}$ is recognised by a finite monoid,
- the syntactic congruence $\sim_{\mathcal{L}}$ has finitely many equivalence classes,
- the syntactic monoid $M_{\mathcal{L}}$ is finite.

Moreover, in this case, the syntactic monoid and the transition monoid of the minimal automaton of $\mathcal{L}$ are the same.

Ideal structure. Let M be a monoid. A *left ideal* I is a subset of M such that for $x \in M$ and $y \in I$, $x \cdot y \in I$. A *right ideal* I is a subset of M such that for $z \in M$ and $y \in I$, $y \cdot z \in I$. A *two-sided ideal* I is a subset of M such that for $x, z \in M$ and $y \in I$, $x \cdot y \cdot z \in I$. Ideals generated by an element are of particular interest. For $x \in M$, its left, right, and two-sided ideals are respectively $M \cdot x$, $x \cdot M$ and $M \cdot x \cdot M$.

The *Green's relations*, introduced by Green [49], are a crucial tool in the understanding of monoid structure.

Definition 2.5.
Let M be a monoid. We define four orders on M. Let $x, y \in M$,

- $x \leq_{\mathcal{R}} y$ whenever $xM \subseteq yM$,
- $x \leq_{\mathcal{L}} y$ whenever $Mx \subseteq My$,
- $x \leq_{\mathcal{J}} y$ whenever $MxM \subseteq MyM$,
- $x \leq_{\mathcal{H}} y$ whenever $x \leq_{\mathcal{R}} y$ and $x \leq_{\mathcal{L}} y$.

This allows to define the four Green's relations:

- $x\mathcal{R}y$ whenever $x \leq_{\mathcal{R}} y$ and $y \leq_{\mathcal{R}} x$,
- $x\mathcal{L}y$ whenever $x \leq_{\mathcal{L}} y$ and $y \leq_{\mathcal{L}} x$,
- $x\mathcal{J}y$ whenever $x \leq_{\mathcal{J}} y$ and $y \leq_{\mathcal{J}} x$,
- $x\mathcal{H}y$ whenever $x \leq_{\mathcal{H}} y$ and $y \leq_{\mathcal{H}} x$.

This definition alone does not give much. We need a few statements to really grasp its value. First of all, we know that all the $\mathcal{H}$-classes that contain an idempotent are groups. Note that Green's relations refine each other along the graph:

$$\begin{array}{ccccc} \mathcal{H} & \subseteq & \mathcal{R} \\ \cap & & \cap \\ \mathcal{L} & \subseteq & \mathcal{J} \end{array}$$

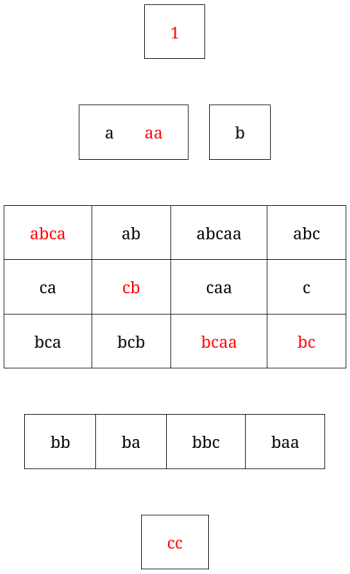
We can therefore mention $\mathcal{R}$-classes and $\mathcal{L}$-classes in a $\mathcal{J}$-class, and $\mathcal{H}$-classes in any other type of Green's class. Green's lemma highlights the structure of a $\mathcal{J}$-class.

Lemma 2.6 (Green)**.**
Let J be a $\mathcal{J}$-class in a monoid M.

(i) All the $\mathcal{H}$-classes with an idempotent are isomorphic groups.
(ii) For every $x, y \in J$, the intersection of the $\mathcal{R}$-classes of x and the $\mathcal{L}$-class of y is not empty.
(iii) There is a bijection between every couple of $\mathcal{R}$-classes (resp $\mathcal{L}$-classes) of J. that preserves the $\mathcal{H}$-classes.

We can therefore represent any $\mathcal{J}$-class as a rectangular grid in which columns represent $\mathcal{L}$-classes, lines $\mathcal{R}$-classes and cells are $\mathcal{H}$-classes. Indeed, (iii) implies that all of the $\mathcal{R}$-classe inside a $\mathcal{J}$-class have the same number of distinct $\mathcal{L}$-classes. By (ii), we can arrange the elements of two $\mathcal{R}$-classes to make the $\mathcal{L}$-classes match. Moreover, all the cells have the same size. The *egg-box diagram* of a monoid is a nice representation of the monoid that emphasises the ideal structure. Each $\mathcal{J}$-class is represented as a rectangular grid and is sorted up to bottom respecting the $\leq_{\mathcal{J}}$ order.

Example 2.7.
We give the egg-box diagram of the syntactic monoid of $b((aa)^*bc)^*$. It has been computed with Paperman's tool Semigroup Online [88]. Elements in red are idempotents.



A striking feature of Green's relations is that the product xy of two elements is always $\mathcal{J}$-smaller than both x and y. It means that when we are doing a product of many monoid elements, the intermediary results only go to the bottom of the egg-box diagram.

2.2 Varieties of finite monoids

2.2.1 Languages and monoids

When investigating a complexity class, the best scenario for describing its regular languages is when we can reduce the problem to an algebraic one. In this case, we have to find a property P about monoids such that a regular language is in the complexity class if and only if its syntactic monoid satisfies P. But for this method to work, the syntactic monoid has to faithfully reflect the property under study. This is why our language classes have to be well-behaved. By a language class, we mean a function that maps an alphabet A to a set of languages over A^*.

Definition 2.8.
A language class $\mathcal{V}$ is a variety if it satisfies, given A and B two alphabets:

- for $\mathcal{L}, \mathcal{M} \in \mathcal{V}(A)$, all of $\mathcal{L} \cap \mathcal{M}$, $\mathcal{L} \cup \mathcal{M}$ and $\mathcal{L}^c$ belong to $\mathcal{V}(A)$,
- for $\mathcal{L} \in \mathcal{V}(A)$ and a morphism $\mu : B^* \to A^*$, then $\mu^{-1}(\mathcal{L})$ is in $\mathcal{V}(B)$,
- for $\mathcal{L} \in \mathcal{V}(A)$ and $u \in A^*$, both $u^{-1}\mathcal{L}$ and $\mathcal{L}u^{-1}$ are in $\mathcal{V}(A)$.

If a language class $\mathcal{V}$ is a variety, then given a language $\mathcal{L}$, we can check if $\mathcal{L} \in \mathcal{V}$ by only computing its syntactic monoid. The class of monoids recognising languages in $\mathcal{L}$ also possesses similar closure properties. We define the *direct product* of two monoids M and N as the monoid whose base set is the Cartesian product $M \times N$ of M and N and whose operation is $(x, y) \cdot (x', y') = (xx', yy')$.

Definition 2.9.
Let **V** be a set of finite monoids. We say that **V** is a variety of monoids whenever:

- if $M, N \in \mathbf{V}$, the monoid $M \times N$ is in **V**,
- if $M \in \mathbf{V}$ and N divides M, then N is in **V**.

Eilenberg [37] exhibited a sturdy connection between varieties of languages and varieties of monoids. Let T_{Monoids} be the function that associates to a variety of languages $\mathcal{V}$ the variety of monoids generated by the syntactic monoids of languages in $\mathcal{V}$. Let $T_{\text{Languages}}$ be the function that associates to a variety of monoids **V** the variety of languages that are recognised by a monoid in **V**.

Theorem 2.10 (Eilenberg [37])**.**
The two functions T_{Monoids} and $T_{\text{Languages}}$ are bijections, and are inverses of each other.

2.2.2 Principal varieties of monoids.

This subsection is devoted to enumerating the varieties of monoids we will encounter in this document. We will also mention the relationship between some of these classes and logical fragments of MSO[<].

Aperiodic monoids. A monoid is said to be *aperiodic* if all of its $\mathcal{H}$-classes are of size one. It is equivalent to ask that any subsemigroup that is a group is of size one. We denote by **A** the variety of all aperiodic monoids. This variety is known to correspond to the star-free languages, and therefore to the logic FO[<].

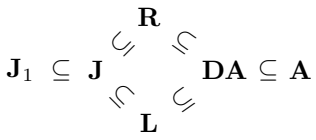
Groups. The variety of all monoids that are groups is denoted by **G**. Morally, this variety corresponds to languages that can be computed with a reversible computation. Indeed, this language class is exactly the class of regular languages that can be computed by a deterministic complete automaton whose transition functions are injective.

Idempotent and commutative monoids. We call $\mathbf{J}_1$ the variety of monoids that are commutative and whose elements are all idempotent. It is a very limited variety. It corresponds to $\text{FO}^1[<]$, the one variable restriction of FO.

$\{\mathcal{R}, \mathcal{L}, \mathcal{J}\}$-trivial monoids. Green's relations are determinant in the study of finite monoids, it is legitimate to check that monoids with restrictions on their Green's relations are varieties. If a monoid only has $\mathcal{J}$ (resp. $\mathcal{L}$, $\mathcal{R}$) classes of size one, we say that it is $\mathcal{J}$ (resp. $\mathcal{L}$, $\mathcal{R}$) trivial, and we denote the corresponding variety by **J** (resp. **L**, **R**). The variety of $\mathcal{J}$-trivial monoids has been studied by Simon [119] who showed that they correspond to the $\mathcal{B}\Sigma_1[<]$ logic.

DA. One last variety is the variety of aperiodic monoids such that all $\mathcal{J}$-classes that contain one idempotent only contain idempotents. In terms of logic, it is known to correspond to $\text{FO}^2[<]$. It is a very fruitful variety that has been extensively studied. An excellent survey on the matter is due to Tesson and Thérien [129].

The following schema summarises the dependencies amongst all those classes. The variety **G** is not comparable to any of the other varieties and is therefore not shown.



2.3 Ordered monoids

It happens that a class of interest is not closed under complementation. It is for instance the case when considering logics like Σ_i or Π_i, as the negation of a formula that starts with an existential quantifier will start with a universal quantifier. A *positive variety* of languages $\mathcal{V}$ is defined exactly like a variety but without the condition that $\mathcal{L}^c \in \mathcal{V}(A)$ whenever $\mathcal{L} \in \mathcal{V}$. This led Pin [92] to define a new type of algebraic object for which there is an Eilenberg-like theorem.

Ordered monoids. An *ordered monoid* is a couple $(M, \leq)$ where M is a monoid and $\leq$ is an order on M that satisfies:

$$\text{for all } x, y, z \in M, \text{ then } x \leq y \text{ implies } xz \leq yz \text{ and } zx \leq zy.$$

An *upset* (resp. *downset*) is a set P such that $y \in P$ whenever $x \leq y$ (resp. $y \leq x$) for some $x \in P$. A morphism of ordered monoids μ is a monoid morphism from M to N with the additional requirement that

$$x \leq y \text{ implies } \mu(x) \leq \mu(y).$$

We can see the free monoid over an alphabet A has an ordered monoid with the trivial order: $x \leq y$ if and only if $x = y$.

An important operation on ordered monoids is duality. For $(M, \leq)$ an ordered monoid, its *dual* is the ordered monoid $(\widetilde{M}, \widetilde{\leq})$ defined by:

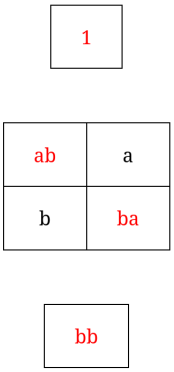
$$\widetilde{M} = M \text{ and } x \widetilde{\leq} y \Leftrightarrow y \leq x.$$

Recognisability. A language $\mathcal{L}$ is said to be recognised by an ordered monoid M if there exists an ordered monoid morphism μ from A^* to M and an upset P of M such that:

$$\mathcal{L} = \mu^{-1}(P).$$

With this definition, a same ordered monoid can recognise a language but not its complement. Indeed, the complement of an upset is not *a priori* an upset.

Example 2.11.
The following monoid (given as an egg-box diagram) recognises the language $(ab)^*$. The order is given by $ab \leq 1$, $ba \leq 1$ and bb is smaller than anyone else. The accepting upset is $P = \{1, ab\}$.



We can see that the complement of $(ab)^*$ is not recognised by this ordered monoid. Indeed, it would be recognised by $P = \{b, ba, a, bb\}$, which is not an upset.

With the knowledge that the complement of an upset is a downset, and is therefore an upset regarding the dual order, we have that for an ordered monoid $(M, \leq)$ and a language $\mathcal{L}$,

$$\mathcal{L} \text{ is recognised by } (M, \leq) \quad \text{if and only if} \quad \mathcal{L}^c \text{ is recognised by } (\widetilde{M}, \widetilde{\leq}).$$

Syntactic ordered monoid. There is an ordered analogue to the syntactic monoid, defined in the same fashion. Let $\mathcal{L}$ be a regular language over A. Its *syntactic order* is the order $\leq_{\mathcal{L}}$ described by:

$$u \leq_{\mathcal{L}} v \text{ if } \forall x, y \in A^*,\ xuy \in \mathcal{L} \Rightarrow xvy \in \mathcal{L}.$$

We can find again the syntactic congruence:

$$u \sim_{\mathcal{L}} v \text{ if and only if } u \leq_{\mathcal{L}} v \text{ and } v \leq_{\mathcal{L}} u.$$

So the syntactic ordered monoid of $\mathcal{L}$ is defined as $(M_{\mathcal{L}}, \leq_{\mathcal{L}})$ where $M_{\mathcal{L}}$ is the syntactic monoid of $\mathcal{L}$.

Computation. To compute the syntactic ordered monoid of a language, it is enough to calculate its minimal automaton $\mathcal{A} = (Q, A, \delta, I, F)$ first, then take its transition monoid. The syntactic order is given as, for two transition functions δ_1 and δ_2:

$$\delta_1 \leq_{\mathcal{L}} \delta_2 \text{ whenever for all transition function } \delta_3 \text{ and } q \in Q,\ \delta_3(\delta_1(q)) \in F \Rightarrow \delta_3(\delta_2(q)) \in F.$$

Positive varieties of monoids. The classical operations on monoids are possible as well with ordered monoids. Let $(M, \leq_1)$ and $(N, \leq_2)$ be two ordered monoids. The direct product of M and N is $(M \times N, \leq)$ where $x \leq y$ stands for $x \leq_1 y$ and $x \leq_2 y$. The notion of submonoid, quotient and division carries to the ordered setting, with the supplementary condition that the morphisms are between ordered monoids. Like before, a set of finite ordered monoids $\mathbf{V}$ is a *positive variety* of monoids if it is stable by direct product and division.

We have an Eilenberg theorem for ordered monoids. Let T^+_{Monoids} be the function that associates to a positive variety of languages $\mathcal{V}$ the positive variety of monoids generated by the syntactic ordered monoids of languages in $\mathcal{V}$. Let $T^+_{\text{Languages}}$ be the function that associates to a positive variety of monoids $\mathbf{V}$ the positive variety of languages that are recognised by an ordered monoid in $\mathbf{V}$.

Theorem 2.12 (Pin [92]).
The two functions T^+_{Monoids} and $T^+_{\text{Languages}}$ are bijections, and are inverses of each other.

We can therefore try to identify the classes of languages and ordered monoids of some logical classes. For instance, for any integer i, the class $\Sigma_i[<]$ is a positive variety of languages.

Fact 2.13.
It is known that the languages in $\Sigma_1[<]$ are exactly those with 1 as a maximum in their syntactic ordered monoid. This positive variety of ordered monoids is known as $\mathbf{J}^+$.

We also have a notion of duality for positive varieties of monoids. For $\mathbf{V}$ a positive variety of monoids, we denote by $\widetilde{\mathbf{V}}$ its dual: the positive variety of all duals of monoids in $\mathbf{V}$. Regarding languages, the dual variety recognises precisely all the complements of languages recognised by $\mathbf{V}$.

2.4 Adding regular predicates

We have described tools to handle, for F a fragment, many classes of the form F[<]. For instance, FO[<] is a variety of monoids. But these tools are not powerful enough to deal with the presence of every regular predicates.

Example 2.14.
The regular language $(aa)^*$ (over a one letter alphabet) is in FO[<, mod] thanks to the formula:

$$\exists x, x \equiv_2 0 \wedge (\forall y, y > x \Rightarrow y = x).$$

This formula ensures that the last letter is at en even position. Its syntactic monoid is the group $\mathbb{Z}/2\mathbb{Z}$ with two elements. Hence it is not aperiodic, and not in FO[<]. Moreover, let $\mu : \{a, b\}^* \to \{a\}^*$ be the morphism defined by $\mu(a) = a$ and $\mu(b) = aa$. We have that $\mu^{-1}((aa)^*) = (b^*ab^*ab^* + b)^*$. With the developments of this section, we will be able to show that this language is not in FO[<, mod]. This implies that FO[<, mod] is not a variety of languages. In can also be seen with the fact that $(aa)^*$ and $(b^*ab^*ab^*)^* + b$ have the same syntactic monoid. Regardless, they can be differentiated by their syntactic morphisms.

2.4.1 $\mathcal{C}$-varieties

Example 2.14 illustrated the fact that the framework of varieties of monoids was not precise enough for our purposes. Pin and Straubing [99] came with the notion of *$\mathcal{C}$-varieties* to capture more phenomena. As advertised, we work with morphisms.

Let $\mu : A^* \to B^*$ be a morphism between two free monoids. We say that it is:

- *non-erasing* if the image of every letter is not the empty word,
- *length-preserving* if the image of every letter is a letter,
- *length-multiplying* if the image of every letter has the same size and is not empty.

In the following, $\mathcal{C}$ is a class of morphisms that contains all length-preserving morphisms and is closed under composition. In practice, $\mathcal{C}$ will either be the class of all morphisms *all*, the class of non-erasing morphisms *ne* or the class of length-multiplying morphisms *lm*.

Definition 2.15.
Let $\mathcal{C}$ be a class of morphisms. A language class $\mathcal{V}$ is a $\mathcal{C}$-variety if it satisfies, for A and B two alphabets:

- for $\mathcal{L}, \mathcal{M} \in \mathcal{V}(A)$, all of $\mathcal{L} \cap \mathcal{M}$, $\mathcal{L} \cup \mathcal{M}$ and $\mathcal{L}^c$ belong to $\mathcal{V}(A)$,
- for $\mathcal{L} \in \mathcal{V}(A)$ and a morphism $\mu : B^* \to A^*$ in $\mathcal{C}$, then $\mu^{-1}(\mathcal{L})$ is in $\mathcal{V}(B)$,
- for $\mathcal{L} \in \mathcal{V}(A)$ and $u \in A^*$, both $u^{-1}\mathcal{L}$ and $\mathcal{L}u^{-1}$ are in $\mathcal{V}(A)$.

Once again, Example 2.14 highlights that monoids cannot capture such varieties and we have to consider morphisms instead. A surjective morphism from a free monoid to a finite monoid is called a *stamp*. Syntactic morphisms are examples of stamps. We define operations on stamps, analogously to monoids. Let $\mu : A^* \to M$ and $\nu : B^* \to N$ be two stamps. A *$\mathcal{C}$-morphism* from μ to ν is a couple $(\alpha : A^* \to B^*, \beta : M \to N)$, where α is in $\mathcal{C}$ and $\beta \circ \mu = \nu \circ \alpha$. It is a *$\mathcal{C}$-quotient* if $\alpha(A) = B$, and a *$\mathcal{C}$-inclusion* if β is injective. The stamp μ *$\mathcal{C}$-divides* ν if there exists a third stamp which is a $\mathcal{C}$-inclusion for μ, and a $\mathcal{C}$-quotient for ν. If $A = B$, the product $\mu \times \nu$ is the stamp $\theta : A^* \to O$ such that for a letter a, $\theta(a) = (\mu(a), \nu(a))$ and O is the submonoid of $M \times N$ generated by the elements $\theta(a)$. We can now state what is a $\mathcal{C}$-variety of stamps.

Definition 2.16.
Let $\mathcal{C}$ be a class of morphisms. Let $\mathbf{V}$ be a set of stamps. We say that $\mathbf{V}$ is a $\mathcal{C}$-variety of stamps whenever:

- for $M, N \in \mathbf{V}$, the stamp $M \times N$ is in $\mathbf{V}$,
- for $M \in \mathbf{V}$ and N that $\mathcal{C}$-divides M, then N is in $\mathbf{V}$.

Note that being a variety of monoids and an *all*-variety of stamps is exactly the same. There is an equivalent of Eilenberg theorem for $\mathcal{C}$-varieties of languages and $\mathcal{C}$-varieties of stamps [99], as well as an equational theory.

An *ordered stamp* is simply a morphism $A^* \to (M, \leq)$ from a free monoid into a finite ordered monoid. As mentionned in [99], we have again a correspondance between positive $\mathcal{C}$-varieties of languages and $\mathcal{C}$-varieties of ordered stamps.

Semigroups. For *ne*-varieties of stamps, there exists another algebraic object with the same recognisability power: semigroups. The relation between the two has been studied by Pin and Straubing in [99, lemma 7.3]. A *variety of semigroups* if a set of semigroups closed under product and division (of semigroups). There is a natural bijection between the two notions of varieties, and they capture the same languages. For S a semigroup, the associated stamp is $\mu : A^* \to S^1$ where S^1 is S with a neutral element added if needed. For $\mu : A^* \to M$ a stamp, the associated semigroup is $\mu(A^+)$. Thanks to that, all the theory afterward could be developed with *ne*-variety of stamps replaced by variety of semigroups (and *ne*-variety of ordered stamps replaced with variety of ordered semigroups).

2.4.2 Wreath product

Adding predicates P into a logic F[σ] corresponds morally to adding the computational power of P into a processed word, before feeding it into a F[σ] formula. The corresponding automata notion is the *cascade* of automata.

Definition 2.17.
Let $\mathcal{A} = (Q, A, \delta, i, F)$ and $\mathcal{A}' = (Q', A \times Q, \delta', i', F')$ be two deterministic automata. Their cascade is the automaton $\mathcal{A}' \circ \mathcal{A} = (Q \times Q', A, \delta'', (i, i'), Q \times F')$ where, for $a \in A$, $q \in Q$ and $q' \in Q'$:

$$\delta''_a(q, q') = (\delta_a(q), \delta'_{(a,q)}(q')).$$

Intuitively, the cascade takes a word u, enhance it with the information given by the computation in $\mathcal{A}$, and finally process it with $\mathcal{A}'$. The *wreath product* is the algebraic counterpart of the cascade operation on automata. Given two monoids M and N, we denote by M^N the set of functions from N to M.

Definition 2.18.
Let M and N be two monoids. The wreath product of M by N, denoted by $M \circ N$ is the monoid with base set $M^N \times N$ and operation

$$(f, x) \cdot (g, y) = (z \mapsto f(z)g(xz), xy).$$

For $\mathbf{V}$ and $\mathbf{W}$ two varieties of monoids, we denote by $\mathbf{V} * \mathbf{W}$ the variety of monoids generated by the monoids of the form $M \circ N$ where $M \in \mathbf{V}$ and $N \in \mathbf{W}$. Both notions are related thanks to the celebrated wreath product principle of Straubing [125].

Theorem 2.19 (Straubing [125])**.**
Let $\mathbf{V}$ and $\mathbf{W}$ be two varieties of monoids. Let $\mathcal{W}$ and $\mathcal{U}$ be the varieties of languages associated with $\mathbf{V}$ and $\mathbf{V} * \mathbf{W}$. Then $\mathcal{U}$ is the smallest variety of languages such that $\mathcal{U}(A)$ contains $\mathcal{W}(A)$ and every language computed by $\mathcal{A}' \circ \mathcal{A}$ where $\mathcal{A}$ has a transition monoid in $\mathbf{W}$ and $\mathcal{A}'$ has a transition monoid in $\mathbf{V}$.

The cascade operation and the wreath product, as well as the wreath product principle, can be extended to make sense for stamps [26] and ordered structures [101]. This operation is crucial in monoid theory. For instance, the celebrated Krohn-Rhodes theorem [74] states that every monoid divides a wreath product of very simple monoids.

Theorem 2.20 (Krohn, Rhodes [74])**.**
Let M be a monoid. Then M divides a wreath product of the form

$$M_1 \circ \cdots \circ M_n$$

where every M_i are either:

- a simple group, that is to say a group that cannot be written as the product of two smaller groups,
- U_2, the syntactic monoid of $(a+b)^*a$.

An important open problem is to decide the minimal number of groups needed for a monoid M to divide a wreath product as in the theorem. We define two varieties of stamps with which wreath products will be applied.

Right trivial stamps. We will be interested in stamps $\mu : A^* \to M$ such that for all u and v non-empty words, the identity

$$\mu(v)\mu(u)^\omega = \mu(u)^\omega$$

stands. We call such a stamp *right trivial*. The class of all such stamps is a *ne*-variety, and is denoted by $\mathbf{D}$. For $\mathbf{V}$ a variety of (ordered) monoids, we denote by $\mathbf{V} * \mathbf{D}$ the *ne*-variety of (ordered) stamps generated by the stamps of the form $M \circ N$ where $M \in \mathbf{V}$ and $N \in \mathbf{D}$.

Modular stamps. For q an integer, the q-modular stamp is the stamp from $\{a\}^*$ to $\mathbb{Z}/q\mathbb{Z}$ that computes the length of a word modulo q. The *lm*-variety $\mathbf{MOD}_q$ is the *lm*-variety of stamps generated by the q-modular stamps. The *lm*-variety $\mathbf{MOD}$ is the union $\bigcup_{q\in\mathbb{N}} \mathbf{MOD}_q$. For $\mathbf{V}$ a *ne*-variety of (ordered) stamps, we denote by $\mathbf{V} * \mathbf{MOD}$ the *lm*-variety of (ordered) stamps generated by the stamps of the form $M \circ N$ where $M \in \mathbf{V}$ and $N \in \mathbf{MOD}$.

2.4.3 Interplay with logic

We refer the reader to [87] for a complete study of the additon of regular predicates into a signature.

Local letter predicates. Adding local predicates is not always an easy task. This is why we introduce a new class of predicates that are not numerical. The local letter predicates, for an integer k, are denoted by $\mathbf{a}_{-\mathbf{k}}$ and are interpreted as, with w a $(\mathcal{V}_1, \mathcal{V}_2)$-structure,

$$w \models_{\mathcal{I}} \mathbf{a}_{-\mathbf{k}}(x) \text{ whenever } w \text{ has a letter } (a, A, B) \text{ with } x - k \in A.$$

We also define $\mathbf{a}(\max - k)$ to be true if the k^{th} letter from the end of the word is an a. Altogether, they form the class of local letter predicates loc_α. Thanks to their unary arity, they will be well suited for an algebraic study. For F a fragment, we will allow writing $\mathrm{F}[\mathrm{loc}_\alpha]$ like with the addition of numerical predicates. In many cases, adding local numerical predicates or local letter predicates give the same result.

Fact 2.21 (Paperman [87, Proposition 3.23])**.**
Let $i \in \mathbb{N}$. Adding local numerical predicates or local letter predicates to the following logics give the same languages: $\Sigma_i[<], \mathcal{B}\Sigma_i[<], \mathrm{FO}^2[<]$.

Wreath products. We first consider the action of adding the local predicates into a signature. The wreath product by $\mathbf{D}$ corresponds to the addition of the local letter predicate.

Theorem 2.22 (Folklore, see [87, Thm 1.13, Thm 3.28]).
Let F be a fragment such that F[<] is a (positive) variety of languages associate to the variety of (ordered) monoids **V**. Then F[<, loc_α] is a (positive) *ne*-variety of languages associated to the *ne*-variety of (ordered) stamps $\mathbf{V} * \mathbf{D}$.

We then consider the adddition of modular predicates into a signature. The wreath product by **MOD** corresponds to the addition of modular predicates. We phrase it to match our purpose of capturing regular predicates.

Theorem 2.23 (Folklore, see [87, Thm 1.13, Thm 4.28]).
Let F be a fragment such that F[<, loc] is a (positive) *ne*-variety of languages associated to the *ne*-variety of (ordered) stamps **V**. Then F[reg] is a (positive) *lm*-variety of languages associated to the *lm*-variety of (ordered) stamps $\mathbf{V} * \mathbf{MOD}$.

Deciding membership in a variety of the form $\mathbf{V} * \mathbf{D}$ or $\mathbf{V} * \mathbf{MOD}$ is notoriously difficult, even if we have the decidability of **V**. However, in some cases, there is a property that can be proved on **V** to simplify membership on $\mathbf{V} * \mathbf{D}$ and $\mathbf{V} * \mathbf{MOD}$.

Local monoids. Let M be a monoid. A *local monoid* of M is a submonoid of the form eMe where e is an idempotent. It is indeed a monoid with identity e. Given a variety of monoids **V**, a very important class of *ne*-variety is the set of stamps $\mu : A^* \to M$ such that the local monoids of $\mu(A^* \setminus \{\varepsilon\})$ are all in **V**. We call this class **LV**. Membership in **LV** is decidable as long as membership in **V** is.

Stable stamps. Let $\mu : A^* \to M$ be a stamp. The set $\mu(A)$ lives in the monoid of subsets of M, hence it possesses an idempotent power. The *stability index* of μ is the smallest integer s such that

$$\mu(A)^s = \mu(A)^{2s}.$$

It implies that $\mu(A^s)$ is a semigroup. We denote by $\mu(A^s)^1$ this semigroup, adjoined with the identity of M if needed. The *stable monoid* is the monoid $\mu(A^s)^1$. The *stable stamp* of μ is the stamp

$$\mu_s : (A^s)^* \to \mu(A^s)^1.$$

For **V** a variety of monoids, **QV** is the *lm*-variety of stamps whose stable monoid is in **V**. For **V** a *ne*-variety of stamps, **QV** is the *lm*-variety of stamps whose stable stamp is in **V**.

Locality. The condition on **V** that allow to simplify the variety $\mathbf{V} * \mathbf{D}$ or $\mathbf{V} * \mathbf{MOD}$ is called *locality*. It involves category theory and is rather involved, so we will not define it, and only states that some varieties have this property. We refer the reader to [131, 8] for references. For instance, **A**, **R**, **L** and **DA** are known to be local. The first variety shown to be not local is **J**, even if $\mathbf{J} * \mathbf{D}$ is decidable. Whenever a local variety is expressive enough to contain $\mathbf{J}_1$, locality is transferred to **LV** [87, Prop 4.24].

Fact 2.24 (Folklore, see [87, Cor 3.32])**.**
Let **V** be a variety of (ordered) monoids. Then

$$\mathbf{V} * \mathbf{D} \subseteq \mathbf{LV}.$$

If moreover **V** is local then

$$\mathbf{V} * \mathbf{D} = \mathbf{LV}.$$

In particular when **V** is local, membership in $\mathbf{V} * \mathbf{D}$ is decidable whenever membership in **V** is.

We can also deduce from the previous fact that, for a language with a neutral letter, its syntactic stamp belongs to $\mathbf{V} * \mathbf{D}$ if and only if it belongs to **V**. The reason behind is that the image of a neutral letter under the syntactic morphism is the neutral element of the monoid, and hence the monoid itself is a local monoid.

Fact 2.25 (Paperman [87, Cor 4.29, Cor 4.31])**.**
Let **V** be a *ne*-variety of (ordered) stamps. Then

$$\mathbf{V} * \mathbf{MOD} \subseteq \mathbf{QV}.$$

If moreover **V** is local then

$$\mathbf{V} * \mathbf{MOD} = \mathbf{QV}.$$

In particular when **V** is local, membership in $\mathbf{V} * \mathbf{MOD}$ is decidable whenever membership in **V** is.

We can also deduce from the previous fact that for a language with a neutral letter, its syntactic stamp belongs to $\mathbf{V} * \mathbf{MOD}$ if and only if it belongs to **V**. It comes from the fact that the neutral letter can be used to fill in spaces, and therefore the image of every word is in the stable monoid.

We show how to use these theorems with two classes of languages.

Example 2.26.
With the fact that **A** and **DA** are both local and contain $\mathbf{J}_1$, we have that

$$\mathrm{FO}[\mathrm{reg}] = \mathbf{QA}$$
$$\mathrm{FO}^2[\mathrm{reg}] = \mathbf{QLDA}$$

The first equality comes from the fact that local predicates can be expressed in first-order logic, and therefore $\mathrm{FO}[<, \mathrm{loc}] = \mathrm{FO}[<]$.

2.5 The profinite realm

The profinite topology is a wonderful and successful tool to study regular languages. It is a generalisation of p-adic topologies that were instrumental in number theory. We first need to recall some topological definitions.

2.5.1 Reminders of topology

A *metric space* (X, d) is a set X with a distance $d : X \times X \to \mathbb{R}$ such that:

- for $x \in X,\ d(x,x) = 0$,
- for $x \neq y,\ d(x,y) > 0$,
- for $x, y \in X,\ d(x,y) = d(y,x)$,
- for $x, y, z \in X, d(x,z) \leq d(x,y) + d(y,z)$.

The last item is called the triangular inequality.

A *Cauchy sequence* is a sequence $(x_n)_{n\in\mathbb{N}}$ such that the elements are arbitrarily close towards infinity. Formally, for all $\varepsilon > 0$, there exists $k \in \mathbb{N}$ such that for all $n, m \geq k$, $d(x_n, x_m) \leq \varepsilon$. A metric space is *complete* if every Cauchy sequence is convergent. The completion of a metric space (X, d) is the metric space $\widehat{X}$ defined as follow. Let $\mathcal{C}(X)$ be the set of Cauchy sequences over X. We define a distance d' of this set as $d'((x_n),(y_n)) = \lim_{n\to\infty} d(x_n, y_n)$. Being at distance zero is an equivalence relation that we denote by $\sim$. The completion $\widehat{X}$ is defined as $(\mathcal{C}(X)/\sim, d')$. It can be shown to be the unique smallest complete metric space containing X, up to isomorphism.

2.5.2 The profinite topology

We say that a monoid M separates two words u and v if there exists a morphism from A^* to M for which u and v have distinct images.

Definition 2.27.
We define a metric d on A^*, for $u, v \in A^*$, as

$$d(u,v) = 2^{-\min\{|M| \mid M \text{ separates } u \text{ and } v\}}.$$

The free profinite space over A is $\widehat{A^*}$, the completion of A^* for this metric d.

Its elements are Cauchy sequences of finite words (u_n) such that, for every morphism μ into a finite monoid, the sequence $(\mu(u_n))$ is ultimately constant. We call them the profinite words.

Therefore, we can extend any morphism $\mu : A^* \to M$ into a continuous morphism $\widehat{\mu} : \widehat{A^*} \to M$.

We can endow $\widehat{A^*}$ with a product structure to make it a monoid. This is just the pointwise concatenation of sequences.

Omega-words. Let $u \in \widehat{A^*}$. The sequence $(u^{n!})$ can be shown to be a Cauchy sequence. This sequence is an element of the free profinite monoid that we denote by u^ω, and call the omega-power of u. It is not random that we chose the symbol ω. Indeed, for all morphism μ into a finite monoid, $\widehat{\mu}(u^\omega) = \widehat{\mu}(u)^\omega$ stands. The set of profinite words constructed from finite words, concatenation and omega-power is called the set of omega-words. It is an important subclass of profinite words thanks to their well-behaved and intuitive behaviour. There are many results that hold for omega-words that no one is able to show in the general case.

Equations. An identity is a couple of profinite words (u, v). Let M be a monoid, it satisfies the identity (u, v) if, for all morphisms μ into M, we have $\mu(u) = \mu(v)$. In the same spirit, let M be an ordered monoid, it satisfies the ordered identity (u, v) if, for all morphisms μ into M, we have $\mu(u) \leq \mu(v)$. It allows defining varieties (resp. positive varieties) by a set of identities (resp. ordered identities): it is the set of monoids that satisfy all the identities. Reiterman [109] showed

variety	identities	logic
G	$x^\omega = 1$	
$\mathbf{J}_1$	$xy = yx, x^2 = x$	$\mathrm{FO}^1[<]$
J	$(xy)^\omega x = (xy)^\omega = y(xy)^\omega$	$\mathcal{B}\Sigma_1[<]$
R	$(xy)^\omega x = (xy)^\omega$	
L	$y(xy)^\omega = (xy)^\omega$	
DA	$(xy)^\omega(yx)^\omega(xy)^\omega = (xy)^\omega, x^{\omega+1} = x^\omega$	$\mathrm{FO}^2[<]$
A	$x^{\omega+1} = x^\omega$	$\mathrm{FO}[<]$
$\mathbf{J}^+$	$x \leq 1$	$\Sigma_1[<]$
$\mathbf{J}^-$	$x \geq 1$	$\Pi_1[<]$

Figure 2.1: Principal varieties

that every variety of monoids can be defined thanks to a (possibly infinite) set of identities. It was later extended to the positive setting.

Let $\mathbf{V}$ be a variety of monoids and u, v two profinite words. We write $u =_{\mathbf{V}} v$ if, for all morphism μ into a monoid of $\mathbf{V}$, we have $\mu(u) = \mu(v)$. If $\mathbf{V}$ is a positive variety instead, we write $u \leq_{\mathbf{V}} v$ if, for all morphism μ into an ordered monoid of $\mathbf{V}$, we have $\mu(u) \leq \mu(v)$.

Theorem 2.28 (Reiterman [109])**.**
We have that:

- Every variety of monoids $\mathbf{V}$ is defined by the set of identities (u, v) such that $u =_{\mathbf{V}} v$.
- Every positive variety of monoids $\mathbf{V}$ is defined by the set of identities (u, v) such that $u \leq_{\mathbf{V}} v$.

Such an equational characterisation can be very handy when dealing with decidability problems. Indeed, when a variety is defined by a finite set of identities involving only omega-words, it is effortless to check if a given monoid belongs to the variety.

Principal varieties. We summarise the varieties seen so far. Finite set of identities for the varieties given in Section 2.2 are given by Fig. 2.1.

The equations for the quantifier alternation hierarchy are of particular importance and we choose to put them apart. They come from a long line of research around the so-called *dot-depth hierarchy*.

Theorem 2.29 (Pin, Weil [100, Theorem 5.9, Corollary 2.5])**.**
Let i be an integer. The languages in $\Sigma_i[<]$ form a positive variety of languages and are defined by the equations:

$$x^\omega \leq x^\omega y x^\omega \text{ for every } y \leq_{\Sigma_{i-1}[<]} x.$$

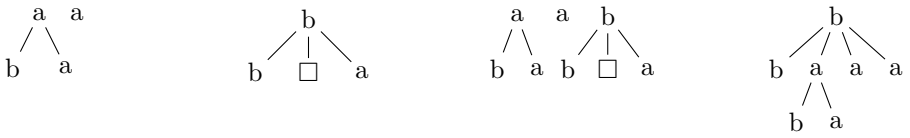
Note that this does not imply that we can decide membership in $\Sigma_i[<]$ for every i. This is a very long-standing open problem, the state-of-the-art being the decidability of $\Sigma_4[<]$ by Place and Zeitoun [103].

2.6 Forest algebras

The algebraic method is dramatically successful in the world of finite words. The obvious next step is to extend this method to regular tree languages. A set of algebraic objects recognising trees has been proposed by Bojanczyk and Walukiewicz [17], the so-called *forest algebras*. However, these are less canonic than the monoid framework. Indeed, it is possible to imagine many algebraic objects, and their use gave less striking results than for words.

The key idea is to mimic forest automata. In their definition, we can see two sorts of regular behaviours, a horizontal one that comes from the regular languages in the rules $a(\mathcal{L}) \to q$, and a vertical one which comes from the global automaton structure. Another difference with words is that the considered operation was straightforwardly chosen: the concatenation of words was the only possibility. The horizontal operation is the concatenation of forests: it is the concatenation of the two lists of trees of the forests. This operation is denoted by $+$. However, it is not clear what a vertical operation can be. To which leaf of the first tree should the root of the second tree be linked? The answer to that question is to introduce another type of object, namely *contexts*. A context is a forest in which one (and only one) leaf is distinguished and carries the symbol $\Box$. Now, the vertical operation is the composition of two contexts: the roots of the second context are now replacing $\Box$ and are the children of its parent. We use $\cdot$ to denote this operation. Note that we can concatenate a forest with a context, giving a context. We can also compose a context with a forest, giving a forest.

Example 2.30.
We give an example of these operations. Here f is a forest, c is a context, $f + c$ is a context, $c \cdot f$ is a forest.



f c $f + c$ $c \cdot f$

With these two types of objects, we can define our algebraic recognisers. It is a two-sorted algebra that mimics the operations we have seen.

Definition 2.31.
A *forest algebra* is a tuple $(H, +, V, \cdot, act, in_l, in_r)$ where

- $(H, +)$ is a monoid, dubbed horizontal monoid, with neutral element 0,
- $(V, \cdot)$ is a monoid, dubbed vertical monoid, with neutral element 1,
- act is an action $V \times H \to H$,
- in_l and in_r are two operations $H \to V$.

We will denote vh instead of $act(v, h)$ and $h + 1$ and $1 + h$ instead of $in_l(h)$ and $in_r(h)$. The algebra must satisfy the following properties:

- $(v \cdot w)h = v(wh)$,
- $(h + 1)g = h + g$ and $(1 + h)g = g + h$,
- for every $v \neq w \in V$, there exists $h \in H$ such that $vh \neq wh$.

The last condition is known as *faithfulness*. When it does not hold in a forest algebra (H, V), we define the *faithful quotient* of (H, V) as the forest algebra (H, V') where V' is the quotient of V by

$$v \sim w \quad \text{iff} \quad \forall h \in H,\ v \cdot h = w \cdot h.$$

This quotient is indeed faithful.

Notice that we denote H additively and V multiplicatively, with neutral elements respectively 0 and 1. This is also respected for powers: for $n \in \mathbb{N}$, $h \in H$ and $v \in V$, the n^{th} power of h is denoted $n \cdot h$ and the one of v is denoted v^n. Similarly, the omega powers of $h \in H$ and $v \in V$ are respectively $\omega \cdot h$ and v^ω.

Thanks to the operation in_r, we have that H is a submonoid of V. It can also be seen symmetrically with in_l.

Fact 2.32.
For every forest algebra (H, V), we have that H is a submonoid of V.

Proof. We use the second axiom of the definition of a forest algebra: for every $h, h' \in H$, $in_r(h) \cdot h' = h + h'$. We consider the operation $in_r : H \to V$. We only have to prove that it is an injective morphism to conclude.

Let $h, h' \in H$, by faithfullness the two elements $in_r(h + h')$ and $in_r(h) \cdot in_r(h')$ have the same image for the action under every $g \in H$. On one hand, $in_r(h + h') \cdot g = h + h' + g$. On the other hand, $in_r(h) \cdot in_r(h') \cdot g = in_r(h) \cdot (h' + g) = h + h' + g$. So in_r is a morphism.

For the injectivity, let $h, g \in H$ such that $in_r(h) = in_r(g)$. In particular, the action of both these elements on 0 gives the same result. Hence, $h = h + 0 = g + 0 = g$. □

The *free forest algebra* over an alphabet A is the forest algebra $A^\Delta = (H_A, V_A)$ where H_A is the set of all trees labelled by A with the concatenation, and V_A is the set of all contexts labelled by A with the composition. The action *act* is the composition between a context and a forest. The operations in_l and in_r are the concatenation between a forest and the neutral context 1 on the left and on the right.

Recognising languages. A morphism between forest algebras (H, V) and (G, W) is a couple of morphisms $\mu : H \to G$ and $\nu : V \to W$ that preserve the operations *act*, in_l and in_r. A tree language $\mathcal{L}$ is recognised by the forest algebra (H, L) if there is a morphism from the free forest algebra $(\mu, \nu) : A^\Delta \to (H, V)$ and a subset P of H such that

$$\mathcal{L} = \mu^{-1}(P).$$

Like for the word case, a morphism from the free forest algebra is entirely determined by its image on the contexts that are a single letter a with a hole directly underneath.

Syntactic forest algebra. Let $\mathcal{L}$ be a tree language. Its syntactic congruence $\sim_\mathcal{L}$ is a duo of congruences on the free forest algebra defined by

$$\begin{aligned} &\text{for } f, g \in H_A, && f \sim_\mathcal{L} g \text{ if } \forall v \in V_A,\ v \cdot f \in \mathcal{L} \Leftrightarrow v \cdot g \in \mathcal{L}, \\ &\text{for } c, d \in V_A, && c \sim_\mathcal{L} d \text{ if } \forall v \in V_A, h \in H_A,\ v \cdot c(h) \in \mathcal{L} \Leftrightarrow v \cdot d(h) \in \mathcal{L}. \end{aligned}$$

This gives a congruence that is compatible with all the forest algebra operations, hence we can quotient the free forest algebra by the syntactic congruence. We call the obtained forest algebra the *syntactic forest algebra* $(H_{\mathcal{L}}, V_{\mathcal{L}})$.

Fact 2.33.
Let $\mathcal{L}$ be a tree language. The following are equivalent:

- $\mathcal{L}$ is regular,
- $\mathcal{L}$ is recognised by a finite forest algebra,
- the syntactic forest algebra of $\mathcal{L}$ is finite.

Thanks to the general theory of monads of Bojanczyk [19], there exist equivalents to the variety theory and to the profinite results that stand for finite words. It is also possible to think of ordered notion for tree languages. We only present here the definitions of unordered varieties and the Eilenberg-like theorem, that will be needed in Chapter 7.

Let $\mathcal{L}$ be a tree language and c be a context, the quotient of $\mathcal{L}$ by c is the language:

$$c^{-1}\mathcal{L} = \{f \mid c \cdot f \in \mathcal{L}\}.$$

We could think of left and right quotient by a forest f as well. They are not needed because they are covered by the quotients by the contexts $f + \Box$ and $\Box + f$. As for words, a tree language class is a function that maps an alphabet A to a set of tree languages over A.

Definition 2.34.
A tree language class $\mathcal{V}$ is a variety if it satisfies, given A and B two alphabets:

- for $\mathcal{L}, \mathcal{M} \in \mathcal{V}(A)$, all of $\mathcal{L} \cap \mathcal{M}$, $\mathcal{L} \cup \mathcal{M}$ and $\mathcal{L}^c$ belong to $\mathcal{V}(A)$,
- for $\mathcal{L} \in \mathcal{V}(A)$ and a morphism $(\mu, \nu) : B^\Delta \to A^\Delta$, then $\mu^{-1}(\mathcal{L})$ is in $\mathcal{V}(B)$,
- for $\mathcal{L} \in \mathcal{V}(A)$ and $c \in V_A$, $c^{-1}\mathcal{L}$ is in $\mathcal{V}(A)$.

We define all operations on forest algebras that allow to define the notion of variety of forest algebras. Let (H, V) and (G, W) be two forest algebras, and μ, ν a morphism between them. The product of (H, V) and (G, W) is simply $(H \times G, V \times W)$ with the act, in_l and in_r being applied component-wise. We say that (H, V) is a *sub-forest algebra* of (G, W) if both μ and ν are injective. We say that (G, W) is a *quotient* of (H, V) if both μ and ν are surjective. We say that (H, V) *divides* (G, W) if (H, V) is the quotient of a sub-forest algebra of (G, W).

Usually, as for words, we would like to specify a sub-forest algebra by giving two subsets of H and V. However, because of the faithfullness condition, it is possible that two subsets given does not form forest algebra. To fix this issue, we can take the faithful quotient of the obtained structure.

Definition 2.35.
Let $\mathbf{V}$ be a set of finite forest algebras. We say that $\mathbf{V}$ is a variety of forest algebras whenever:

- if $(H, V), (G, W) \in \mathbf{V}$, then $(H \times G, V \times W) \in \mathbf{V}$,
- if $(H, V) \in \mathbf{V}$ and (G, W) divides (H, V), then (G, W) is in $\mathbf{V}$.

The functions between varieties of tree languages and varieties of forest algebras are the one

expected. Let T_{Algebras} be the function that associates to a variety of tree languages $\mathcal{V}$ the variety of forest algebras generated by the syntactic forest algebras of languages in $\mathcal{V}$. Let $T_{\text{Languages}}$ be the function that associates to a variety of forest algebras $\mathbf{V}$ the variety of tree languages that are recognised by a forest algebra in $\mathbf{V}$.

Theorem 2.36.
The two functions T_{Algebras} and $T_{\text{Languages}}$ are bijections, and are inverses of each other.

Bibliography of the current chapter

[8] Antoine Amarilli and Charles Paperman. "Locality and Centrality: The Variety ZG". In: *Logical Methods in Computer Science* Volume 19, Issue 4 (Oct. 2023). DOI: 10.46298/lmcs-19(4:4)2023.

[17] Mikolaj Bojanczyk and Igor Walukiewicz. "Forest Algebras". en. In: *Logic and Automata* (Oct. 2006). DOI: https://hal.science/hal-00346087/.

[18] Mikołaj Bojańczyk. "Algebra for trees". en. In: *Handbook of Automata Theory*. Ed. by Jean-Éric Pin. Zuerich, Switzerland: European Mathematical Society Publishing House, Sept. 2021. DOI: 10.4171/Automata-1/22.

[19] Mikołaj Bojańczyk. "Recognisable Languages over Monads". In: *Developments in Language Theory*. Ed. by Igor Potapov. Cham: Springer International Publishing, 2015. DOI: 10.1007/978-3-319-21500-6_1.

[26] Laura Chaubard, Jean-Éric Pin, and Howard Straubing. "Actions, wreath products of $\mathcal{C}$-varieties and concatenation product". In: *Theoretical Computer Science*. In honour of Professor Christian Choffrut on the occasion of his 60th birthday 356.1 (May 2006). DOI: 10.1016/j.tcs.2006.01.039.

[37] Samuel Eilenberg. "Automata, Languages and Machines, Vol. B". In: Verlag: Academic Press Inc, 1976.

[42] Mai Gehrke, Serge Grigorieff, and Jean-Éric Pin. "Duality and Equational Theory of Regular Languages". In: *Automata, Languages and Programming*. Ed. by Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. DOI: 10.1007/978-3-540-70583-3_21.

[49] J. A. Green. "On the Structure of Semigroups". In: *Annals of Mathematics* 54.1 (1951).

[74] Kenneth Krohn and John Rhodes. "Algebraic Theory of Machines. I. Prime Decomposition Theorem for Finite Semigroups and Machines". In: *Transactions of the American Mathematical Society* 116 (1965).

[87] Charles Paperman. "Circuits booléens, prédicats modulaires et langages réguliers". PhD thesis. Université Paris Diderot, 2014.

[88] Charles Paperman. *Semigroup Online*. 2015. URL: https://paperman.name/semigroup/.

[92] Jean-Eric Pin. "A variety theorem without complementation". In: *Russian Mathematics (Izvestija vuzov.Matematika)* 39 (1995).

[95] Jean-Eric Pin. *Mathematical foundations of automata theory*. 2014. URL: http://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf.

[97] Jean-Eric Pin. "Profinite Methods in Automata Theory". In: *26th International Symposium on Theoretical Aspects of Computer Science*. Ed. by Susanne Albers and Jean-Yves Marion. Vol. 3. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2009. DOI: 10.4230/LIPIcs.STACS.2009.1856.

[99] Jean-Eric Pin and Howard Straubing. "Some results on C-varieties". eng. In: *RAIRO - Theoretical Informatics and Applications* 39.1 (Mar. 2010). DOI: 10.1051/ita:2005014.

[100] Jean-Eric Pin and Pascal Weil. "Polynomial Closure and Unambiguous Product". In: *Proceedings of the 22nd International Colloquium on Automata, Languages and Programming*. ICALP '95. Berlin, Heidelberg: Springer-Verlag, 1995. DOI: 10.5555/646249.685349.

[101] Jean-Eric Pin and Pascal Weil. "The wreath product principle for ordered semigroups". In: *Communications in Algebra* 30 (2002).

[103] Thomas Place and Marc Zeitoun. "Going Higher in First-Order Quantifier Alternation Hierarchies on Words". In: *Journal of the ACM* 66.2 (Mar. 2019). DOI: 10.1145/3303991.

[109] Jan Reiterman. "The Birkhoff theorem for finite algebras". In: *algebra universalis* 14 (1982). DOI: 10.1007/BF02483902.

[112] M. P. Schützenberger. "Une théorie algébrique du codage". fre. In: *Séminaire Dubreil. Algèbre et théorie des nombres* 9 (1955).

[113] M.P. Schützenberger. "On finite monoids having only trivial subgroups". In: *Information and Control* 8.2 (1965). DOI: 10.1016/S0019-9958(65)90108-7.

[119] Imre Simon. "Piecewise testable events". In: *Automata Theory and Formal Languages*. Ed. by H. Brakhage. Berlin, Heidelberg: Springer Berlin Heidelberg, 1975. DOI: 10.1007/3-540-07407-4_23.

[125] Howard Straubing. "Families of recognizable sets corresponding to certain varieties of finite monoids". In: *Journal of Pure and Applied Algebra* 15.3 (1979). DOI: 10.1016/0022-4049(79)90024-0.

[129] Pascal Tesson and Denis Thérien. "Diamonds are forever: the variety da". In: *Semigroups, Algorithms, Automata and Languages*. WORLD SCIENTIFIC, Nov. 2002. DOI: 10.1142/9789812776884_0021.

[131] Bret Tilson. "Categories as algebra: An essential ingredient in the theory of monoids". en. In: *Journal of Pure and Applied Algebra* 48.1 (Sept. 1987). DOI: 10.1016/0022-4049(87)90108-3.

Chapter 3

Circuit Complexity and Lower Bounds

Outline of the current chapter

In 1857, George Boole introduced a formal way of describing logical operations: the Boolean algebra. Its object are truth values *true* and *false*, and its operations are logical operators as conjunctions or disjunctions. Almost a century later, Claude Shannon [117] used this mathematical framework to successfully abstract the design of digital circuits. From that efficient electronical circuits, both in size and velocity, could be constructed. This was only the beginning for that abstract model of computation, as it supplies with a parrallel model of computation. Nowadays, its study is a consequent part of complexity theory. In fact, they provide a non-uniform version of the renowned class P of functions computable in polynomial time by a deterministic Turing machine. By their combinatorial design, they are more suitable for the task of finding undoable problems with a certain amount of given ressources. In fact, finding lower bounds against circuit classes is a major theoretical task, with only a few striking success. One of them is due to Furst, Saxe and Sipser [41] and show that counting modulo 2 can not be done with circuits that can be evaluated in constant time in parallel. As for regular languages, Boolean circuits have a great interplay with formal logic, where the inherent non-uniformity of circuits is captured by having arbitrary numerical predicates. In particular, small depth circuits and extensions of first-order logic have numerous bindings. See [136] for an introduction to circuit complexity.

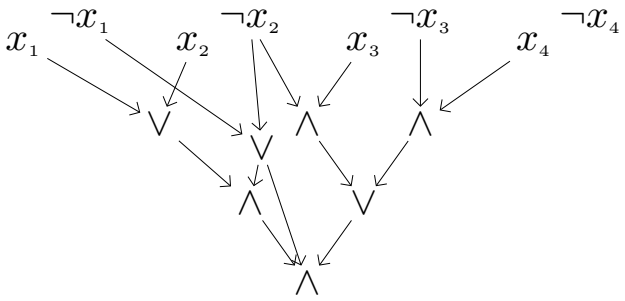


Figure 3.1: A first circuit

3.1 Boolean circuits

3.1.1 Definitions

A first Boolean circuit is given in Fig. 3.1.

We can see that there are three key ingredients in a circuit:

- several inputs labelled with variables and negated variables,
- many nodes labelled with Boolean functions (here conjunctions and disjunctions),
- one special node which is the output (usually drawn at the bottom).

Informally, this example is a recognition device for words over $\{0,1\}$ of size four. Given a word $u = u_1u_2u_3u_4$, each input with a variable takes the value of u_i, and each input with a negated variable takes the value $\neg u_i$. We then evaluate the circuit top-down thanks to the Boolean functions. The word is accepted if the output has value 1, and rejected otherwise. In this example, 0110 is accepted and 0101 is rejected.

We can formally describe what is a circuit. We recall that the fan-in (resp. fan-out) of a node of directed graph is the number of incoming (resp. outcoming) edges.

Definition 3.1.
A *circuit* with n inputs is a labelled directed acyclic graph (DAG for short) such that:

- there are exactly $2n$ nodes of fan-in 0. They are labelled by $x_1, \cdots, x_n$ and $\neg x_1, \cdots, \neg x_n$ and called the input gates,
- every other node is labelled with a Boolean function that takes as many entries as the fan-in of the node. They are called gates instead of nodes,
- there is exactly one node of fan-out 0 called the output gate.

The edges of the graph will be called *wires*.

Let C be a circuit with n inputs and $u \in \{0,1\}^*$ be a binary word of size n. The output of C on input u is defined inductively on the structure of the DAG. The output of an input gate labelled by x_i is 1 if $u_i = 1$ and 0 otherwise. The output of an input gate labelled by $\neg x_i$ is 1 if $u_i = 0$ and 0 otherwise. The output of a gate labelled by a Boolean function f, is $f(y_1, \ldots, y_k)$ where $y_1, \ldots, y_k$ are the outputs of the preceding gates. The output of the circuit is the output of the output gate. A word is accepted if the output is 1. The language $\mathcal{L}(C)$ of the circuit is the set of words of size n that are accepted by C.

Families of circuits. We want a notion of ressources needed by a circuit asymptotically. This is the reason why we consider families of circuits instead. A *family of circuits* is a sequence $(C_n)_{n\in\mathbb{N}}$ of circuits such that C_n has n input gates. The language recognised by the family is the union of all the languages of its circuits:

$$\mathcal{L}(\mathcal{C}) = \bigcup_n \mathcal{L}(C_n).$$

After some point, we will allow ourselves to stop making a distinction between circuits and families of circuits, and we will identify a class of circuits with their languages. We will often also consider circuits with several outputs.

Complexity of a family. We will be interested in languages recognised by "small" circuits. Therefore, we have to define what "small" means. There are several possible parameters we can look at. The *size* of a circuit is its number of gates. The *depth* of a circuit is the size of the longest path from an input gate to the output gate. The *fan-in* of a circuit is the maximal fan-in of the underlying DAG. We can extend all of these notions to families by looking at the function that associates to n the size (resp. depth, fan-in) of C_n.

Another notion of "simplicity" for a circuit is to look at the Boolean functions used in the gates.

Example 3.2.
They are several classical possibilities.

- $\wedge$-gates, labelled by a conjunction: the output is 1 if and only if all the inputs are 1.
- $\vee$-gates, labelled by a disjunction: the output is 1 if and only of one if the inputs is 1.
- MOD_q-gates, labelled by a counting function: the output is 1 if and only if the number of inputs that are 1 are divisible by q.
- MAJ-gates, labelled by a majority function: the output is 1 if and only if more than half of the inputs are 1.

More than two letters. So far, all the languages that can be recognised are over a binary alphabet. To extend circuits to languages over arbitrary alphabets, we adopt the so-called *one-hot encoding*. Let $A = \{a_1, \dots, a_k\}$ be an alphabet with k letters. A circuit for words of size n will have kn input gates labelled with $a_1(x_1), \dots, a_k(x_1), \cdots, a_1(x_n), \dots, a_k(x_n)$. Hence, each variable will have one input per possible letter. On input u, the output of a gate $a_j(x_i)$ will be 1 if and only if $u_i = a_j$.

3.1.2 Classes of small circuits

Recall that one of the reasons for the introduction of circuits is to have a combinatorial leverage on the $\mathsf{P} \neq \mathsf{NP}$ question. We call P/poly the class of languages defined by families of circuits whose size is growing *polynomially*. This name is not coincidental, this class also corresponds to the class of languages definable by a poynomial-time deterministic Turing machine with polynomial advice: for each size of words there exists a string of polynomial size that can be used for the computation. This gives immediately that:

$$\mathsf{P} \subseteq \mathsf{P/poly}.$$

Accordingly, exhibiting a language in NP that is not in P/poly would solve the P ≠ NP question for good.

Fact 3.3 (Shannon [118, Theorem 7])**.**
Almost all Boolean functions are not in P/poly.

Even if this result gives hope, it has to be relativised. First, it is proved via a counting argument. Therefore it does not give any specific example of a function not in P/poly. Indeed, we yet have to find such a function (the best known bounds are linear). Secondly, the "almost all" in the statement doesn't say anything about a relationship with NP.

Since we are so far to proving a lower bound against P/poly, it has been proposed to start with simpler circuits. For instance, by restricting the depth.

Definition 3.4.
We define the following classes of circuit families only using $\wedge$-gates and $\vee$-gates:

- AC^0 has all the circuits with the size bounded by a polynomial and the depth bounded by a constant,
- NC^0 has all the circuits with both the fan-in and the depth bounded by a constant,
- NC^1 has all the circuits with the size bounded by a polynomial, and the depth bounded by $O(\log(n))$.

The class NC^0 can only express a few languages: there are only a bounded number of inputs that are connected to the output gate. In particular, their size is bounded by a constant. It is possible to find an equivalent circuit of depth only 2 that stays in NC^0. It is worthless to restrict only the depth in AC^0 and NC^1, as they would be able to express every language. Indeed, we can put any Boolean function into conjunctive normal form (a conjunction of disjunctions) and therefore compute it with a circuit of depth 2, but with an exponential size. We have the containment $\mathsf{AC}^0 \subseteq \mathsf{NC}^1$. To see it, we can replace every unbounded fan-in $\wedge$-gate and $\vee$-gate in an AC^0 circuit by a balanced binary tree of $\wedge$-gates of fan-in 2. With this procedure, the size only grows by a polynomial and the depth is bounded by a logarithm. This ensures that the obtained circuit is in NC^1.

Adding more gates. We mentioned in the previous subsection a couple of other types of gates: MOD-gates and MAJ-gates. We can add them to our circuits, assuming that we have their computational power for free.

Definition 3.5.
The following three classes of circuits are defined exactly like AC^0, but with extra gates:

- $\mathsf{ACC}^0[p]$ has MOD_p-gates for a prime integer p,
- ACC^0 has MOD_q-gates for every integer q,
- TC^0 has MAJ-gates.

The ACC^0 circuits without $\wedge$-gates or $\vee$-gates form the class CC^0. For X a set of integers, we will also write $\mathsf{ACC}^0[X]$ for the AC^0 circuits that can use MOD_q gates with $Q \in X$. These classes

are obviously more expressive than AC^0. We know how they relate between each other.

Fact 3.6.
The function computed by a MAJ-gate with n inputs can be computed with a circuit with only fan-in 2 $\wedge$-gates and $\vee$-gates, polynomial size and logarithmic depth.

Proof sketch. First we have to accept that we can perform the simultaneous addition of n integers encoded over n bits, with a NC^1 circuit (with $n + \log(n)$ outputs). The addition of two n bits numbers is in AC^0, hence the naive circuits with a binary tree over the n words and a binary tree for each unbounded fan-in gate would give a depth of $O(\log(n)^2)$. A clever trick that allows to only perform partial intermediate addition gives the desired depth of $O(\log(n))$. Then to compute a MAJ-gate, we consider the inputs are encoded over n bits (by plugging $n-1$ 0s in front of them), and we add them all together. We end up with the binary representation of the sum of the inputs. We can check that it is smaller than the binary representation of $\frac{n}{2}$ with a simple AC^0 circuit. □

Fact 3.7.
Let q be an integer. The function computed by a MOD_q-gate with n inputs can be computed with a circuit with $\wedge$-gates, $\vee$-gates and MAJ-gates, size $3\left\lfloor \frac{n}{q} \right\rfloor + 4$ and depth 3.

Proof. For $0 \leq i \leq n$ an integer, let THR_i be the circuit with n inputs that consists in one single MAJ-gate with $2n$ inputs that copy the inputs of THR_i and plugs 1s in $n-i$ extra inputs and 0s or the remaining i extra inputs. The circuit THR_i outputs 1 if and only if there are more than i 1s in the input.

We can now define a circuit EQ_i that outputs 1 if and only if there are precisely i 1s in its inputs:

$$\mathrm{EQ}_i = \mathrm{THR}_i \wedge \neg \mathrm{THR}_{i+1}.$$

The circuit for a MOD_q gate just checks whether there is $0, q, 2 \times q, \cdots$, or $\left\lfloor \frac{n}{q} \right\rfloor \times q$ 1s in the input:

$$\mathrm{MOD}_q = \bigvee_{i=0}^{\left\lfloor \frac{n}{q} \right\rfloor} \mathrm{EQ}_{i \times q}.$$

□

Fine-grained complexity. Another focus of circuit complexity is to look at subclasses of AC^0 to obtain a very precise classification of problems. The class AC^0 consists of constant depth circuits. It is natural to look at the hierarchy of circuits with depth d, when d is fixed. Note that in the definition we give here, there is an extra NC^0 layer near the inputs. It is a foreshadowing of the links with logical formalisms we will see in the next section.

Definition 3.8.
The classes Σ_0 and Π_0 are defined as NC^0. For an integer $i \geq 1$, a circuit is in Σ_i if its output gate is a $\vee$-gate that is fed by a bunch of Π_{i-1} circuits. Dually, a circuit is in Π_i if its output

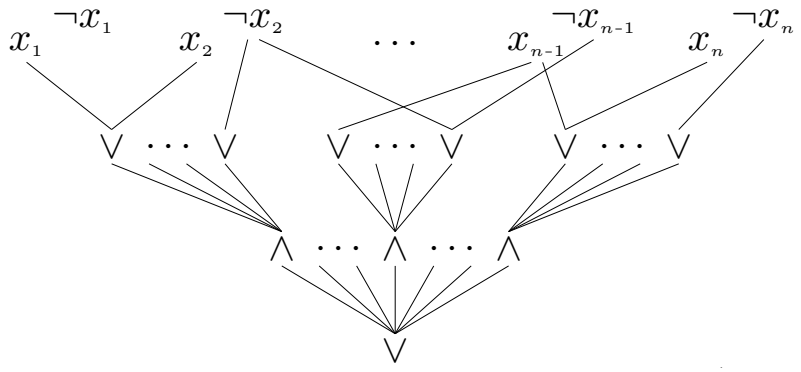


Figure 3.2: A Σ_2 circuit

gate is a $\wedge$-gate that is fed by a bunch of Σ_{i-1} circuits.

We moreover define, for every i and j, the classes Σ_i^j and Π_i^j analogously to Σ_i and Π_i but with the first layer restricted to NC^0-circuits of depth 2 and fan-in bounded by j. We have that $\Sigma_i = \bigcup_j \Sigma_i^j$. For instance, a Σ_2 circuit is of polynomial size, depth 3 and consists of an output $\vee$-gate with $\wedge$-gates as inputs, each of these having $\wedge$-gates as inputs, each of these having a bounded number of variables as inputs. Such a circuit looks like Fig. 3.2.

There is one last notion of circuit complexity to define. This approach is different than the previous one, this time we will impose stronger conditions on the size of the circuit. For such precise a complexity measure, it makes sense to distinguish between the two notions of size for a graph: the number of vertices and the number of edges.

Definition 3.9.
The subclass of AC^0 of circuits with only a linear number of gates is called LAC^0. When it is the number of wires that is linear, it is called WLAC^0 instead.

Because a connected graph has at least as many edges than nodes, we have that WLAC^0 is weaker than LAC^0.

The relations among the small circuit classes mentioned so far are summarised in the following diagram:

$$\begin{array}{ccccccccccccc} & & \mathsf{WLAC}^0 & \subseteq & \mathsf{LAC}^0 & & & & & & & & \\ & \subsetneq & & & & \subsetneq & & & & & & & \\ \mathsf{NC}^0 & & & & & & \mathsf{AC}^0 & \subseteq & \mathsf{ACC}^0[p] & \subseteq & \mathsf{ACC}^0 & \subseteq & \mathsf{TC}^0 & \subseteq & \mathsf{NC}^1 \\ & \subsetneq & \Sigma_1 & \subseteq & \Sigma_2 & \subseteq \cdots \subseteq & \Sigma_i & & & & & & & & \end{array}$$

3.2 Adding arbitrary predicates to the logic

We have seen that automata and logic share deep connections. There exists a similar interplay between circuits and logic. However, circuit families are a *non-uniform* model of computation: the circuits for each size must not satisfy any particular relation. For instance, with an enumeration of Turing machines, we can construct a family (C_n) with C_n is the constant 1 if the n^{th} Turing

machine halts and 0 otherwise. This family even belongs to NC^0. To match such a behaviour, we need to use the power of arbitrary numerical predicates. There are many results of equivalences between circuits and logical classes.

Theorem 3.10 (Gurevitch and Lewis [55], Immerman [63])**.**
It stands that

$$\mathsf{AC}^0 = \mathrm{FO}[\mathrm{arb}].$$

This result can be adapted for many subclasses of AC^0.

Lemma 3.11 (Barrington, Compton, Straubing and Thérien [14])**.**
With X any set of integers,

$$\mathsf{ACC}^0 = (\mathrm{FO} + \mathrm{MOD})[\mathrm{arb}],$$

$$\mathsf{ACC}^0[X] = (\mathrm{FO} + \mathrm{MOD}[X])[\mathrm{arb}].$$

Lemma 3.12 (Koucký, Lautemann, Poloczek and Thérien [71])**.**
It stands that:

$$\mathsf{LAC}^0 = \mathrm{FO}^2[\mathrm{arb}].$$

Lemma 3.13 (Maciel, Péladeau and Thérien [78])**.**
For i an integer, the classes on the left are circuit classes,

$$\Sigma_i = \Sigma_i[\mathrm{arb}],$$

$$\Pi_i = \Pi_i[\mathrm{arb}].$$

Note that this implies in particular that NC^0 corresponds to the class of languages definable by a quantifier-free formula with arbitrary numerical predicates. There is a similar characterisation of TC^0 with some ad-hoc quantifiers that we are not showing here. All the proofs of these theorems are along the same line. The translation from formulae translates existential quantifiers to an unbounded fan-in $\vee$-gates, universal quantifiers to an unbounded fan-in $\wedge$-gates, disjunction to fan-in 2 $\vee$-gates, conjunction to fan-in 2 $\wedge$-gates. For the other way around, we can encode the structure of the circuits into the arbitrary predicates.

Lindström quantifiers Like the previous logical characterisation, there are circuit equivalents to logical fragments with any Lindström quantifiers. For $\mathcal{C}$ a class of regular languages, we denote by $\mathsf{AC}^0[\mathcal{C}]$ the class of circuits of polynomial size, bounded depth and bounded fan-in that can only use gates labelled by regular languages in $\mathcal{C}$. We have a general theorem for the usage of Linström quantifiers.

Theorem 3.14 (Barrington, Immerman, and Straubing[12, Theorem 9.1])**.**
Let $\mathcal{C}$ be a class of regular languages. It stands that

$$\mathsf{AC}^0[\mathcal{C}] = \mathrm{Lin}(\mathcal{C})[\mathrm{arb}].$$

Proof. The theorem referenced above is about a uniform version of both classes. It can be easily adapted to match the non uniform version with arbitrary predicates. □

3.3 Lower bounds

A tremendous line of research consists in finding lower bounds against classes of circuits, that is to say exhibiting a language and showing that it cannot be computed by a given type of circuit. For instance, a lower bound in NP against P/poly would settle the $\mathsf{P} \neq \mathsf{NP}$ question. So far, the best result in this regard is due to Ryan Williams.

Theorem 3.15 (Williams [138])**.**
There exists a language in NEXP that cannot be computed by an ACC^0 circuit:

$$\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0.$$

This result from the state of the art should be contrasted one might like: NEXP is much larger than NP, and ACC^0 is much smaller than P/poly.

Another reason to look at lower bounds is to show that two classes of circuits are distinct. For instance, was it necessary to introduce MOD-gates, or can they be simulated inside of AC^0? We will present here the main techniques known to prove inexpressibility results for circuits.

3.3.1 The parity language

One of the first influential lower bounds against a circuit class was against AC^0. It is often thought of as one of the greatest results in the field. The chosen language is very simple.

Definition 3.16.
The PARITY language is the language over $\{0,1\}$ of words with an even number of 1s. A regular expression for it is $(0^*10^*10^*)^*$.

The lower bound can now be stated.

Theorem 3.17.
We have:

$$\mathrm{PARITY} \notin \mathsf{AC}^0.$$

There exist many proofs of this theorem. Ajtai [1] used model theory. Furst, Saxe and Sipser [41] worked directly with the combinatorial structure of the circuits. They introduced the

"random restriction" method. It works by showing, with a probabilistic method, that it is possible to set a small fraction of the inputs of a circuit to reduce its depth by one. All is left to do is to easily show that every depth 2 circuit computing PARITY has size 2^n.

One last line of proof is the algebraic method of Razborov [108]. It works by showing that any function computed by an AC^0 circuit can be accurately approximated by a low-degree polynomial. To conclude, he shows that the parity function has no such approximation.

The later techniques have later been extended by Smolenski [123] to show a stronger result.

Lemma 3.18.
Let q be an integer and let MOD_q be the language of words with a number of 1s divisible by q. With X the set of integers that are prime with q, we have that

$$\mathrm{MOD}_q \notin \mathsf{ACC}^0[X].$$

Those results can be used to show that the inclusion between some classes of circuits are strict: for any prime p

$$\mathsf{AC}^0 \subsetneq \mathsf{ACC}^0[p] \subsetneq \mathsf{ACC}^0.$$

3.3.2 Depth hierarchy

There also are lower bounds against the fine structure of AC^0, that is to say against the classes we denoted by Σ_i and Π_i. Let d be a depth. The Sipser languages $\mathcal{L}_{d,i}$ are defined for a number of inputs $i = j^d$ for some integer j, over a binary alphabet. We will consider that we are encoding the positions in the word with a vector of $\{0,\dots,m-1\}^d$. We define the arbitrary predicate that only has tuples with $0 \leq l < i$ and $0 \leq k_1,\dots,k_d < j$ such that

$$\mathrm{Encode}(l,k_1,\dots,k_d) \text{ iff } l = k_1 + mk_2 + \cdots + m^{d-1}k_d.$$

The languages $\mathcal{L}_{d,i}$ are defined by the formulae

$$\exists k_1, \forall k_2, \dots, \exists k_{d-1}, \forall k_d, \forall l, \mathrm{Encode}(l,k_1,\dots,k_d) \Rightarrow \mathbf{a}(l)$$

for even ds and

$$\exists k_1, \forall k_2, \dots, \forall k_{d-1}, \exists k_d, \exists l, \mathrm{Encode}(l,k_1,\dots,k_d) \wedge \mathbf{a}(l)$$

for odd ds.

They are in $\Sigma_i[\mathrm{arb}]$ and hence in the circuit class Σ_i. The idea is that every gate is of fan-in j and the gates of a same layer partition the set of inputs.

Sipser [121] showed that they need superpolynomial size for Σ_{i-1}-circuit, and later Håstad in his PhD thesis [59] gave an exponential lower bound.

Theorem 3.19 (Sipser [121], Håstad [59])**.**
The language $\mathcal{L}_{d,n}$ is in Σ_i but not in Σ_{i-1}.

There is an obvious analogue for the hierarchy Π_i.

3.3.3 Superconcentrators

Another angle of study is to look at the underlying graph of a circuit, and show that it must satisfy high connectivity properties. For instance, Valiant [134, 133] proposed to use the so-called *superconcentrators* to that end.

Definition 3.20.
A n-superconcentrator is a DAG with n nodes of fan-in 0 and n nodes of fan-out 0. For every $1 \leq k \leq n$ and subset A of the fan-in 0 nodes and subset B of the fan-out 0 nodes both of size k, there must exist k vertex-disjoint paths from A to B.

It can be used to show lower bounds for many problems arising from algebra. For instance, consider the problem of multiplying two Boolean polynomials of degree $n-1$. The $2n$ inputs are the coefficients of both polynomials, and the output is the $2n$ coefficients of the product. It can be shown [134] that any Boolean circuit for the polynomial multiplication is a n-superconcentrator. Hence it remains to prove lower bounds on the size of an n-superconcentrator. However, the quest for a general lower bound ran short because of a result of Valiant, later improved by Pippinger.

Fact 3.21 (Valiant [133], Pippinger [102])**.**
There exists a superconcentrator with linear number of edges, and a logarithmic depth.

Nevertheless, a superlinear lower bound has been proved for circuits of bounded depth. We need to define the usual hierarchy of slowly growing functions from $\mathbb{N}$ to $\mathbb{N}$. Let f_0 be the identity function, multiplied by 2. For $i \leq 1$, we define the i^{th} to be, for $n \in \mathbb{N}$,

$$f_i(n) = f_{i-1}(f_{i-1}(\cdots f_{i-1}(2)))$$

where f_{i-1} is applied n times. These functions are rapidly growing, we need to use their inverses

$$f_i^{-1}(n) = \min\{m \mid f_i(m) \geq n\}.$$

Theorem 3.22 (Dolev, Dwork, Pippinger and Widgerson [32], Pudlák [106])**.**
Let $k \geq 4$, every n-superconcentrator of depth k must have at least $O(nf^{-1}_{\lfloor k/2 \rfloor}(n))$ edges.

The remaining cases of depth 2 and 3 have been settled by Alon and Pudlák [4]. This theorem allows to show that problems such as polynomial multiplication are not in WLAC^0. Koucky, Pudlák and Thérien [72, Theorem 1] used these methods to exhibit a language that is in LAC^0 but not in WLAC^0 (even with modular gates). It should be mentioned that it is also possible to separate LAC^0 from AC^0, with the k-clique language [111, Theorem 1.2], with completely different techniques.

3.3.4 Discriminator lemma

There are only a few tools to prove lower bounds against AC^0. Hence it is even more difficult to fight against circuits with more type of gates, like TC^0. One of the few techniques available in this case is due to Hajnal and al. However, it was only successfully applied to TC^0 circuits that are very shallow (of depth 2 or 3), or with a limited number of MAJ-gates (bounded by a logarithm). For instance, we have shown in Fact 3.7 that PARITY is in TC^0, but it is known that it requires more than a logarithmic number of MAJ-gates [104]. It can be extended to any MOD_q function, though the precise bound is less precise.

Theorem 3.23 (Gopalan, Servedio [47, Theorem 26])**.**
The function MOD_q cannot be computed by an AC^0 circuit even with $O(\log(n))$ MAJ-gates.

Definition 3.24.
Let C be a circuit with n inputs, A and B two disjoint subsets of $\{0,1\}^n$, and $\varepsilon > 0$. We consider $\mathbb{P}_A$ and $\mathbb{P}_B$ the uniform probability distributions on A and B. We say that C is an ε-discriminator for A and B if

$$|\mathbb{P}_A(C=1) - \mathbb{P}_B(C=1)| \geq \varepsilon.$$

The discriminator lemma states that a MAJ-gate has to be correlated with one of its inputs.

Lemma 3.25 (Hajnal, Maass, Pudlák, Szegedy, Turán [56])**.**
Let C be a circuit with a MAJ-gate for output that is fed by $C_1, \ldots, C_m$. Let $A, B \subseteq \{0,1\}^n$ a subset of accepted and rejected words for C. Then one of the C_i is an $\frac{1}{m}$-discriminator for A and B.

To illustrate this tool, we will consider the equality function EQ that outputs 1 if and only if there are precisely half of the inputs with value 1. In particular, it always outputs 0 on odd inputs. We have seen in the proof of Fact 3.7 that this function can be computed with a TC^0 circuit which is an $\wedge$-gate of two MAJ-gates. There exists a trick, due to Arka Gosh, that allows to reduce the number of MAJ-gates needed.

We say that a circuit is in Majority Normal Form (MNF for short) if it is of depth 2 with a single MAJ-gate, as the output.

Property 3.26.
The equality function EQ is computable by a MNF of quadratic size.

Proof. Let n be an even input size. We assume we can do a $\mathrm{THR}_{n^2/4}$ circuit as in the proof of Fact 3.7, that outputs 1 whenever there are more than $\frac{n^2}{4}$ 1s in its input. The circuit for EQ is then

$$\mathrm{THR}_{n^2/4}(x_i \wedge \neg x_j \quad \text{for } 1 \leq i, j \leq n).$$

This circuit is indeed of quadratic size, even when replacing THR by a MAJ-gate only a quadratic number of constants are added. Let w be a word. It is easy to see that the number of subcircuits of THR that is evaluated to 1 is precisely $|w|_0 \times |w|_1$. But this value, when constrained with $|w|_0 + |w|_1 = n$ is known to be always less or equal to $\frac{n^2}{4}$, with this value reached if and only if $|w|_0 = |w|_1 = \frac{n}{2}$. Therefore the whole circuit evaluates to 1 if and only if precisely half of the inputs are 1. □

The goal is to show that it is essentially optimal, that is to say that there is no MNF of linear size that computes the EQ function. Thanks to the discriminator lemma, it is enough to show that for carefully selected A and B, EQ is poorly correlated by any $\wedge$-gate (resp. $\vee$-gate).

Recall that the (k,n) binomial coefficient, denoted $\binom{n}{k}$, is the number of way of choosing k elements in a set of size n. It satisfies many relations that we list here. For all $0 \leq k < n$:

i) $\binom{n}{k} = \frac{n!}{k!(n-k)!}$,
ii) $\binom{n}{k} = \binom{n}{n-k}$,
iii) $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$,
iv) $\binom{n}{k} \leq 2^n$.

Let C be an MNF over n inputs computing EQ. Let $k = \frac{n}{2}$ and A the set of words with k 1s and B the set of word with $k-1$ or $k+1$ 1s. We have that C accepts A and rejects B. Let α be one of the subcircuits of C. We assume that it is a $\wedge$-gate of variables, the other case being dual. We assume that it is bound to a non negated inputs and b negated inputs. We have to evaluate and bound by above the quantity:

$$\Delta = |\mathbb{P}_A(\alpha = 1) - \mathbb{P}_B(\alpha = 1)|.$$

We can assume that $a, b \leq k$, otherwise the probabilities become null. First of all, the size of A is $\binom{n}{k}$. The words such that α evaluates to A are the ones with a 1 in the position of the non negated inputs and a 0 in the position of the negated inputs. Hence to obtain such a word that falls in A, we have to choose the $k-a$ positions in the remaining $n-a-b$ positions that will carry a 1. Therefore,

$$\begin{aligned}\mathbb{P}_A(\alpha = 1) &= \frac{\binom{n-a-b}{k-a}}{\binom{n}{k}} = \frac{\frac{(n-a-b)!}{(k-a)!(n-k-b)!}}{\frac{n!}{k!(n-k)!}} = \frac{\frac{(n-a-b)!}{(k-a)!(k-b)!}}{\frac{n!}{k!^2}} \\ &= \frac{k \cdots (k-a+1) \cdot k \cdots (k-b+1)}{n \cdots (n-a-b+1)}.\end{aligned}$$

Similarly, the size of B is $\binom{n}{k+1} + \binom{n}{k-1} = 2\binom{n}{k+1}$ thanks to ii) and the fact that $k = n-k$. We will often use this last fact silently. Therefore,

$$\begin{aligned}\mathbb{P}_B(\alpha = 1) &= \frac{\binom{n-a-b}{k-a-1} + \binom{n-a-b}{k-a+1}}{2\binom{n}{k+1}} = \frac{\frac{(n-a-b)!}{(k-a-1)!(k-b+1)!} + \frac{(n-a-b)!}{(k-a+1)!(k-b-1)!}}{2\frac{n!}{(k+1)!(k-1)!}} \\ &= \frac{(k+1)\cdots(k-b+2)\cdot(k-1)\cdots(k-a) + (k+1)\cdots(k-a+2)\cdot(k-1)\cdots(k-b)}{2n\cdots(n-a-b+1)} \\ &= \frac{(k+1)\cdot k\cdot(k-1)\cdots(k-b)\cdot(k-1)\cdots(k-a)}{2n\cdots(n-a-b+1)}\left(\frac{1}{(k-b+1)(k-b)} + \frac{1}{(k-a+1)(k-a)}\right).\end{aligned}$$

Putting those two bits of computation together:

$$\begin{aligned}
&\mathbb{P}_A(\alpha=1)-\mathbb{P}_B(\alpha=1)\\
&\qquad=\mathbb{P}_A(\alpha=1)\left[1-\frac{(k+1)(k-a)(k-b)}{2k}\left(\frac{1}{(k-b+1)(k-b)}+\frac{1}{(k-a+1)(k-a)}\right)\right]\\
&\qquad=\mathbb{P}_A(\alpha=1)\left[1-\frac{(k+1)(k-a)}{2k(k-b+1)}-\frac{(k+1)(k-b)}{2k(k-a+1)}\right]\\
&\qquad=\mathbb{P}_A(\alpha=1)\left[\frac{\delta}{2k(k-a+1)(k-b+1)}\right]
\end{aligned}$$

where:

$$\delta=2k(k-a+1)(k-b+1)-(k+1)(k-a+1)(k-a)-(k+1)(k-b+1)(k-b).$$

The key observation is that δ is a polynomial in k. A quick glance immediately gives that it is a degree 3, but it is actually of degree 1. To see that we develop the polynomial with the function "coeff" of Xcas:

$$\delta=(2ab+a+b-a^2-b^2)k+(a+b-a^2-b^2).$$

There are two regimes depending of if a and b are small or not. We only need an estimate on $\binom{n}{k}$, which comes from the fact that it is the largest i,n binomial coefficient for $0\leq i\leq n$:

$$\binom{n}{k}\geq\frac{2^n}{n+1}.$$

First case. Assume that $a\geq\log(n)^2$ or $b\geq\log(n)^2$. The values k,a and b are bounded above by n, so we can ruthlessly bound the right part of Δ:

$$\left|\frac{\delta}{2k(k-a+1)(k-b+1)}\right|\leq 10n^3.$$

Now we need to bound $\mathbb{P}_A(\alpha=1)$ using item (iv) and the bound on central coefficients:

$$\mathbb{P}_A(\alpha=1)\leq\frac{\binom{n-a-b}{k-a}}{\binom{n}{k}}\leq 2^{n-a-b}\frac{n+1}{2^n}\leq\frac{n+1}{2^{\log(n)^2}}.$$

In the end

$$\Delta\leq\frac{12n^3}{n^{\log(n)}}=o(\frac{1}{n}).$$

Second case. Assume that both $a\leq\log(n)^2$ and $b\leq\log(n)^2$. We bound $\mathbb{P}_A(\alpha=1)$ by 1 and hence

$$\Delta\leq\frac{10n\log(n)^4}{2k(k-a+1)(k-b+1)}\leq\frac{10n\log(n)^4}{n(\frac{n}{2}-\log(n)^2)^2}\sim 40\frac{\log(n)^4}{n^2}=o(\frac{1}{n}).$$

We can conclude.

Theorem 3.27.
The equality function EQ cannot be computed by a MNF of linear size.

Proof. Let (C_n) be a MNF such that C_n has less than ln gates. By Lemma 3.25, one of the bottom $\wedge$-gates or $\vee$-gates α must satisfy $\Delta = |\mathbb{P}_A(\alpha = 1) - \mathbb{P}_B(\alpha = 1)| \geq \frac{1}{ln}$. However, by the previous computations, there is an n big enough such that every such Δ is strictly smaller than $\frac{1}{ln}$. This is a contradiction. □

Bibliography of the current chapter

[1] M. Ajtai. "Σ_1^1-Formulae on finite structures". en. In: *Annals of Pure and Applied Logic* 24.1 (July 1983). DOI: 10.1016/0168-0072(83)90038-6.

[4] Noga Alon and Pavel Pudlak. "Superconcentrators of depths 2 and 3; odd levels help (rarely)". In: *Journal of Computer and System Sciences* 48.1 (Feb. 1994). DOI: 10.1016/S0022-0000(05)80027-3.

[12] David Barrington, Neil Immerman, and Howard Straubing. "On uniformity within NC_1". In: July 1988. DOI: 10.1109/SCT.1988.5262.

[14] David A. Barrington, Kevin Compton, Howard Straubing, and Denis Thérien. "Regular languages in NC_1". In: *Journal of Computer and System Sciences* 44.3 (1992). DOI: 10.1016/0022-0000(92)90014-A.

[32] Danny Dolev, Cynthia Dwork, Nicholas Pippenger, and Avi Wigderson. "Superconcentrators, Generalizers and Generalized Connectors with Limited Depth (Preliminary Version)". In: Jan. 1983. DOI: 10.1145/800061.808731.

[41] Merrick Furst, James B Saxe, and Michael Sipser. "Parity, circuits, and the polynomial-time hierarchy". en. In: (1984). DOI: 10.1007/BF01744431.

[47] Parikshit Gopalan and Rocco Servedio. *Learning and Lower Bounds for AC^0 with Threshold Gates*. en. Tech. rep. TR10-074. Electronic Colloquium on Computational Complexity (ECCC), Apr. 2010. DOI: 10.1007/978-3-642-15369-3_44.

[55] Yuri Gurevich and Harry R. Lewis. "A logic for constant-depth circuits". In: *Information and Control* 61.1 (1984). DOI: 10.1016/S0019-9958(84)80062-5.

[56] András Hajnal, Wolfgang Maass, Pavel Pudlák, Mário Szegedy, and György Turán. "Threshold circuits of bounded depth". en. In: *Journal of Computer and System Sciences* 46.2 (Apr. 1993). DOI: 10.1016/0022-0000(93)90001-D.

[59] Johan Håstad. "Computational limitations for small depth circuits". en. Thesis. Massachusetts Institute of Technology, 1986.

[63] Neil Immerman. "Languages that Capture Complexity Classes". In: *SIAM Journal on Computing* 16.4 (1987). DOI: 10.1137/0216051.

[71] M. Koucky, S. Poloczek, C. Lautemann, and Denis Therien. "Circuit lower bounds via Ehrenfeucht-Fraisse games". In: vol. 2006. Jan. 2006. DOI: 10.1109/CCC.2006.12.

[72] Michal Koucký, Pavel Pudlak, and Denis Therien. "Bounded-depth circuits: Separating wires from gates". In: May 2005. DOI: 10.1145/1060590.1060629.

[78] Alexis Maciel, Pierre Péladeau, and Denis Thérien. "Programs over semigroups of dot-depth one". In: *Theoretical Computer Science* 245.1 (2000). DOI: 10.1016/S0304-3975(99)00278-9.

[102] Nicholas Pippenger. "Superconcentrators". In: *All HMC Faculty Publications and Research* (Jan. 1977). DOI: 10.1137/0206022.

[104] Vladimir V. Podolskii. "Exponential lower bound for bounded depth circuits with few threshold gates". en. In: *Information Processing Letters* 112.7 (Mar. 2012). DOI: 10.1016/j.ipl.2011.12.011.

[106] P. Pudlák. "Communication in bounded depth circuits". en. In: *Combinatorica* 14.2 (June 1994). DOI: 10.1007/BF01215351.

[108] A. A. Razborov. "Lower bounds on the size of bounded depth circuits over a complete basis with logical addition". en. In: *Mathematical notes of the Academy of Sciences of the USSR* 41.4 (Apr. 1987). DOI: 10.1007/BF01137685.

[111] Benjamin Rossman. "On the constant-depth complexity of k-clique". In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. STOC '08. Victoria, British Columbia, Canada: Association for Computing Machinery, 2008. DOI: 10.1145/1374376.1374480.

[117] Claude E. Shannon. "A symbolic analysis of relay and switching circuits". In: *Transactions of the American Institute of Electrical Engineers* 57.12 (1938). DOI: 10.1109/T-AIEE.1938.5057767.

[118] Claude. E. Shannon. "The synthesis of two-terminal switching circuits". In: *The Bell System Technical Journal* 28.1 (1949). DOI: 10.1002/j.1538-7305.1949.tb03624.x.

[121] Michael Sipser. "Borel sets and circuit complexity". In: *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*. STOC '83. New York, NY, USA: Association for Computing Machinery, 1983. DOI: 10.1145/800061.808733.

[123] R. Smolensky. "Algebraic methods in the theory of lower bounds for Boolean circuit complexity". en. In: *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*. New York, New York, United States: ACM Press, 1987. DOI: 10.1145/28395.28404.

[133] Leslie G. Valiant. "Graph-theoretic arguments in low-level complexity". en. In: *Mathematical Foundations of Computer Science 1977*. Ed. by Jozef Gruska. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1977. DOI: 10.1007/3-540-08353-7_135.

[134] Leslie G. Valiant. "On non-linear lower bounds in computational complexity". en. In: *Proceedings of seventh annual ACM symposium on Theory of computing - STOC '75*. Albuquerque, New Mexico, United States: ACM Press, 1975. DOI: 10.1145/800116.803752.

[136] Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer Berlin Heidelberg, 1999. DOI: 10.1007/978-3-662-03927-4.

[138] Ryan Williams. "Non-uniform ACC Circuit Lower Bounds". In: *2011 IEEE 26th Annual Conference on Computational Complexity*. 2011. DOI: 10.1109/CCC.2011.36.

Chapter 4

Regular Languages and Circuit Classes

Outline of the current chapter

We have seen that regular languages, one of the simplest object in computer science, correspond to a paradigm of sequential computation and have a wide range of applications. It is therefore natural to challenge complexity classes by trying to identify the regular languages they are able to compute, with the hope to grab understanding of the class in the process. This approach obviously only works for small complexity classes: whenever the class is expressive enough to compute all regular languages, like for instance any class greater than P, this line of research becomes irrelevant. It has already be done for some models of computation already. For instance, Amarilli, Jachiet and Paperman studied the space complexity of maintaining regular languages under updates in the RAM model. This gave a powerful trichotomy of time complexity. We continue this study for regular languages of trees in Chapter 7. In this thesis, we will also consider computation in streaming Chapter 6, and incremental maintenance by first-order updates on relational databases Chapter 7. However, the most dramatic success of this line of work is when looking at circuit classes, building a bridge between sequential and parallel computations.

When studying circuit complexity, regular languages appeared to sensationally capture the essence of the complexity of many classes included in NC^1. Indeed, we will see that NC^1 can compute all regular languages. As said before, looking at regular languages is only useful for

small complexity classes, because too powerful classes are all equally expressive when restricted to regular languages. In this chapter, we advocate that studying regular languages have to be studied to understand circuit classes. This point of view is developed in Howard Straubing's book "Finite automata, formal logic and circuit complexity" [126]. For instance, when two classes of circuits are different, it is often the case that this separation is witnessed by a regular language. It is the case for the famous lower bound of Furst, Saxe and Sipser [41] that PARITY is not in AC^0, showing that the sepration between AC^0 and ACC^0 is witnessed by a regular language. Moreover, we will see that there are regular languages that are complete for some classes of circuits, notably giving an unexpected link between regular languages and the class NC^1. This motivates a systematic identification of regular languages inside given circuit classes.

In Chapter 3, we have presented links between circuit classes and logical formalism with arbitrary numerical predicates. In his book [126], Straubing conjectured a logical characterisation of the regular languages in circuits classes. For F a logical fragment, the regular languages of F[arb] are exactly the languages of F[reg]. In other words, if a formula with arbitrary predicates expresses a regular languages, then we can rewrite the formula so that it only uses regular predicates. We give some fragments for which the conjecture is true, and some for which it is false.

We start by showing that all regular languages are in NC^1, which justify that we are only interested in small depth circuits. To compute a regular language, we can adopt a divide and conquer algorithm, and simulate in parallel the executions of every portion of a word thanks to the syntactic monoid.

Theorem 4.1.
Every regular language can be computed by a NC^1 circuit. In symbols

$$\mathrm{Reg} \subseteq \mathsf{NC}^1.$$

Proof. Let $\mathcal{L}$ be a regular language and $M_{\mathcal{L}}$ its syntactic monoid and $\mu_{\mathcal{L}}$ its syntactic morphism. Let also P be a subset of $M_{\mathcal{L}}$ that recognises $\mathcal{L}$. We denote by m the size of the monoid. We will encode an element of $M_{\mathcal{L}}$ by a sequence of m bits, with only one of them set to 1. We need a few gadgets.

- A circuit that takes as inputs the encoding of a letter a and outputs the encoding of $\mu_{\mathcal{L}}(a)$,
- a circuit that takes as inputs the encoding of two monoid elements x and y and outputs the encoding of $x \cdot y$,
- a circuit that takes as input a monoid element and outputs a bit that indicates if it belongs to P or not.

We will respectively refer to them as input (resp. multiplication, output) gadgets. All those circuits can be assumed of fan-in 2, constant size and constant depth. Indeed, we can take the conjunctive normal form for them, and expand the unbounded $\wedge$-gates and $\vee$-gates. Their size will be exponential but in the number of inputs, which is a constant bounded by 2^{2m}.

We can now construct a circuit that recognises $\mathcal{L}$. Under each input, we put an input gadget. Then we have to multiply all these monoid elements together to obtain the image of the word in the monoid. To do it efficiently, we adopt a divide and conquer approach and perform the multiplication gadgets along a balanced binary tree whose leafs are the input gadgets. This heavily uses the fact that a monoid operation is associative. In the end, we use the output gadget once to know if the image of the word under $\mu_{\mathcal{L}}$ is in P. We use a linear number of

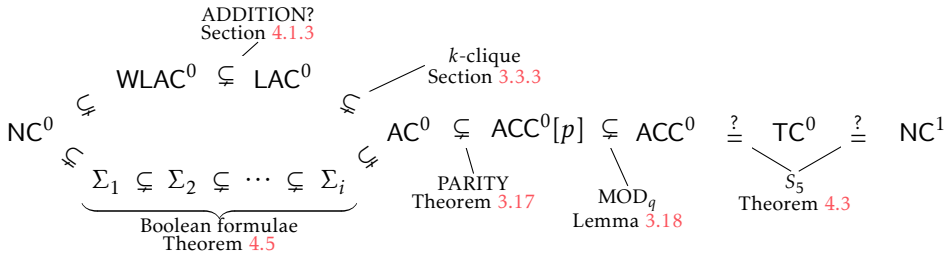


Figure 4.1: Known relations

gadgets in total, each of them being of constant size and depth. This gives the desired bound of a polynomial size and logarithmic depth. □

4.1 Separations witnessed by regular languages

A strong evidence regarding the importance of regular languages in circuit classes is that almost all the separations we know in the important classes defined in this document are witnessed by regular languages. It is particularly striking that such powerful and non-uniform classes are faithfully represented by the highly uniform regular languages. We have already seen with Theorem 3.17 that $\mathsf{AC}^0 \subsetneq \mathsf{ACC}^0[2]$ is witnessed by the PARITY language which is regular. Its extension Lemma 3.18 furthermore gives that $\mathsf{ACC}^0[p] \subsetneq \mathsf{ACC}^0$ is witnessed by any language MOD_q for an integer q coprime with p, which are regular as well. This section is aimed at exposing the results to have the global picture given in Fig. 4.1.

4.1.1 Barrington's theorem.

Regarding the chain of inclusion $\mathsf{ACC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1$, it is not known whether any equality holds. However, if any of ACC^0 or TC^0 is strictly included in NC^1, then it is witnessed by a regular language. It comes from the celebrated and unexpected Barrington's theorem, first exhibited in [13]. It was stated in terms of bounded-width branching programs, but we give here a formulation as the completeness of certain regular languages for NC^1, as exposed in [14].

When speaking of completeness, one has to specify the type of reductions considered. For Barrington's theorem, the most simple type of reductions are enough. There is a *polynomial size projection* of $\mathcal{L}'$ to another language $\mathcal{L}$ if there is a circuit that computes $\mathcal{L}'$ and is composed of a single gate labelled by $\mathcal{L}$, such that the number of inputs of this gate is bounded by a polynomial. A language $\mathcal{L}$ is said to be NC^1*-complete under projections* if it is in NC^1 and every language of NC^1 admits a polynomial size projection to $\mathcal{L}$. Such a definition can be extended to any circuit class.

We will also consider NC^0 reductions. A language $\mathcal{L}'$ has a NC^0 reduction to a language $\mathcal{L}$ if $\mathcal{L}'$ can be computed by an NC^0 circuit with additional gates labelled by $\mathcal{L}$. We defined completness under NC^0-reduction analogously. In particular, if a NC^1-complete under NC^0 reductions language were to belong to ACC^0 (or TC^0) then the whole class would collapse to NC^1.

The notion that makes the theorem work is the one of solvable groups, that measure if a group is close of being commutative or not. For G a group and $x, y \in G$, the *commutator* of x and y is the quantity $[x,y] = xyx^{-1}y^{-1}$. The commutator of two subgroups $[G,H]$ is the group

generated by the element $[x,y]$ for $x \in G$ and $y \in H$. The *derived series* of G is the sequence

$$G^0 = G,$$
$$G^{i+1} = [G^i, G^i].$$

This series is decreasing for the subgroup inclusion.

Definition 4.2.
A group is said to be *solvable* if its derived series reaches the trivial group with one element. We denote by $\mathbf{G}_{sol}$ the set of solvable groups.

A monoid is said to be *solvable* if all the groups it contains are solvable. We denote by $\mathbf{M}_{sol}$ the set of solvable monoids.

For instance, any commutative group is solvable as the derived series is trivial after the second element. Everything is set for the statement of Barrington's theorem.

Theorem 4.3 (Barrington [14])**.**
Every non-solvable monoid is NC^1-complete under NC^0 reductions. Some of them are even NC^1-complete under projections.

The group of all permutations over a set with five elements, also known as S_5, is the standard example of a non-solvable group. Hence studying whether S_5 is in ACC^0 or TC^0 is a fundamental and difficult open problem.

4.1.2 Depth hierarchy

We saw that the depth hierarchy of AC^0 is strict. This non equality of the levels of the hierarchy can also be witnessed by regular languages. We will even exhibit a family of languages that are complete for every level of the hierarchy. A language $\mathcal{L}$ is said to be AC^0*-complete under projections* if every language of AC^0 admits a polynomial size projection to $\mathcal{L}$. We draw on ideas from [16, 77, 22].

The class of regular languages of interest is the set of Boolean formulae of bounded depth i and bounded bottom fan-in j, denoted by $\mathcal{O}_i^j$ and $\mathcal{A}_i^j$. For the i^{th} level, the alphabet is the set $A_i = \{0,1\} \cup \{\#_k \mid 1 \leq k \leq i\}$.

$$\mathcal{O}_0^j = \{w_1 \cdots w_j \mid \forall k, |w_k| = j \text{ and } \exists k, w_k = 1^j\}$$
$$\mathcal{A}_0^j = \{w_1 \cdots w_j \mid \forall k, |w_k| = j \text{ and } \forall k, |w_k|_1 \geq 1\}$$
$$\mathcal{O}_i^j = A_i^* \#_i \mathcal{A}_{i-1}^j \#_i A_i^*$$
$$\mathcal{A}_i^j = (\#_i \mathcal{O}_{i-1}^j)^* \#_i$$

Lemma 4.4.
For all $i, j \in \mathbb{N}$, the languages $\mathcal{O}_i^j$ and $\mathcal{A}_i^j$ are respectively in Σ_i^j and Π_i^j.

Proof. We proceed by induction on i. For $i = 0$, we can recognise the finite language $\mathcal{O}_0^j$ with the family of circuits whose only non empty circuit is the one of size j^2. This circuit then groups the inputs into j groups of size j, and aggregates them with an $\wedge$-gate. Those j $\wedge$-gates are grouped with an $\vee$-gate. The case $\mathcal{A}_0^j$ is dual with the inversion of $\wedge$-gates and $\vee$-gates.

Let $i \geq 1$. The top gate recognising $\mathcal{O}_i^j$ is an $\vee$-gate ranging over all couples of positions $k < l$. It is fed by a Π_{i-1}^j circuit which is

$$C \wedge \#_i(x_k) \wedge \#_i(x_l)$$

where C is the Π_{i-1}^j circuit recognising $\mathcal{A}_{i-1}^j$ given by the induction. Notice that the circuit is indeed in Π_{i-1}^j because the two extra $\wedge$-gates can be absorbed by the top gate of C.

The case $\mathcal{A}_i^j$ is dual with a top $\wedge$-gate and an intermediate circuit

$$C \vee \neg\#_i(x_k) \vee \neg\#_i(x_l)$$

where C is the Σ_{i-1}^j circuit recognising $\mathcal{O}_{i-1}^j$ given by the induction. The negation is not an issue as it can be reformulated as an $\vee$-gate of all $a(x_i)$ where a is a letter different than $\#_i$. □

We can prove the completeness theorem.

Theorem 4.5.
For all $i, j \in \mathbb{N}$, the languages $\mathcal{O}_i^j$ and $\mathcal{A}_i^j$ are respectively Σ_i^j-complete and Π_i^j-complete under projections.

Proof. Let $\mathcal{L}$ be a language recognised by a Σ_i^j-circuit (C_n). Let p be the polynomial that bounds the size of the circuit. The case Π_i^j is analogous. If $i = 0$, C_n is an $\vee$-gate fed by less than j $\wedge$-gates, themselves fed by less than j inputs. We can plug 0 to the $\vee$-gates and 1 to the $\wedge$-gates to compute the same language with the fan-in of every gate being precisely j. Let in_k for $1 \leq k \leq j^2$ be the tuple of inputs in the order of appearance in C_n, ie the first j are the inputs of the first $\wedge$-gate, and so on. The projection is the $\mathcal{O}_0^j$ circuit with j^2 inputs and such that the k^{th} input is wired to in_k.

If $i \geq 1$. We expand C_n into a tree, that is to say that each node as fan-out 1. This is done by duplicating shared nodes. Thanks to the constant depth, the circuit remains in Σ_i^j. We again number the inputs in order in_k for $1 \leq k \leq p(n)$. The projection is the $\mathcal{O}_i^j$-circuit with less than $2p(n)$ inputs. The wiring is the following: the first inputs take the constant $\#_i$, then the next ones take the wiring given for the first Π_{i-1}^j circuit. We proceed in this fashion for at most a polynomial number of inputs, then we wire a last $\#_i$. □

This allows to show that there are regular languages in Σ_i not in Σ_{i-1} for every i. Indeed, at least one of the $\mathcal{O}_i^j$ must not be in Σ_{i-1}. If it were not the case, every language $\mathcal{L}$ of Σ_i would

belong to some Σ_i^j, and hence would admit a polynomial size projection to $\mathcal{O}_i^j$. This would imply that $\mathcal{L}$ is in Σ_{i-1}, yielding $\Sigma_i = \Sigma_{i-1}$. This contradicts Theorem 3.19 of the separation of the depth hierarchy. It is worthy to note that once again the existence of regular complete languages gives that the separation by any language implies the separation by a regular language.

4.1.3 Importance of the addition function

Adding two numbers is the first mathematical operation taught in school. We can define the function (binary) ADDITION that takes $2n$ inputs and outputs the $n+1$ bits of the sum of the two integers over n bits given in input. We assume as a convention that the most significant bit is on the right. An algorithm to compute it is very simple: add the digits from left to right, and whenever the sum is greater than 2, we have to propagate a carry. This immediately gives that we can compute addition with a linear size circuit, but with a linear depth. Nevertheless, it is possible to compute the local sums and propagate the carry in parallel. This implies

$$\text{ADDITION} \in \mathsf{AC}^0.$$

The next natural question is can we show whether ADDITION is in the class LAC^0 or not. This question was already identified in the seminal paper of Furst, Saxe and Sipser on PARITY [41], and remains unsolved to this day. Thanks to the graph-theoretic technics of superconcentrators, it can be shown that ADDITION is not in WLAC^0. Hence ADDITION (to be more precise, the language defined by the right-most output bit) is a regular candidate to separate WLAC^0 and LAC^0. We recall that we know that the two classes are distinct. We also recall that we can separate LAC^0 and AC^0, but we do not know if it can be done by a regular language.

Completeness. In addition of finding the precise complexity ADDITION, this function also has tight links with the class AC^0. We define a class of circuits that can only perform additions.

Definition 4.6.
An ADD-circuit is a Boolean circuit with a constant number of gates labelled by the addition functions (and no other gates). It is asked that the gates have polynomial fan-in. The output of the circuit is a distinguished wire of one of the addition gates.
The class of ADD-circuits is called ADD^0.

In particular, an ADD-circuit has polynomially many edges. This class is clearly included in AC^0, as we can perform an addition in AC^0. The other direction, while not difficult, is much more interesting: we can emulate the whole AC^0 class with only a constant number of additions.

Theorem 4.7.
We have the language classes equality

$$\mathsf{ADD}^0 = \mathsf{AC}^0.$$

Proof. We prove $\mathsf{AC}^0 \subseteq \mathsf{ADD}^0$. Let $\mathcal{L}$ be a language defined by the circuit C. Let d be the depth of the circuit and $p(n)$ its size where p is a polynomial. It is usual to assume that the circuit is layered: we can split the gate into d layers, each gate of layer i getting fed by gates from

layer $i-1$. Here, we convert every $\vee$-gates into an $\neg$-gate of an $\wedge$-gate of $\neg$-gates, thanks to De Morgan's laws. The circuit hence only has $\wedge$-gates and $\neg$-gates. We will encode each layer by a single addition gate.

First, assume we want to compute an $\wedge$-gate of inputs $x_1, \dots, x_m$. The requested circuit is a single addition gate fed with $x_1, \dots, x_m$ and $m-1$ 0s and one 1. The most significant bit of the output is 1, that is to say the carry overflowed, if and only if all of the x_i are 1s.

Secondly, assume we want to compute a $\neg$-gate of a single input x. This can be seen as the addition of x and 1, in the least significant digit of the answer.

Last, assume we want to compute a bunch of $\wedge$-gates and $\neg$-gates at the same time. We will compute all the gates of layer i at the same time. Let j be its number of gates and for $1 \leq k \leq j$ let v_k, w_k be the vectors of inputs such that gate k is computed with the addition of v_k and w_k. In particular, they have the same size. The wanted circuit is the addition of the concatenation $v_1, 0, v_2, 0, \dots, 0, v_j$ and the concatenation $w_1, 0, w_2, 0, \dots, 0, w_j$. The output is the local most (or least) significant bit according to the type of the gate. The plugged 0s ensure that the different computation do not interfere with each other.

□

This theorem means that the addition is complete for AC^0 in a very strong sense (though weaker than projections). When dealing with the regular languages in AC^0, that is to say star-free languages, the theorem can be strengthened thanks to a result of Paperman, Salvati, and Soyez-Martin [89].

Theorem 4.8 ([89, theorem 6])**.**
Every regular language of AC^0 can be computed with an ADD^0-circuit with linear fan-in, and hence linearly many wires.

Proof. In [89], the result is stated in the framework of the so-called ADD-vectorial circuit. These circuits have access to addition gates, as well as vectorial versions of $\neg$, $\wedge$, pref-$\wedge$, pref-$\vee$, suf-$\wedge$, suf-lor, MSB and LSB. The prefix and suffix gates have n inputs and n outputs containing the $\wedge$ or $\vee$ of the first (resp last) inputs. MSB (resp. LSB) takes the first (resp. last) input that is 1 and flips it to 0. They construct an ADD-vectorial circuit with constantly many gates. By construction, they demand that their addition gates have to be of linear size. We have to see that we can remove the other type of gates to only have addition gates. The $\wedge$-gates and $\neg$-gates are like in the previous proof. A pref-$\vee$ is the negation of the addition of the input with the all 1s vector of the same size. The other prefix and suffix functions are similar. We can compute MSB and LSB with similar bit tricks. □

This implies that if ADDITION is in LAC^0, then every regular language is in LAC^0. The contrapositive is interesting as well, a lower bound for LAC^0 for any regular language implies that ADDITION is not in LAC^0.

All the relations between circuit classes that we have seen are summarised in Fig. 4.1.

4.2 Straubing properties

We have seen that many circuit classes that are distinct possess regular languages that witness the separation. We can take this one step further, and study precisely what are the regular languages in circuit classes. This is reminiscent of the consideration of the membership problem for regular languages. Indeed, designing an algorithm that answers if a given regular language is

in some class often enlightened us on the properties for said class. We defend in this thesis that regular languages are the backbone of circuit classes, and identifying them help to shed lights on their complicated behaviour. This have been achieved for a few classes that we will present here.

4.2.1 Statement

Straubing, in his book [126], presented a unified conjecture using logic to approach the problem.

Definition 4.9.
Let F be a logical fragment. We say that the *Straubing property* holds for F if

$$\mathrm{F}[\mathrm{arb}] \cap \mathrm{Reg} = \mathrm{F}[\mathrm{reg}].$$

Note that Straubing only made the conjecture that the Straubing property holds for logical fragments using existential and modular quantifiers. As we will see, the conjecture is false in general, hence the need to distinguish the fragments within which the Straubing property holds.

This property is important because it identifies the regular language in the non-uniform complexity class defined by F[arb]. Combined with the power of algebra, it may even yield decidability of $\mathrm{F}[\mathrm{arb}] \cap \mathrm{Reg}$. Moreover, it states that any formula of F with any arbitrary complicated predicates that defined a simple regular language can be rephrased with only regular predicates. In other words, the extra power given by arbitrary predicates is useless when defining regular languages.

To prove that a fragment has the Straubing property, there is a general recipe that can be applied. It highlights the interplay between algebra and combinatorics, most of the time of circuits. Of course, having a recipe does not mean that proving such properties is easy: proving lower bounds is notoriously difficult.

- Remark that by the definition of regular predicates, F[reg] only defines regular languages. Therefore all is left to do is to take a regular language not in F[reg] and show that it is not in F[arb].
- Prove that F[reg] and $\mathrm{F}[\mathrm{arb}] \cap \mathrm{Reg}$ are lm-varieties of stamps.
- Identify algebraically F[reg].
- Use this knowledge to find a class of regular languages for which a lower bound against F[arb] has to be found. An equivalent circuit form might be useful.

Sometimes, using lm-varieties is uneasy. Indeed, the algebraic part of the proof schemes often requires to prove that a variety of monoids is local, which is difficult. It even happens that the variety in question is not local, in which case even more algebraic work has to be done. This calls for a restricted version of the Straubing property, with neutral letters, that retains most of its interest and combinatorial flavour.

Definition 4.10.
Let F be a logical fragment. We say that the *neutral Straubing property* holds for F if

$$\mathrm{F}[\mathrm{arb}] \cap \mathrm{Reg} \cap \mathrm{Neut} = \mathrm{F}[<] \cap \mathrm{Neut}.$$

The neutral Straubing property is almost always weaker than the Straubing property. This is the case whenever adding local numerical predicates and local letter predicates gives the same languages.

Lemma 4.11.
Let F be a fragment such that F[<] is a monoid variety and F[<, loc] = F[<, loc_α]. Then F has the neutral Straubing property if F has the Straubing property.

Proof. We first prove that

$$\mathrm{F}[\mathrm{reg}] \cap \mathrm{Neut} = \mathrm{F}[<] \cap \mathrm{Neut}.$$

This is enough to conclude by intersecting both sides in the Straubing property by Neut. We write F[reg] = F[<, loc, mod] = F[<, loc_α, mod]. Let $\mathbf{V}$ be the variety of monoids associated to F[<]. By Theorem 2.22, F[<, loc_α] is associated to $\mathbf{V} * \mathbf{D}$. By Theorem 2.23, F[<, loc_α, mod] is associated to $\mathbf{V} * \mathbf{D} * \mathbf{MOD}$. Now if a language in F[reg] has a neutral letter, its syntactic stamp is in $\mathbf{V} * \mathbf{D} * \mathbf{MOD}$ and then is in $\mathbf{V}$. In this case, it is also in F[<] concluding the proof. □

4.2.2 Positive examples

We illustrate the recipe with FO. We first prove that FO[arb] ∩ Reg satisfies the wanted closure properties.

Lemma 4.12.
The class FO[arb] ∩ Reg is a lm-variety of languages.

Proof. We show all three points that define lm-varieties for FO[arb] = AC^0, since we already know that they stand for regular languages.

- Boolean operation: First-order formulae are closed by definition under ∧, ∨ and ¬.
- Quotients: Let $\mathcal{L} \in \mathrm{FO}[\mathrm{arb}]$ and a be a letter. Consider the circuit for the words of length n in $\mathcal{L}$. We can hardwire the first letter to a, the resulting circuit has $n-1$ inputs, and recognises a word w if and only if $w \in \mathcal{L}$. The family thus obtained recognises $a^{-1}\mathcal{L}$. The argument for right quotient is similar.
- lm-morphisms: Let $\mathcal{L}$ over the alphabet B and h be an lm-morphism such that $h(A) \subseteq B^k$ for some k. Consider the circuit for the words of $\mathcal{L}$ of length kn for some n. Given a word in A^n , we can use NC^0-circuits to map each input letter $a \in A$ to $h(a)$, and we can feed the resulting word to the circuit for $\mathcal{L}$. A word $w \in A^n$ is thus accepted if and only if $h(w) \in \mathcal{L}$, hence the circuit family thus defined recognises $h^{-1}(\mathcal{L})$.

□

This proof emphasises why the length-multiplying assumption in the definition of lm-varieties is needed when dealing with circuits. We recall that $\mathbf{A}$ is the class of aperiodic monoids and that $\mathbf{Q}$ is the operator introduced in Section 2.4 to study the addition of modular predicates.

Theorem 4.13 (Barrington, Compton, Straubing, and Thérien[14, theorem 3])**.**
The fragment FO has the Straubing property:

$$\mathsf{AC}^0 \cap \mathrm{Reg} = \mathrm{FO}[\mathrm{arb}] \cap \mathrm{Reg} = \mathrm{FO}[\mathrm{reg}] = \mathbf{QA}.$$

Proof. Obviously, FO[reg] ⊆ FO[arb] ∩ Reg. We already know by Example 2.26 that FO[reg] is a lm-variety of stamps equal to **QA**. Take a regular language $\mathcal{L}$ not in **QA**, we want to show that it is not in AC^0. Let its syntactic monoid be M and syntactic morphism be μ. Let s be its stability index. It means that M_s, the stable monoid of M, is not in A. Hence there exists a word w of size s such that

$$\mu(w)^{\omega+1} \neq \mu(w)^{\omega}.$$

Let q the smallest integer such that $\mu(w)^{\omega+q} = \mu(w)^{\omega}$. By the property of the stable monoid, we can find two words of size s such that $\mu(w)^{\omega+1} = \mu(u)$ and $\mu(w)^{\omega} = \mu(v)$.

We will show that if $\mathcal{L}$ were to be AC^0, then MOD_q would be as well. This would lead to a contradiction by Lemma 3.18. Thanks to the fact that F[arb] ∩ Reg is a lm-variety, $\mathcal{L}' = \mu^{-1}(\mu(v))$ is also in AC^0. We need a gadget that takes a single input bit, and outputs the encoding of u on input 1, and the encoding of v otherwise. To construct a circuit C of size n, we put one such gadget under each input, giving a word x. There is beneath it an AC^0 circuit for $\mathcal{L}'$, being fed by x. The monoid value $\mu(x)$ is a concatenation of $\mu(u)$ and $\mu(v)$, and satisfies

$$\mu(x) = \mu(w)^{n\cdot\omega+p} = \mu(w)^{\omega+r}$$

where p is the number of 1s in the input and r is the remainder in the Euclidean division of p by q. If p is divisible by q, then $r = 0$ and $\mu(x) = \mu(v)$ hence the circuit is accepting. Otherwise, $0 < r < q$ and $\mu(x) \neq \mu(v)$ and the circuit is not accepting. This gives an AC^0 circuit for MOD_q. □

We list in the following the fragments for which we know that the Straubing property holds. For p a prime integer, the variety $\mathbf{M}_{sol}^p$ is the class of monoids whose groups are solvable and size a power of p.

Theorem 4.14 (Barrington, Compton, Straubing, and Thérien[14, theorem 5])**.**
Let p be a prime integer. The fragment FO + MOD[p] has the Straubing property:

$$\mathsf{ACC}^0[p] \cap \mathrm{Reg} = (\mathrm{FO} + \mathrm{MOD}[p])[\mathrm{arb}] \cap \mathrm{Reg} = (\mathrm{FO} + \mathrm{MOD}[p])[\mathrm{reg}] = \mathbf{M}_{sol}^p.$$

It can be extended to show that ACC^0 can do all of M_{sol}. Combined with Barrington's theorem (Theorem 4.3), it implies that the regular languages of ACC^0 (with non prime moduli) are either precisely $\mathbf{M}_{sol}$ or every regular languages.

Concerning, the quantifiers alternation hierarchy (or equivalently the depth hierarchy of AC^0), the Straubing property is only known to hold for the first levels.

Theorem 4.15 (Krebs, Straubing [73])**.**
The Straubing property holds for Σ_1 and $\mathcal{B}\Sigma_1$:

$$\Sigma_1[\mathrm{arb}] \cap \mathrm{Reg} = \Sigma_1[\mathrm{reg}]$$
$$\mathcal{B}\Sigma_1[\mathrm{arb}] \cap \mathrm{Reg} = \mathcal{B}\Sigma_1[\mathrm{reg}]$$

When applying the recipe, the proof by contradiction tells us that every regular language that can be defined with arbitrary predicates can be restated with only regular predicates, but does not construct such a sentence. The only case for which a direct constructive proof has been found is for Σ_1 and $\mathcal{B}\Sigma_1$. The case for Σ_2 is the subject of Chapter 5.

There are also classes for which we can identify the regular languages within, without an equivalent logical formalism to properly state a Straubing property. We recall that **DA** is the variety of aperiodic monoids such that all $\mathcal{J}$-classes that contain one idempotent only contain idempotents, and that **Q** and **L** are the operators introduced during the study of regular predicates in Section 2.4.

Theorem 4.16 (Koucký, Pudlak, and Therien[72, theorem 14], Cadilhac and Paperman[23])**.** The regular languages of WLAC^0 are identified. In symbols:

$$\mathsf{WLAC}^0 \cap \mathrm{Reg} = \mathbf{QLDA}.$$

The first group of authors showed it for regular languages with a neutral letter, the second one extended it to the full statement.

A result that does not appear in the literature, but that is easy to obtain thanks a known lower bound concerns AC^0 circuits enhanced with a logarithmic number of maj-gates. We can prove they cannot do more regular languages than AC^0.

Theorem 4.17.
The regular languages computable with an AC^0 circuit with $O(\log(n))$ maj-gates are exactly the regular languages of AC^0, that is to say **QA**.

Proof. The proof is the same as the proof of Theorem 4.13. This class of circuits is wider than AC^0 and therefore can compute every language in **QA**. For $q \in \mathbb{N}$, the same reduction works to show that the existence of a circuit for MOD_q contradicts Theorem 3.23. □

4.2.3 First negative example: FO + S_5

The local predicates are all expressible in first-order logic, hence we can rewrite the local predicates in a $\mathrm{FO} + S_5$ sentence, to obtain a sentence without them. For instance, $x = y + 1$ get replaced by

$$\forall z,\ x \leq z \leq y \Rightarrow (z = x \vee z = y).$$

This implies that $(\mathrm{FO} + S_5)[\mathrm{reg}] = (\mathrm{FO} + S_5)[<, \mathrm{mod}]$. In particular, this fragment falls into the precondition of Lemma 4.11. We will then only show that $\mathrm{FO} + S_5$ does not have the neutral Straubing property. This will imply that the Straubing property does not hold as well.

First of all, the quantifiers over S_5 hint a link to Barrington's theorem.

Lemma 4.18.
We have the following

$$\mathrm{Reg} \subseteq \mathrm{Lin}(S_5)[\mathrm{arb}].$$

This implies the same expressivity result of the stronger logic $\mathrm{FO} + S_5$:

$$(\mathrm{FO} + S_5)[\mathrm{arb}] \cap \mathrm{Reg} = \mathrm{Reg}.$$

Proof. By Theorem 3.14, $\mathrm{Lin}(S_5)[\mathrm{arb}]$ is equivalent to $\mathsf{AC}^0[S_5]$, a class of circuits that can use gates labelled by S_5. By Barrington's theorem (Theorem 4.3), every NC^1 language, and hence every regular language, can be computed by a single gate labelled by S_5. □

Determining the power of $(\mathrm{FO}+S_5)[\mathrm{reg}]$ requires more work. For $\mathcal{G}$ a set of groups, we denote by $[\mathcal{G}]_{\mathbf{A}}$ its Krohn-Rhodes closure. It is the closure of $\mathcal{G}$ and $\mathbf{A}$ by wreath product and division. By Krohn-Rhodes theorem, $[\mathcal{G}]_{\mathbf{A}}$ is the variety of monoids that divide a monoid whose Krohn-Rhode decomposition can only use simple groups that divide G. Barrington, Immerman and Straubing found a powerful general theorem that we instanciate here for the case under study.

Theorem 4.19 (Barrington, Immerman, and Straubing [12, Theorem 12.1])**.**
The variety of monoids associated to $(\mathrm{FO}+S_5)[<]$ is $[S_5]_{\mathbf{A}}$.

All is left to do is find a monoid not in $[S_5]_{\mathbf{A}}$. Any simple group which does not divides S_5 would work. There is a rich mathematical literature on finite simple groups, for instance the monster group of order

$$808,017,424,794,512,875,886,459,904,961,710,757,005,754,368,000,000,000$$

does not divides S_5.

We can now conclude.

Theorem 4.20.
The fragment $\mathrm{FO}+S_5$ does not have the Straubing property:

$$(\mathrm{FO}+S_5)[\mathrm{reg}] \subsetneq (\mathrm{FO}+S_5)[\mathrm{arb}] \cap \mathrm{Reg}.$$

We can deduce the same theorem for the logic with usual quantifiers.

Corollary 4.21.
The fragment $\mathrm{Lin}(S_5)$ does not have the Straubing property:

$$\mathrm{Lin}(S_5)[\mathrm{reg}] \subsetneq \mathrm{Lin}(S_5)[\mathrm{arb}] \cap \mathrm{Reg}.$$

Proof. By Lemma 4.18, $\mathrm{Lin}(S_5)[\mathrm{arb}] \cap \mathrm{Reg} = \mathrm{Reg}$. Moreover, $\mathrm{Lin}(S_5)[\mathrm{reg}] \subseteq (\mathrm{FO}+S_5)[\mathrm{reg}] \subsetneq \mathrm{Reg}$. □

4.2.4 Second negative example: FO ∘ MOD ∘ FO

For the same reason as at the beginning of Section 4.2.3, $\mathrm{FO} \circ \mathrm{MOD} \circ \mathrm{FO}$ can express every local predicate, hence it satisfies the precondition of Lemma 4.11. We then focus on disproving the neutral Straubing property for $\mathrm{FO} \circ \mathrm{MOD} \circ \mathrm{FO}$. Note that it is specific to fragments that impose an order on the standard and modular quantifications. No new insight is given towards the Straubing property for $\mathrm{FO}+\mathrm{MOD}$, whose status is still unknown.

As before, we rely on an unexpected circuit result that gives unreasonable power to $(\mathrm{FO} \circ \mathrm{MOD} \circ \mathrm{FO})[\mathrm{arb}]$. This time it follows the fact that all of ACC^0 can be simulated with a probabilistic

circuit with only mod-gates. Hansen and Koucky proved that, and gave a derandomisation procedure for the latter model. From that they give the following theorem, where $\Sigma_2 \circ \mathsf{CC}^0$ is the class of circuits with a top $\vee$-gate, a bunch of $\wedge$-gates, and then CC^0 circuits:

Theorem 4.22 (Hansen and Koucky [57, Corollary 4.7])**.**
The following circuit classes are equivalent:

$$\mathsf{ACC}^0 = \Sigma_2 \circ \mathsf{CC}^0 = \Pi_2 \circ \mathsf{CC}^0.$$

This implies that $(\mathrm{FO} \circ \mathrm{MOD} \circ \mathrm{FO})[\mathrm{arb}]$, with equivalent circuit class is the class of circuits with at most two alternations between mod-gates and other type of gates, can do all the regular languages in ACC^0.

Lemma 4.23.
We have

$$\mathbf{M}_{sol} \subseteq (\mathrm{FO} \circ \mathrm{MOD} \circ \mathrm{FO})[\mathrm{arb}] \cap \mathrm{Reg}.$$

We can use the same result as in Theorem 4.19, to characterise $(\mathrm{FO} \circ \mathrm{MOD} \circ \mathrm{FO})[\mathrm{reg}]$.

Lemma 4.24.
We have

$$(\mathrm{FO} \circ \mathrm{MOD} \circ \mathrm{FO})[\mathrm{reg}] \subseteq \mathbf{A} * \mathbf{G} * \mathbf{A}.$$

Proof. We overapproximate the mod-gates by gates labelled by any groups. Analysing the proof of [12] gives that alternation between group quantifiers and usual quantifiers gives the alternation between wreath products of aperiodic monoids and groups. □

All is left to do is find a monoid in $\mathbf{M}_{sol}$ not in $\mathbf{A} * \mathbf{G} * \mathbf{A}$. Let $M_3(\mathbb{Z}/2\mathbb{Z})$ be the monoid of 3×3 upper triangular matrices over $\mathbb{Z}/2\mathbb{Z}$ with matrix multiplication. An element of $M_3(\mathbb{Z}/2\mathbb{Z})$ is a matrix:

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ 0 & x_{22} & x_{23} \\ 0 & 0 & x_{33} \end{pmatrix}$$

where $x_{ij} \in \mathbb{Z}/2\mathbb{Z}$.

It is known that the Krohn-Rhodes complexity hierarchy, that computes the number of alternation between aperiodic monoids and groups, is strict. This separation is in particular witnessed by monoids of upper triangular matrices.

Lemma 4.25 (Kambites [67])**.**
The monoid $M_3(\mathbb{Z}/2\mathbb{Z})$ has Krohn-Rhodes complexity 2. In particular, $M_3(\mathbb{Z}/2\mathbb{Z}) \notin \mathbf{A} * \mathbf{G} * \mathbf{A}$.

To conclude, we have to see that this monoid is in $\mathbf{M}_{sol}$. For that, we use the well known fact that the smallest non-solvable group, a subgroup of S_5, has 60 elements. The monoid $M_3(\mathbb{Z}/2\mathbb{Z})$

having $2^6 = 64$, we fall short of a very simple proof that it is a solvable monoid. We have to look at it in more detail. We call a matrix invertible if it has only ones on the diagonal. There are 8 invertible matrices and 56 non-invertible matrices. It is easy to see that multiplying a non-invertible matrix by anything gives a non-invertible matrix. This implies that invertible and non-invertible matrices are never $\mathcal{J}$ equivalent. So any group in $M_3(\mathbb{Z}/2\mathbb{Z})$ has cardinality at most 56, and is therefore solvable.

Theorem 4.26.
The fragment FO ∘ MOD ∘ FO does not have the Straubing property:

$$(\mathrm{FO} \circ \mathrm{MOD} \circ \mathrm{FO})[\mathrm{reg}] \subsetneq (\mathrm{FO} \circ \mathrm{MOD} \circ \mathrm{FO})[\mathrm{arb}] \cap \mathrm{Reg}.$$

It allows to deduce a few other falsifications of Straubing properties, for subfragments of FO ∘ MOD ∘ FO that are expressive enough to apply Theorem 4.22.

Corollary 4.27.
Let $i \geq 2$. The following fragments do not have the Straubing property:

- $\Sigma_i \circ \mathrm{MOD}$,
- $\Pi_i \circ \mathrm{MOD}$,
- $\mathrm{FO} \circ \mathrm{MOD}$,
- $\Sigma_i \circ \mathrm{MOD} \circ \mathrm{FO}$,
- $\Pi_i \circ \mathrm{MOD} \circ \mathrm{FO}$.

Bibliography of the current chapter

[12] David Barrington, Neil Immerman, and Howard Straubing. "On uniformity within NC_1". In: July 1988. DOI: 10.1109/SCT.1988.5262.

[13] David A. Barrington. "Bounded-width polynomial-size branching programs recognize exactly those languages in NC1". In: *Journal of Computer and System Sciences* 38 (1989). DOI: 10.1016/0022-0000(89)90037-8.

[14] David A. Barrington, Kevin Compton, Howard Straubing, and Denis Thérien. "Regular languages in NC_1". In: *Journal of Computer and System Sciences* 44.3 (1992). DOI: 10.1016/0022-0000(92)90014-A.

[16] David A. Mix Barrington and Denis Thérien. "Finite monoids and the fine structure of NC_1". In: *J. ACM* 35.4 (1988). DOI: 10.1145/48014.63138.

[22] S. R. Buss. "The Boolean formula value problem is in ALOGTIME". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987. DOI: 10.1145/28395.28409.

[23] Michaël Cadilhac and Charles Paperman. *The Regular Languages of Wire Linear AC 0*. en. Dec. 2021. DOI: 10.1007/s00236-022-00432-2.

[41] Merrick Furst, James B Saxe, and Michael Sipser. "Parity, circuits, and the polynomial-time hierarchy". en. In: (1984). DOI: 10.1007/BF01744431.

[57] Kristoffer Arnsfelt Hansen and Michal Koucký. "A New Characterization of ACC0 and Probabilistic CC0". en. In: *computational complexity* 19.2 (May 2010). DOI: 10.1007/s00037-010-0287-z.

[67] Mark Kambites. "On the Krohn–Rhodes complexity of semigroups of upper triangular matrices". In: *International Journal of Algebra and Computation* 17.01 (2007). DOI: 10.1142/S0218196707003548.

[72] Michal Koucký, Pavel Pudlak, and Denis Therien. "Bounded-depth circuits: Separating wires from gates". In: May 2005. DOI: 10.1145/1060590.1060629.

[73] Andreas Krebs and Howard Straubing. *Regular languages defined by first-order formulas without quantifier alternation*. Aug. 2022. DOI: 10.48550/arXiv.2208.10480.

[77] Nancy A. Lynch. "Log Space Recognition and Translation of Parenthesis Languages". In: *J. ACM* 24 (1977). DOI: 10.1145/322033.322037.

[89] Charles Paperman, Sylvain Salvati, and Claire Soyez-Martin. "An Algebraic Approach to Vectorial Programs". In: *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*. Ed. by Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté. Vol. 254. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. DOI: 10.4230/LIPIcs.STACS.2023.51.

[126] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. en. Boston, MA: Birkhäuser, 1994. DOI: 10.1007/978-1-4612-0289-9.

Part II

Results on Regular Languages

Chapter 5

Circuit Complexity: the Regular Languages of Σ_2

Outline of the current chapter

This chapter is based on the paper "The Regular Languages of First-Order Logic with One Alternation", which is joint work with Michaël Cadilhac, Charles Paperman and Thomas Zeume [10]. The result of Section 5.4 is new.

In Chapter 4, regular languages were advocated to be the backbone of circuit complexity for numerous reasons. Thanks to the framework of logic, Howard Straubing emitted a conjecture on the form of regular languages inside circuit classes (Definition 4.9). This Straubing property has been shown to hold for a handful of logic classes and not to hold for even fewer classes. Its status remains open for plentiful natural logics. We investigate in this chapter the fragment Σ_2 of formulae with a single alternation of quantifiers and that start with an existential quantifier. We show that this fragment has the Straubing property, giving the first significant progress on the study of Straubing properties in more than 20 years. In a first part, we prove the neutral

Straubing property (Definition 4.10) thanks to new lower bounds against depth-3 Boolean circuits. Then we use the power of finite categories to study the addition of regular predicates into $\Sigma_2[<]$ (whose study started in Section 2.4), to obtain the Straubing property for Σ_2 in its full generality.

5.1 Lower bounds against Σ_2[arb]

We will exclusively use Lemma 3.13 that states that the logic Σ_2[arb] and the circuit class Σ_2 can express the same languages. The combinatorial essence of circuits will be useful in proving lower bounds. After presenting *limits*, that will fool depth-3 circuits, we will present two way of finding them. The first one, *sunflowers*, is simpler to use and is only needed to carry out the warm-up in Section 5.2. The second one, *tangledness*, is more involved but is powerful enough to have lower bounds to prove the Straubing property for Σ_2.

5.1.1 Limits

The class of circuits under study has depth 3 (with the last layer of bounded fan-in). We reuse techniques developed by Håstad, Jukna and Pudlàk in [58]. Therein, the paternity of the idea is given to Sipser [120]. These ideas are exposed in chapter 11.3 of a book of Jukna [65]

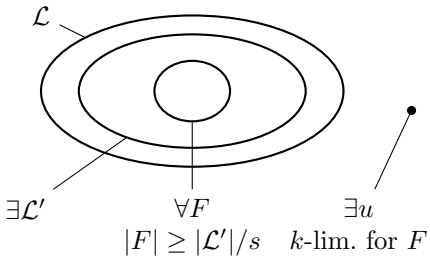
Definition 5.1.
Let F be a set of words, all of same length n, and $k > 0$. A *k-limit* for F is a word u of length n such that for any set of k positions, a word in F matches u on all these positions. In symbols, u satisfies:

$$\forall P \in \{1,\dots,n\}^k,\ \exists v \in F,\ \forall p \in P,\ u_p = v_p.$$

Any non-empty set of words F possesses a k-limit for every k: any word in F satisfies the property. However, we will only be interested in k-limits outside of F, and even outside of a superset of F.

Lemma 5.2 (Håstad, Jukna, and Pudlàk[58, Lemma 2.2])**.**
Let $\mathcal{L}$ be a set of words all of same length n and C be a Σ_2 circuit that accepts at least all the words of $\mathcal{L}$. Let k be the top fan-in of C and s its size.

Assume there is a subset $\mathcal{L}' \subseteq \mathcal{L}$ such that for any $F \subseteq \mathcal{L}'$ of size at least $|\mathcal{L}'|/s$ there is a k-limit for F that does not belong to $\mathcal{L}$. Then C accepts a word outside of $\mathcal{L}$. The hypothesis can be represented graphically as:



Proof. At the bottom of C, we have an $\vee$-gate of fan-in at most s that receives the result of some $\wedge$-gates. By counting, one of these $\wedge$-gates should accept a subset F of $\mathcal{L}'$ of size at least $|\mathcal{L}'|/s$; we will now focus on that gate. Let $u \notin \mathcal{L}$ be the k-limit for F that exists by hypothesis. Consider an $\vee$-gate that feeds into the $\wedge$-gate under consideration. This $\vee$-gate checks the contents of a subset $P \subseteq [n]$ of at most k positions of the input. By hypothesis, there is a word v in F that matches u on all the positions in P, hence the $\vee$-gate cannot distinguish between u and v and must output 1 (true) on u as v must be accepted. This holds for all the $\vee$-gates feeding into the $\wedge$-gate under consideration, hence the $\wedge$-gate must accept u, and so does C. □

We immediately deduce from this statement a combinatorial property on a language for its non-expressibility by a Σ_2 circuit.

Corollary 5.3.
Let $\mathcal{L}$ be a language and write $\mathcal{L}_n$ for the subset of words of length n in $\mathcal{L}$. Assume that for any $k, d \in \mathbb{N}$, there is an $n \geq 2$ and a subset $\mathcal{L}' \subseteq \mathcal{L}_n$ such that every subset $F \subseteq \mathcal{L}'$ of size at least $|\mathcal{L}'|/n^d$ admits a k-limit outside of $\mathcal{L}$. Then $\mathcal{L}$ is not in Σ_2[arb].

Proof. For a contradiction, assume there is a Σ_2 circuit family for $\mathcal{L}$, with top fan-in k and size n^d for $n \geq 2$. Let n be the value provided by the hypothesis, then the circuit C for $\mathcal{L}_n$ satisfies the hypotheses of Lemma 5.2, hence C accepts a word outside of $\mathcal{L}$, a contradiction. □

5.1.2 Sunflower lemma

To find limits, we need a tool from extremal combinatorics: the renowned Erdős-Rado sunflower lemma. We consider families $\mathcal{F}$ containing sets of size s for some s. The *core* of a family $\mathcal{F}$ is the set

$$Y = \bigcap_{X \in \mathcal{F}} X.$$

The core of a family $\mathcal{F}$ may be the empty set. A *sunflower* is a family $\mathcal{F}$ in which the intersection between any two distinct sets in $\mathcal{F}$ is always Y. In other words, the mutual intersection is equal to any pairwise intersection. In this case, the size of $\mathcal{F}$ is its number of *petals*. Ërdos and Rado proved that any family big enough has to contain a large sunflower.

Lemma 5.4 (Erdős, Rado [38])**.**
Let $\mathcal{F}$ be a family of sets of size s and $p \geq 1$. If $|\mathcal{F}| > s!(p-1)^s$ then $\mathcal{F}$ contains a sunflower with p petals.

Families of size $(p-1)^s$ without a sunflower with p petals have been found. Refining the bound of $\mathcal{F}$ so that it contains a sunflower is a major open problem. The current record holders are Alweiss, Lovett, Wu and Zhang [6] and Rao [107] with a lower bound of order $(p \log(ps))^s$.

For our lower bounds, we will adopt a weaker version of sunflower, that will give better bounds. The *coreless* version of a family $\mathcal{F}$ with core Y is the family

$$\mathcal{F}_Y = \{X \setminus Y \mid X \in \mathcal{F}\}.$$

A sunflower is precisely the family such that $\mathcal{F}_Y$ is a family of pairwise disjoint sets. We relax this condition so that $\mathcal{F}_Y$ has only few intersections. A set X intersects a family $\mathcal{F}$ if all the sets of $\mathcal{F}$ have a non-empty intersection with X.

Definition 5.5.
A *flower* with p petals is a family $\mathcal{F}$ of sets of the same size s and core Y such that there are no sets of size strictly less than p that intersect $\mathcal{F}_Y$.

This new notion has a theorem that looks like the sunflower lemma. It was stated in [58] and is exposed, with other flower-related notions, in a book of Jukna [66].

Lemma 5.6 (Jukna [66, lemma 6.4])**.**
Let $\mathcal{F}$ be a family of sets of size s and $p \geq 1$. If $|\mathcal{F}| > (p-1)^s$ then $\mathcal{F}$ contains a flower with p petals.

5.1.3 Tangledness

After introducing the sunflower lemma for the warm-up in Section 5.2, we present a novel combinatorial tool tailored for lower bounds against languages not in $\Sigma_2[\mathrm{reg}]$.

Let Φ be a set of words of size n over alphabet of size m. The *entailment relation* of Φ is relating sets of pairs (i,c) of position/letters in words of Φ. Let us say that a word and a pair position/contents (i,c) *agree* if the letters at position i of the word is c, and that a word and a set of such pairs agree if the word agrees with *each* pair. We write $S \rhd u$ when a word u and a set S of pairs position/contents agree:

$$S \rhd u \quad \text{iff} \quad \forall (i,c) \in S,\ u_i = d.$$

We say that a set of pairs is an i-set if all its pairs have i as position.

A set S of pairs position/letters *entails* an i-set D if all words in Φ that agree with S also agree with some pair of D (in which case the pair is unique); additionally, the position i should not appear in S. In this case we write $S \vdash D$. In symbols:

$$\begin{aligned} S \vdash D \quad \text{iff} \quad & \forall c,\ (i,c) \notin S \wedge \\ & \forall u \in \Phi,\ S \rhd u \Rightarrow [\exists (i,c) \in D,\ u_i = c]. \end{aligned}$$

A set of words is called k-tangled if every position of a word is entailed by a subset of size k of its positions. We make this definition precise.

Definition 5.7.
A set of words Φ is said to be *k-tangled* if for any word $u \in \Phi$ and any position $i \leq n$, there are:

- an i-set D of pairs of size $\leq k$ that contains (i, u_i),
- a set S of pairs of size k that agrees with u,

such that $S \vdash D$.

We drop the k in k-tangled if it is clear from context. Our lemma asserts that tangled sets cannot have large cardinality.

Lemma 5.8.
Let k be an integer. Let Φ be a set of words of size n over an alphabet of size $m \geq 2^{9k^2}$.
If Φ is k-tangled, then $|\Phi| < m^{2kn/(2k+1)}$.

Proof. Assume Φ is k-tangled. We show that every word in Φ can be fully described in Φ by fully specifying a portion $k/(k+1)$ of its positions and encoding the letters of each of the other $1/(k+1)$ positions with integer between 1 and k. That is, if two words in Φ have the same such description, they are the same, hence Φ cannot be larger than the number of such descriptions. We first show this property, then derive the numerical implication on $|\Phi|$. We moreover make the technical assumption that n is a multiple of $k+1$ so as not to bother with integrability issues.

Let $u \in \Phi$, we construct iteratively a set K of positions that we will fully specify and a set K^+ of positions that are restricted when setting the positions in K.

First consider the pair $(1, u_1)$. Since Φ is tangled, there is an 1-set containing $(1, u_1)$, entailed by a set S that agrees with u. We add to K the positions of S and to K^+ the positions of S *and* position 1.

We now iterate this process: Take a pair (i, u_i) such that $i \notin K^+$. There is an i-set containing (i, u_i) that is entailed by a set S that agrees with u. Let S' be the set of positions of S that are not in K^+. We add S' to K, and $S' \cup \{i\}$ to K^+. Note that the size increase for K^+ is one more than that for K. We continue iterating until all positions appear in K^+.

We now bound the size of K at the end of the computation. For each iteration, in the worst case, we need to add k positions to K to obtain $k+1$ new positions in K^+ (this is the worst case in the sense that this is the worst ratio of the number of positions we need to pick in K to the number of positions that are put in K^+). In that case, after s steps, we have $|K| = sk$ and $|K^+| = sk + s$. Thus when $|K^+| = n$, that is, when no more iterations are possible, we have:

$$sk + s = n \Rightarrow s = \frac{n}{k+1}.$$

This shows that $|K| \leq kn/(k+1)$.

We now turn to describing the word u using K. We first provide all the letters of u at positions in K; call Z the set of pairs position/letters of μ that correspond to positions in K. We mark the positions of K as *specified*, and carry on to specify the other positions in a deterministic fashion.

We first fix an arbitrary order on sets of pairs of position/letters. We iterate through all the subsets of Z of size k, in order. For each such subset S, we consider, in order again, the subsets D that are entailed by S. Assume D is an i-set; if position i is already specified, we do nothing, otherwise, we describe which element of D is (i, u_i) using an integer between 1 and $[k]$, and mark i as specified. We proceed until all the subsets of Z have been seen, at which point, by construction of K, all the positions will have been specified. As claimed, given Z and the description of which elements in sets D correspond to the correct letters, we can reconstruct u.

Summing up, to fully describe u, we had to specify the positions of K (one of $\binom{n}{kn/(k+1)}$ possible choices), their letters (one of $m^{kn/(k+1)}$ possible choices), and for each position not specified by K, we needed to provide an integer between 1 and k (one of $k^{n/(k+1)}$ possible choices). This shows that:

$$
\begin{aligned}
|\Phi| &\leq \binom{n}{kn/(k+1)} \cdot m^{kn/(k+1)} \cdot k^{n/(k+1)} \\
&< 2^n \cdot 2^{(n/(k+1))(k\log m)} \cdot 2^{(n/(k+1))\log k} \\
&= 2^{(n/(k+1))((k+1)+k\log m+\log k)} \\
&\leq 2^{n/(k+1))((k+\frac{1}{3k})\log m)} && \text{(m large enough)} \\
&= m^{(k+\frac{1}{3k})n/(k+1)} \leq m^{2kn/(2k+1)}.
\end{aligned}
$$

□

This lemma is reminiscent of a result of Meier and Widgerson [80, Corollary 1.6]. Therein, random strings over $\{0,1\}^n$ are considered. They show that if a random string has high entropy (ie. is very uncertain), then the average over indices of the probability that an index is entailed by a set of size k is low. Whiletheir setting cover more that the uniform distribution over a subset of $\{0,1\}^n$, our result is not on average and works for bigger alphabets (even of size comparable with n).

5.2 Warm-up : $(ac^*b + c)^* \notin \Sigma_2[\mathbf{arb}]$

Let K be the regular language $(ac^*b + c)^*$. In other words, K is the language $(ab)^*$ with a neutral letter c. Following [58], we will prove that K is not in $\Sigma_2[\text{arb}]$. This will serve as an introduction to the proof of the Straubing property of Σ_2. Moreover, this language is expressible in $\Pi_2[<]$, making the lower bound somewhat subtle.

Fact 5.9.
The language $(ac^*b + c)^*$ is in $\Pi_2[<]$.

Proof. To see that, the formula is the conjunction of four clauses. A subformula that checks that the first non-neutral letter is an a:

$$\forall x,\ [\mathbf{b}(x) \Rightarrow \exists y,\ y \leq x \wedge \mathbf{a}(y)].$$

A subformula that checks that the last non-neutral letter is a b:

$$\forall x,\ [\mathbf{a}(x) \Rightarrow \exists y,\ y \geq x \wedge \mathbf{b}(y)].$$

A subformula that checks that there is an a between two consecutive bs:

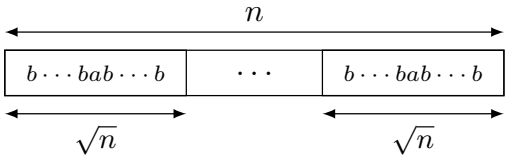
$$\forall x, y,\ [\mathbf{b}(x) \wedge \mathbf{b}(y) \Rightarrow \exists z,\ x \leq z \leq y \wedge \mathbf{a}(z)].$$

A subformula that checks that there is an a between two consecutive bs:

$$\forall x, y,\ [\mathbf{a}(x) \wedge \mathbf{a}(y) \Rightarrow \exists z,\ x \leq z \leq y \wedge \mathbf{b}(z)].$$

□

To show this claim, we consider a slightly different language. For any n that is a perfect square, we let Good_n be the set of words of length n over $\{a,b\}$ of the following shape:



In words, a word is in Good_n if it can be decomposed into $\sqrt{n}$ *blocks* of length $\sqrt{n}$, such that each of them has exactly one a. We let $\text{Good} = \bigcup_n \text{Good}_n$.

Lemma 5.10.
If K is in $\Sigma_2[\text{arb}]$, then so is Good.

Proof. This is easier to see on circuits, so assume there is a Σ_2 circuit family for K. For n a perfect square, we design a circuit for Good_n. On any input, we convert the b's to c's and insert a b every $\sqrt{n}$ positions; we call this the *expansion* of the input word. For instance, with $n=9$, the input $abb\,bab\,bba$ is expanded to $accb\,cacb\,ccab$. Clearly, if the input word is in Good_n, then its expansion is in K. Conversely, if a block of the input had two a's, the expansion will not add a b in between, so the expansion is not in K; similarly, if a block of the input contains only b's, it will be expanded to only c's sandwiched between two b's, and the expansion will not be in K (in the case where the block containing only b's is the first one, the expansion starts with $c\cdots cb$, again putting the expansion outside of K).

Thus a circuit for Good_n can be constructed by computing the expansion (this only requires wires and no gates), then feeding that expansion to a circuit for K. If the circuit family for K were in Σ_2, so would the circuit family for Good. □

We use Corollary 5.3 to show that $\text{Good} \notin \Sigma_2[\text{arb}]$. Let then $k,d \in \mathbb{N}$. The value of L' in Corollary 5.3 will simply be Good_n, and we show:

Lemma 5.11.
If n is large enough, any subset $F \subseteq \text{Good}_n$ with $|F| > k^{\sqrt{n}}$ has a k-limit outside of Good_n. This holds in particular if $|F| \geq |\text{Good}_n|/n^d$.

Proof. We rely on the flower lemma (Lemma 5.6). To apply this lemma, consider the mapping τ from words in Good_n to $2^{[n]}$ that lists all the positions where a word has an a. For instance, with $n=9$, $\tau(bba\,abb\,bab) = \{3,4,8\}$. For any word w in Good_n, $\tau(w)$ is of size $\sqrt{n}$. We let $\mathcal{F} = \{\tau(w) \mid w \in F\}$.

We now apply the lemma with $s = \sqrt{n}$ and $p = k+1$. Since $|\mathcal{F}| = |F|$, we can apply the lemma on $\mathcal{F}$ and obtain a subfamily $\mathcal{F}'$ that is a flower with $k+1$ petals. Let Y be its core. Consider the word u of length n over $\{a,b\}$ which has a's exactly at the positions in Y. Then:

- u is outside of Good_n. Indeed, $|Y| < \sqrt{n}$, since it is the intersection of distinct sets of size $\sqrt{n}$. Hence one of the blocks of u will contain only b's, putting it outside of Good_n.
- u is a k-limit. Let P be a set of k positions, we will find a word that is mapped to $\mathcal{F}'$ that matches u on P. If a position in P points to an a in u, then every word in $\mathcal{F}'$ has an a at

that position (by construction, since this position would belong to the core Y). Therefore, as we are looking for a matching word in $\mathcal{F}'$, we can find a word that is matching on the b's only. So we assume that P contains only positions on which u is b. Since $|P|$ is k, it cannot intersect $\mathcal{F}'$, hence there is a set $S \in \mathcal{F}'$ such that $S \cap P = \emptyset$. The set S is thus $\tau(w)$ for a word $w \in F$ that has a b on all positions in P. This word w thus matches u on P, concluding the proof of the main statement.

The "in particular" part is implied by the fact that, for n large enough:

$$\frac{|\mathrm{Good}_n|}{n^d} = \frac{\sqrt{n}^{\sqrt{n}}}{n^d} \geq k^{\sqrt{n}}.$$

□

This is all we need to conclude.

Theorem 5.12.
The language $K = (ac^*b + c)^*$ is not in $\Sigma_2[\mathrm{arb}]$.

Proof. Corollary 5.3 applied on Good, using Lemma 5.11, implies that Good $\notin \Sigma_2[\mathrm{arb}]$. Lemma 5.10 then asserts that K cannot be in $\Sigma_2[\mathrm{arb}]$ either. □

5.3 Neutral Straubing property

This section is devoted to the proof of the neutral Straubing property for Σ_2.

We can specialise the equations for the dot-depth (Theorem 2.29) in the case of Σ_2.

Theorem 5.13.
A regular language is in $\Sigma_2[<]$ if and only if its ordered syntactic monoid M is such that for any $x, y \in M$ with y a subword of x, it holds that

$$x^\omega \leq x^\omega y x^\omega.$$

The proof is along two main steps:

Section 5.3.1. We will start with a regular language with a neutral letter $\mathcal{L} \notin \Sigma_2[<]$. Since it is not in $\Sigma_2[<]$, there are $x, y \in M$ that falsify the equations of Theorem 5.13. We use these witnesses to build another language $\mathcal{T}$ that behaves the same with regard to $\Sigma_2[\mathrm{arb}]$ and show that it lies outside of $\Sigma_2[\mathrm{arb}]$, implying that $\mathcal{L} \notin \Sigma_2[\mathrm{arb}]$.

To show $\mathcal{T}$ out of $\Sigma_2[\mathrm{arb}]$, we identify (Section 5.3.1.1) a subset of well-behaved words of $\mathcal{T}$, and make some simple syntactical changes (in Section 5.3.1.2) on them so that they look like words in Good, in a similar fashion as the "expansions" of Lemma 5.10. The argument used in Lemma 5.10 then needs to be refined, as we do not have that any word outside of Good comes from a word outside of $\mathcal{T}$. We will define a set Bad of words that look like words in Good except for *one* block that contains only b's; Lemma 5.10 is then worded as: if K is in $\Sigma_2[\mathrm{arb}]$, then there is a $\Sigma_2[\mathrm{arb}]$ language that separates Good from Bad (Lemma 5.14).

Section 5.3.2. We show that no language of $\Sigma_2[\text{arb}]$ can separate Good from Bad. We thus need to provide a statement in the spirit of Lemma 5.11. We first write good and bad words in a succinct ("packed") way, as words in $[\sqrt{n}]^{\sqrt{n}}$, the i-th letter being some value v if the original word had the a of its i-th block in position v (Section 5.3.2.1). We then translate the notion of k-limit to packed words (Lemma 5.16). Finally, we use the fact that big sets of (packed) good words are not tangled (Lemma 5.8) and that we can find a k-limit inside sets of this form (Lemma 5.17).

5.3.1 If a language not in $\Sigma_2[<]$ is in $\Sigma_2[\text{arb}]$, we can separate Good from Bad with a language in $\Sigma_2[\text{arb}]$

5.3.1.1 Target language and some of its words

For the rest of this section, let $\mathcal{L} \subseteq A^*$ be a regular language with a neutral letter that lies outside of $\Sigma_2[<]$ and let M be its ordered syntactic monoid. Since L is not in $\Sigma_2[<]$, there are elements $x, y \in M$ such that $x \not\leq xyx$ with x an idempotent and y a subword of x. Let $\mathcal{T}$ be the language of M^* of words that evaluate to the upset of x. Clearly, any word of M^* that evaluates to xyx does *not* belong to $\mathcal{T}$. Because $\Sigma_2[\text{arb}] \cap \text{Neut}$ is a variety of monoids, both of $\mathcal{T}$ and $\mathcal{L}$ or none of them belongs to $\Sigma_2[\text{arb}]$.

By hypothesis, y is thus also a subword of x; this provides us with words that evaluate to x and y of the shape:

$$x_1 y_1 \dots x_t y_t \text{ evaluates to } x, \quad y_1 \dots y_t \text{ evaluates to } y,$$

where each x_i and y_i are letters in M. (Note that we can use the identity element of M as needed to fill up and ensure we have as many x_i's as y_i's.)

Let $n \in \mathbb{N}$ be a perfect square (whose value is meant to be taken large enough later). We define $\sqrt{n}+1$ words of length $t\sqrt{n}+t$ over M:

- For $1 \leq i \leq \sqrt{n}$,
$$x^{(i)} = \left(1^{i-1} x_1 1^{\sqrt{n}-i} \cdot y_1\right) \cdots \left(1^{i-1} x_t 1^{\sqrt{n}-i} \cdot y_t\right).$$
Here $1 \in M$ is the neutral element of M. Note that these words evaluate to x.
- Additionally, we consider the word $1^{\sqrt{n}} y_1 \cdots 1^{\sqrt{n}} y_t$, which evaluates to y, and we simply write y for it. Note that y can be obtained by replacing all the letters x_j by 1 from any word $x^{(i)}$.

Call $\mathcal{T}$*-good* a concatenation of $\sqrt{n}$ words of the form $x^{(i)}$ sandwiched between two words $x^{(1)}$ (the 1 is arbitrary), and $\mathcal{T}$*-bad* a word obtained by changing, in a $\mathcal{T}$-good word, *exactly one* of the words $x^{(i)}$ to y (but for the $x^{(1)}$ at the beginning and end). By construction, any $\mathcal{T}$-good word evaluates to x in M, so belongs to $\mathcal{T}$, while any $\mathcal{T}$-bad word evaluates to xyx, hence does not belong to $\mathcal{T}$. Note that if we had switched *two* blocks of a $\mathcal{T}$-good word to y, we would not be able to say whether it belonged to $\mathcal{T}$ or not.

5.3.1.2 $\mathcal{T}$-good, $\mathcal{T}$-bad to Good and Bad

The $\mathcal{T}$-good and $\mathcal{T}$-bad words contain a lot of redundant information, for instance $x^{(i)}$ is of length $t\sqrt{n}+t$, while all the information it really contains is $1 \leq i \leq \sqrt{n}$. Recall the set Good_n of Section 5.2 which contains all words of length n over $\{a, b\}$ that can be divided into $\sqrt{n}$ blocks of length $\sqrt{n}$, each containing a single a. Again, we let Good be all such words, of any perfect

square length. Define similarly Bad_n as the set of words that are like Good_n except for *one* block which has only b's, and let $\mathrm{Bad} = \bigcup_n \mathrm{Bad}_{n^2}$.

In the next lemma, we show that we can modify, using only wires in a circuit, words over $\{a,b\}$ so that if they are in Good they become $\mathcal{T}$-good, and if they are in Bad they become $\mathcal{T}$-bad. This modification is simple enough that we can take a Σ_2 circuit family for $\mathcal{T}$, apply the modification at the top of each circuit, and still have a circuit family in Σ_2; the resulting circuit family separates Good from Bad:

Lemma 5.14.
If $\mathcal{T} \in \Sigma_2[\mathrm{arb}]$, then there is a $\Sigma_2[\mathrm{arb}]$ language that separates Good from Bad.

Proof. As in Lemma 5.10, this is easier seen on circuits: we design a circuit for inputs of length n over $\{a,b\}$ that separates Good from Bad.

Consider the first block of $\sqrt{n}$ letters of the input. We replicate it t times, with the i-th replication changing b's to 1 and a's to x_i. We then concatenate these and add y_i between the i-th and $(i+1)$-th replication. For instance, $b^7 a b^{\sqrt{n}-8}$ would turn into:

$$\left(1^7 x_1 1^{\sqrt{n}-8} \cdot y_1\right) \cdots \left(1^7 x_t 1^{\sqrt{n}-8} \cdot y_t\right) = x^{(8)}.$$

In particular, if the block were all b's, we would obtain the word y, which has no letter x_j.

We can do this to each block of $\sqrt{n}$ letters, concatenate the resulting words, then add the word $x^{(1)}$ at the beginning and the end. Note that these operations can be done with only wires, with no gates involved.

If the input word is in Good, then the word produced is $\mathcal{T}$-good, hence in $\mathcal{T}$. If it was in Bad, then the resulting word would be $\mathcal{T}$-bad, hence would lie outside of $\mathcal{T}$. This shows that the desired circuit can be constructed using the above wiring followed by the circuit for $\mathcal{T}$ for inputs of length $(t\sqrt{n}+t)(2+\sqrt{n})$. Since t is a constant and depends solely on L, the resulting circuit is of polynomial size and of the correct shape. □

5.3.2 No language in Σ_2[arb] separates Good from Bad

Note that this section is independent from the previous one. We will now rely on Corollary 5.3 to show that any $\Sigma_2[\mathrm{arb}]$ language L that accepts all of Good must accept a word in Bad. To apply Corollary 5.3, *from this point onward* we let $k, d \in \mathbb{N}$, and set n to be a large enough value that depends only on k and d. The role of L' in the statement of Corollary 5.3 will be played by Good_n and we will build k-limits belonging to Bad_n, which we call *bad k-limits*. The reader may check that the statements of the forthcoming Lemma 5.17 and the already proved Lemma 5.8 conclude the proof.

5.3.2.1 Packed words

Words in Good_n and Bad_n can be described by the position of the letter a in each block of size $\sqrt{n}$. We make this explicit, by seeing $[\sqrt{n}, \perp] = [\sqrt{n}] \cup \{\perp\}$ as an alphabet, and working with words in $[\sqrt{n}, \perp]^{\sqrt{n}}$. We call these words *packed* and will use Greek letters λ, μ, ν for them; we also call the letter at some position in packed words its *contents* at this position, only to stress that we are working with packed words. We define the natural functions to pack and unpack words:

- unpack: $[\sqrt{n}, \perp] \to \{a,b\}^{\sqrt{n}}$ maps i to $b^{i-1} a b^{\sqrt{n}-i}$ and $\perp$ to $b^{\sqrt{n}}$. This extends naturally to *words* over $[\sqrt{n}, \perp]$.

- pack: $\{a,b\}^* \to [\sqrt{n},\bot]^*$ is the inverse of unpack. We will use that function on sets of words too, with the natural meaning.

Example 5.15.
With $n = 9$, $\mathrm{pack}(abb\,bbb\,bab) = 1\bot 2$, and $\mathrm{unpack}(31\bot) = bba\,abb\,bbb$.

We can now rephrase the notion of k-limit using packed words:

Lemma 5.16.
Let $F \subseteq \mathrm{Good}_n$ and define $\Phi = \mathrm{pack}(F)$. If μ is a packed word that has the following properties, then $\mathrm{unpack}(\mu)$ is a bad k-limit for F:

(1) There is a word $\nu \in \Phi$ that differs on a single position i with μ, at which μ has contents $\bot$:
$$\mu_i = \bot \wedge (\forall j \neq i)[\nu_j = \mu_j].$$

(2) For every set $C \subseteq [\sqrt{n}]$ of contents that contains ν_i and every set $P \subseteq [\sqrt{n}]\setminus\{i\}$ of positions such that $|C|+|P| = k$, there is a word $\lambda \in \Phi$ whose contents at position i is not in C and that matches ν on P:
$$\lambda_i \notin C \wedge (\forall p \in P)[\lambda_p = \nu_p].$$

Proof. Write u for $\mathrm{unpack}(\mu)$. That $u \in \mathrm{Bad}$ is immediate from Property 1: u is but a word v of Good in which one block was set to all b's.

We now show that u is a k-limit. Let T be a set of k positions, we split T into two sets:

- T' is the set of positions p that do not belong to the i-th block of u, that is, they do not satisfy $\lceil p/\sqrt{n}\rceil = i$. We let P be each of the elements of T' divided by $\sqrt{n}$, that is, for any $p \in T'$ we add $\lceil p/\sqrt{n}\rceil$ to P.
- T'' is the set of positions that *do* fall in the i-th block. Note that u only has b's at the positions of T''. We let C be that set, modulo $\sqrt{n}$, that is, for any $p \in T''$, we add $p \bmod \sqrt{n}$ to C or $\sqrt{n}$ if this value is 0.

First, if $\mu_i \notin C$, then T indicates positions of u that have the same letters as in $\mathrm{unpack}(\nu) \in F$, so a word of F matches u over T, as required. We thus assume next that $\mu_i \in C$.

Let $\lambda \in \Phi$ be the word given by Property 2 for C and P, we claim that $w = \mathrm{unpack}(\lambda)$ matches u on the positions of T, concluding the proof.

First note that the i-th block of w has its a in a position that is not in T'', hence w matches u on T''. Consider next any position $p \in T'$ and write j for the block in which p falls (i.e., $j = \lceil p/\sqrt{n}\rceil$). Since $\mu_j = \lambda_j$ by hypothesis, the j-th block of u and w are the same, hence $u_p = w_p$. □

5.3.2.2 Tangled sets of good words are small, nontangled ones have a bad k-limit

Consider an $F \subseteq \mathrm{Good}_n$. To find a bad k-limit for F, we need a lot of diversity in F; see in particular Prop. 2 of Lemma 5.16. Hence having some given contents at a given position in a word of Φ should not force too many other positions to have a specific value. The needed notion if the one of tangledness defined earlier. We have already seen that large sets cannot be tangled. We need to prove that sets that are not-tangled possess a k-limit.

Lemma 5.17.
Let $F \subseteq \mathrm{Good}_n$ and $\Phi = \mathrm{pack}(F)$. If Φ is not k-tangled, then F has a bad k-limit.

Proof. That Φ is not tangled means that there is a word $v \in \Phi$ and a position i such that for any set of pairs position/contents S of size k and any i-set D of size at most k that contains (i, v_i), S does not entail D. We define μ to be the word v but with μ_i set to $\perp$. We show that $\mathrm{unpack}(\mu)$ is a bad k-limit using Lemma 5.16. Property 1 therein is true by construction, so we need only show Property 2.

Let $C \subseteq [\sqrt{n}]$ with $\mu_i \in C$ and $P \subseteq [\sqrt{n}] \setminus \{i\}$ with $|C| + |P| = k$. We add some more arbitrary positions in P so that $|P| = k$, avoiding i. Define:

$$S = \{(p, \mu_p) \mid p \in P\},\ D = \{(i, c) \mid c \in C\}.$$

By hypothesis, since $(i, \mu_i) \in D$, S does not entail D. This means that there is a word $\lambda \in \Phi$ such that S and λ agree, but $\lambda_i \notin C$. This is the word needed for Property 2 of Lemma 5.16, concluding the proof. □

Corollary 5.18.
No $\Sigma_2[\mathrm{arb}]$ language can separate Good from Bad.

Proof. We apply Corollary 5.3 on any language $\mathcal{L}$ that separates Good from Bad. We let $k, d \in \mathbb{N}$, and n large enough; $\mathcal{L}'$ in the statement of Corollary 5.3 is set to Good_n. We are then given a set F of size at least $|\mathrm{Good}_n|/n^d$ and Lemma 5.8 shows that F is not tangled (the size of the alphabet of packed words being $\sqrt{n}$ and k being a constant, the assumption on the relative sizes of the alphabet and k will be satisfied with n big enough). Lemma 5.17 then implies that F has a bad k-limit, which is therefore not in $\mathcal{L}$. Corollary 5.3 concludes that F is not in $\Sigma_2[\mathrm{arb}]$, showing the statement. □

We can now state the desired theorem.

Theorem 5.19 (Neutral Straubing Property for Σ_2)**.**

$$\Sigma_2[\mathrm{arb}] \cap \mathrm{Reg} \cap \mathrm{Neut} \subseteq \Sigma_2[<].$$

Proof. Let $\mathcal{L} \notin \Sigma_2[<]$ regular with a neutral letter and $\mathcal{T}$ be the language defined in Section 5.3.1.1. Corollary 5.18 and Lemma 5.14 imply that $\mathcal{T}$ cannot be in $\Sigma_2[\mathrm{arb}]$, and therefore $\mathcal{L}$ cannot be in $\Sigma_2[\mathrm{arb}]$. □

This implies right away that the neutral Straubing property holds for Π_2 as well.

5.4 Full Straubing property

To lift the neutral letter assumption and prove the Straubing property for Σ_2 in its full generality, we need to introduce notions from finite category theory. Indeed, we need to describe $\Sigma_2[\mathrm{reg}]$.

Generic tools to handle that kind of problem have been described in Section 2.4. If $\Sigma_2[<]$ were associated to a local variety of monoids $\mathbf{V}$, we would know that $\Sigma_2[\mathrm{reg}]$ is associated to $\mathbf{QLV}$ (Fact 2.24 and Fact 2.25). However this variety of monoids is known for not being local by a result of Almeida and Escada [3, Theorem 4.17]. The use of finite category was first proposed by Tilson in [131].

5.4.1 Category theory

We present a theory of category to study varieties of monoids. There are categorical generalisations of varieties of monoids and $\mathcal{C}$-varieties of stamps. We have seen in Section 2.4 that varieties of semigroups and *ne*-varieties of stamps define the same languages. Here, for *ne*-varieties of stamps, we choose to use the concept of semigroups instead and introduce its categorical counterpart: semigroupoids. This is for a simplifying the presentation because we will not need an equivalent to *lm*-varieties of stamps. See for instance [131] for the basic definitions, and [98] for their ordered coutnerparts.

Semigroupoids and categories. A *semigroupoid* is a directed labeled (multi-)graph. However, in this setting, we use a terminology different than for graphs: a vertex is called an *object* and an edge is called an *arrow*. For C a semigroupoid, we denote by $\mathrm{Obj}(C)$ its set of objects. For x, y two object, we denote by $C(x, y)$ the set of arrows from x to y. Two arrows in the same $C(x, y)$ are said to be *coterminal*. If z is yet another object, two arrows from $C(x, y)$ and $C(y, z)$ are said to be *consecutive*. An operation on C is a function defined on pairs of consecutive arrows $e \in C(x, y)$ and $f \in C(y, z)$ that returns an arrow from $C(x, z)$. As usual, we denote the operation multiplicatively. An *identity* for an object x is an arrow $1_x \in C(x, x)$ such that $1_x e = e$ for every $e \in C(x, y)$ and $e1_x = e$ for every $e \in C(y, x)$.

Definition 5.20.
A *semigroupoid* is a ordered labeled graph with an associative operation. If moreover there is an identity for each object, we call it a *category*.

It can be seen as a generalisation of semigroups and monoids, with a partial operation. Indeed, a semigroup (resp. a monoid) can be seen as a semigroupoid (resp. a category) with a single object. The elements of the semigroup can be viewed as arrows of the semigroupoid with one object.

An *order* on a semigroupoid (resp. category) C is an order $\leq$ on coterminal arrows that is compatible is the sense that:

- if $e \leq f$ then e and f are coterminal,
- if $e \leq f$ are two coterminal arrows and e and g are consecutive then $eg \leq fg$,
- if $e \leq f$ are two coterminal arrows and g and e are consecutive then $ge \leq gf$.

A semigroupoid endowed with an order is an *ordered semigroupoid*. A category endowed with an order is an *ordered category*.

Varieties of semigroupoids and categories. A *morphism* between two semigroupoids C and D if a function f from the objects of C to the objects of D, along with, for every couple of objects x, y of C, a distinct function $g\colon C(x, y) \to C(f(x), f(y))$ such that for every consecutive arrows e, f:

$$g(xy) = g(x)g(y).$$

A morphism between two categories is a morphism of semigroupoids with the additional condition that the identity of an object is mapped to an identity. It is a morphism of ordered semigroupoids or categories if moreover it is monotone: for every arrows such that $e \leq f$, then $g(e) \leq g(f)$.

We describe some operations on semigroupoids. They can be extended to categories, ordered semigroupoids and ordered categories by adding a requirement on preservation of identities or of the order. Let C and D be two semigroupoids.

We say that C is a *quotient* of D if there exists a morphism (f, g) from C to D such that f is a bijection on the set of objects and g is surjective. We say that C is a *subsemigroupoid* of D if there exists a morphism (f, g) from C to D such that g is injective for each $C(x, y)$. We say that C *divides* D if there exists a third semigroupoid E that is a subsemigroupoid of D and such that C is a quotient of E.

The *product* of C and D is the semigroupoid $C \times D$ with:

- $\mathrm{Obj}(C \times D) = \mathrm{Obj}(C) \times \mathrm{Obj}(D)$,
- for $(x, y), (x', y') \in \mathrm{Obj}(C \times D)$, the arrows are $(C \times D)((x, y), (x', y')) = C(x, x') \times D(y, y')$.

The *coproduct* of C and D is the semigroupoid $C \vee D$ with:

- $\mathrm{Obj}(C \vee D) = \mathrm{Obj}(C)\mathrm{Obj}(D)$,
- for $(x, y) \in \mathrm{Obj}(C)$, the arrows are $(C \vee D)(x, y) = C(x, y)$,
- for $(x, y) \in \mathrm{Obj}(D)$, the arrows are $(C \vee D)(x, y) = D(x, y)$,
- for $x \in \mathrm{Obj}(C)$ and $y \in \mathrm{Obj}(D)$, the arrows are $(C \vee D)(x, y) = (C \vee D)(y, x) = \emptyset$.

Definition 5.21.
A *variety of semigroupoids* (resp. categories, ordered semigroupoids, ordered categories) is a set of semigroupoids that is closed under division, product, and coproduct.

For the case of categories and ordered categories, coproduct can be obtained as a divisor of a product. Therefore, it is not necessary to ask for the closure under coproduct. From this point, all the results will be stated with semigroupoids and semigroups but will also hold for categories and monoids.

A very important class of varieties of ordered semigroupoids is the set of globals of varieties of ordered semigroups. Let $\mathbf{V}$ be an ordered variety of semigroups, $\mathbf{gV}$ is the smallest variety of semigroupoids that contains all the semigroups in $\mathbf{V}$ (seen as one element semigroupoids). Equivalently, it is the set of ordered semigroupoids that divides a semigroup in $\mathbf{V}$. In some cases, it is be possible to deduce the decidability of membership in $\mathbf{gV}$ thanks to the decidability of the membership of $\mathbf{V}$. To do that we need to introduce the consolidated ordered semigroup $\mathrm{Cons}(C)$ of an ordered semigroupoid C. It is the ordered semigroup that simulates the operation in C, with a zero for the case of an operation applied on two non-consecutive arrows. Formally,

$$\mathrm{Cons}(C) = \{(x, e, y) \mid x, y \in \mathrm{Obj}(C), e \in C(x, y)\} \cup \{0\}$$

with the operation

$$(x, e, y) \cdot (x', e', y') = \begin{cases} (x, ee', y') \text{ if } y = x' \\ 0 \text{ otherwise} \end{cases}$$

and order given by 0 being minimal and

$$(x, e, y) \leq (x', e', y') \text{ if and only if } x = x', e \leq e', y = y'.$$

Whenever a variety of ordered semigroups is expressive enough, then we can check membership in $\mathbf{gV}$ with a condition related to $\mathbf{V}$.

Lemma 5.22 (Pin, Pinguet, Weil [98, Proposition 1.2])**.**
Let $\mathbf{V}$ be a variety of ordered semigroups (resp. monoids) that can recognise the regular language $(ab)^*$. For a semigroupoid (resp. category) C

$$C \in \mathbf{gV} \Leftrightarrow \mathrm{Cons}(C) \in \mathbf{V}.$$

Wreath product by D. We define now a category that will help in the understanding of the wreath product $*\mathbf{D}$.

Definition 5.23.
Let S be an ordered semigroup. Its *idempotent ordered category* is the category $\mathrm{ICat}(S)$ defined with

- the objects of $\mathrm{ICat}(S)$ are the idempotents of S,
- for x, y two objects, $\mathrm{ICat}(S)(x,y) = xSy$.

The operations between arrows are done with the operation of S, ie for e, f, g two idempotents, and $x \in eSf$ and $y \in fSg$ two arrows: the operation on x and y is defined as xy, adn is indeed in eSg. The order of the category is derived from the order of S.

The following theorem is known as Straubing delay theorem.

Theorem 5.24 (Straubing [127, Theorem 17.1], Pin,Pinguet,Weil [98, Theorem 3.1])**.**
Let $\mathbf{V}$ be a variety of ordered monoids. For any ordered semigroup S,

$$S \in \mathbf{V} * \mathbf{D} \Leftrightarrow \mathrm{ICat}(S) \in \mathbf{gV}.$$

The idea is to enrich the alphabet with the information of the following letters, like with a slifind window. Then a bound on the number of letters needed to be considered is needed. The core of the delay theorem is that it is enough to consider the idempotent to catch all the power of the wreath product by $\mathbf{D}$.

Wreath product by MOD. Recall that the lm-variety $\mathbf{MOD}_q$ is the lm-variety of stamps generated by the q-modular stamps, that counts the length of a word modulo an integer q. Firstly, we will introduce a categorical way of accounting for the addition of the wreath product by $\mathbf{MOD}_q$ for a fixed integer q. Secondly, we will see that doing a wreath product with the whole class $\mathbf{MOD}$ can be done only with a single $\mathbf{MOD}_q$ for a well chosen q, once a semigroup is fixed. This kind of theorems are also known as delay theorems.

Definition 5.25.
Let $\mu : A^* \to M$ be an ordered stamp, and q an integer. The *q-modulo ordered semigroupoid* of μ is the ordered semigroupoid $\mathrm{MCat}_q(\mu)$ defined with

- the objects of $\mathrm{MCat}_q(\mu)$ are integers from $\mathbb{Z}/q\mathbb{Z}$,
- for r, r' two objects, $\mathrm{MCat}_(\mu, q)r, r') = \mu((A^q)^* A^{r'-r})$.

The operations between arrows are done with the operation of M. The order of the category is derived from the order of μ.

The following is a delay theorem tailored for **MOD**.

Theorem 5.26 (Chaubard, Pin, Straubing [26])**.**
Let **V** be an ordered variety of semigroups and q be an integer. For any ordered stamp μ,

$$\mu \in \mathbf{V} * \mathbf{MOD}_q \Leftrightarrow \mathrm{MCat}_q(\mu) \in \mathbf{gV}.$$

For some cases, the stability index is the only modulo integer we need. It is in particular the case for varieties of the form $\mathbf{V} * \mathbf{D}$.

Theorem 5.27 (Dartois, Paperman [30])**.**
Let **V** be an ordered variety of monoids. Let μ be an ordered stamp of stability index s. Then we have that

$$\mu \in \mathbf{V} * \mathbf{D} * \mathbf{MOD} \Leftrightarrow \mu \in \mathbf{V} * \mathbf{D} * \mathbf{MOD}_s.$$

Proof. The theorem follows from [30], but it is not explicitly stated in that form. They introduce a property on varieties called infinite testability. Thanks to [30, Theorem 11], extended with [30, Remark 12], gives that for every variety **W** of semigroups that are infinitely testable,

$$\mu \in \mathbf{W} * \mathbf{MOD} \Leftrightarrow \mu \in \mathbf{W} * \mathbf{MOD}_s.$$

Then Remark 10 enounces that every variety of the form $\mathbf{V} * \mathbf{D}$ is infinitely testable, giving the desired result. All their proofs can be extended to the ordered setting. □

Simplification for some varieties. We call a positive variety of monoids "stable under the insertion of a minimal zero" if for every ordered monoid in the variety the following also belongs to the variety: the monoid obtained by adjoining an element that behaves as a zero and that is minimal for the order. We know that $\Pi_2[<]$ has this property, thanks to its equations $x^\omega \geq x^\omega y x^\omega$ for y a subword of x. Indeed, adding a minimal zero to a monoid already satisfying the equations also satisfies the equations: either

- $x^\omega = 0$ and the equation trivially stands since $x^\omega y x^\omega = 0$,
- $x^\omega \neq 0$ and $x^\omega y x^\omega = 0$ and the equation stands by minimality of 0,
- $x^\omega \neq 0$ and $x^\omega y x^\omega \neq 0$ and the equation stands because the base monoid satisfies the equations.

We will show that for varieties **V** of this kind and an ordered semigroupoid C, membership to **gV** of $\mathrm{ICat}(\mathrm{Cons}(C))$ and $\mathrm{ICat}(C)$ are the same.

Remark 5.28.
In Tilson's seminal paper on categories as algebras [131], Proposition 16.1 gives a stronger result (for categories) than what we will state, without restriction on the variety, and that can be extended to semigroupoids. However, we believe this result not to hold in its full generality (without breaking anything else in the paper), and we choose to use a weaker but correct statement that is enough for the case of Σ_2 and Π_2.

We first need to extend the operation ICat from semigroups to semigroupoids.

Definition 5.29.
Let C be an ordered semigroupoid. Its idempotent ordered category is the category $\mathrm{ICat}(C)$ defined with

- the objects of $\mathrm{ICat}(C)$ are pairs (x, e) where x is an object of C and $e \in C(x, x)$ is an arrow such that $e^2 = e$,
- for $(x, e), (y, f)$ two objects, $\mathrm{ICat}(C)((x, e), (y, f))$ is the set of arrows of C from x to y that can decomposed as egf for $g \in C(x, y)$.

We can state a lemma that allows to simplify checking of memberships when both consolidations and idempotent categories are present.

Lemma 5.30.
Let $\mathbf{V}$ be a variety of ordered monoids that is stable under the insertion of a minimal zero. For any ordered semigroupoid C:

$$\mathrm{ICat}(\mathrm{Cons}(C)) \in \mathbf{gV} \Leftrightarrow \mathrm{ICat}(C) \in \mathbf{gV}.$$

Proof. Inlining the definition of the idempotent and the consolidated category, we see that $\mathrm{ICat}(\mathrm{Cons}(C))$ is the same category as $\mathrm{ICat}(C)$ with an extra object 0, and an arrow labelled with 0 between every two objects. Hence $\mathrm{ICat}(C)$ is a subcategory of $\mathrm{ICat}(\mathrm{Cons}(C))$, and therefore by the transitivity of the divisibility relation:

$$\mathrm{ICat}(\mathrm{Cons}(C)) \in \mathbf{gV} \Rightarrow \mathrm{ICat}(C) \in \mathbf{gV}.$$

For the other implication, assume that $\mathrm{ICat}(C)$ divides a monoid M of $\mathbf{V}$. Let D be the subcategory of M that has a quotient into $\mathrm{ICat}(C)$. Let D^0 be the operation on D that consists in adding an object 0 and a minimal arrow 0 between any two objects. Let M^0 be M with an extra minimal zero. We know that $M^0 \in \mathbf{V}$. We can extend the quotient to a quotient from D^0 to $\mathrm{ICat}(\mathrm{Cons}(C))$. We can also see that D^0 is a subcategory of M^0. These two morphisms are defined by mapping every zero on the zero of M^0. Hence $\mathrm{ICat}(\mathrm{Cons}(C))$ divides M^0, a monoid in $\mathbf{V}$, and is therefore in $\mathbf{gV}$. □

5.4.2 Adaptation of the proof

All the category theory we need being established, we can show how to adapt the proof of the neutral Straubing property of Σ_2 to the Straubing property. In the first place, we will precisely

describe Σ_2[reg]. We redo Section 5.3.1 to show that if a language not in Σ_2[reg] is in Σ_2[arb], we can separate Good from Bad with a language in Σ_2[arb].

The *lm*-variety of stamps Σ_2[reg]. We will reduce the membership in Σ_2[reg] of a stamp to the membership in $\Sigma_2[<]$ of some well chosen monoid. Let $\mu : A^* \to M$ be an ordered stamp with a stability index s. We recall that M_s is the stabilised semigroup of μ. We define the *enriched monoid* to be

$$M^-_{reg}(\mu) = \{0\} \cup \left\{ (r, e, x, e', r') \;\middle|\; \begin{array}{l} 0 \leq r, r' < s \\ e, e' \in M_s \text{ idempotents} \\ x \in eMe' \\ x \in \mu((A^s)^* A^{r'-r}) \end{array} \right\}$$

The operation is given by

$$(r_1, e_1, x_1, e_1', r_1') \cdot (r_2, e_2, x_2, e_2', r_2') = \begin{cases} (r_1, e_1, x_1 x_2, e_2', r_2') \text{ if } e_1' = e_2 \text{ and } r_1' = r_2 \\ 0 \text{ otherwise} \end{cases}.$$

It is easy to check that x_1, x_2 comply with the conditions of the definition. The order is given by 0 being minimal and

$$(r_1, e_1, x_1, e_1', r_1') \leq (r_2, e_2, x_2, e_2', r_2') \text{ if and only if } r_1 = r_2, e_1 = e_2, e_1' = e_2', r_1' = r_2' \text{ and } x_1 \leq x_2$$

We define a slightly different enriched monoid M^+_{reg} to be the same with 0 being a maximal element. We state the following theorem for Π_2, but it would work for any logic expressive enough to define $(ab)^*$. The statement would be slightly less pleasant for varieties that are not stable under insertion of a minimal zero.

Theorem 5.31.
Let $\mathbf{V}$ be the variety of ordered monoids associated to $\Pi_2[<]$ and $\mathbf{W}$ be the *lm*-variety of ordered stamps associated to Π_2[reg]. Let μ be an ordered stamp. Then

$$\mu \in \mathbf{W} \Leftrightarrow M^-_{reg}(\mu) \in \mathbf{V}.$$

Proof. The salient feature of $M^-_{reg}(\mu)$ needed is that is it precisely equals to Cons(ICat(MCat$_s(\mu)$)), for s the stability index of μ.This is seen by inlining the definitions of both ICat and MCat.

We first express $\mathbf{W}$ with wreath products. By Theorem 2.22, that we can use thanks to Fact 2.21, we have that the variety of semigroups associated with $\Pi_2[<, \text{loc}]$ is $\mathbf{V} * \mathbf{D}$. Then using Theorem 2.23,

$$\mathbf{W} = \mathbf{V} * \mathbf{D} * \mathbf{MOD}.$$

We can use all the machinery developed in the section to obtain a chain of equivalences. The stability index of μ is written s.

The delay theorem for $*\mathbf{MOD}$ (Theorem 5.27) and the derived category theorem for $*\mathbf{MOD}$ (Theorem 5.26) give

$$\mu \in \mathbf{V} * \mathbf{D} * \mathbf{MOD} \Leftrightarrow \text{MCat}_s(\mu) \in \mathbf{g}(\mathbf{V} * \mathbf{D}).$$

We can apply Lemma 5.22, because $\Pi_2[<]$ (and therefore $\Pi_2[<, \text{loc}]$) can express $(ab)^*$ (Fact 5.9):

$$\text{MCat}_s(\mu) \in \mathbf{g}(\mathbf{V} * \mathbf{D}) \Leftrightarrow \text{Cons}(\text{MCat}_s(\mu)) \in \mathbf{V} * \mathbf{D}.$$

Straubing's delay theorem (Theorem 5.24) can now be applied:

$$\mathrm{Cons}(\mathrm{MCat}_s(\mu)) \in \mathbf{V} * \mathbf{D} \Leftrightarrow \mathrm{ICat}(\mathrm{Cons}(\mathrm{MCat}_s(\mu))) \in \mathbf{gV}.$$

With the fact that we can add a minimal zero to monoids in $\Pi_2[<]$, Lemma 5.30 gives:

$$\mathrm{ICat}(\mathrm{Cons}(\mathrm{MCat}_s(\mu))) \in \mathbf{gV} \Leftrightarrow \mathrm{ICat}(\mathrm{MCat}_s(\mu)) \in \mathbf{gV}.$$

Once again, and for the same reason as before, we can use Lemma 5.22:

$$\mathrm{ICat}(\mathrm{MCat}_s(\mu)) \in \mathbf{gV} \Leftrightarrow \mathrm{Cons}(\mathrm{ICat}(\mathrm{MCat}_s(\mu))) \in \mathbf{V}.$$

This concludes the proof. □

By duality, we deduce an analogous theorem for Σ_2 (or for any logic expressive enough to define the complement of $(ab)*$).

Corollary 5.32.
Let $\mathbf{V}$ be the variety of ordered monoids associated to $\Sigma_2[<]$ and $\mathbf{W}$ be the lm-variety of ordered stamps associated to $\Sigma_2[\mathrm{reg}]$. Let μ be an ordered stamp. Then

$$\mu \in \mathbf{W} \Leftrightarrow M^+_{reg}(\mu) \in \mathbf{V}.$$

Proof. Let $\mu : A^* \to M$ be an ordered stamp. Let $\widetilde{\mu} : A^* \to \widetilde{M}$ be the same stamp as μ but with the dual order on M. We know that the dual $\widetilde{\mathbf{V}}$ is the variety recognising $\Pi_2[<]$ and that $\widetilde{\mathbf{W}}$ is the variety recognising $\Pi_2[\mathrm{reg}]$. Therefore we can apply Theorem 5.31. The proof follows from the sequence of equivalences

$$\begin{aligned}
\mu \in \mathbf{W} &\Leftrightarrow \widetilde{\mu} \in \widetilde{\mathbf{W}} \\
&\Leftrightarrow M^-_{reg}(\widetilde{\mu}) \in \widetilde{\mathbf{V}} \\
&\Leftrightarrow \widetilde{M^+_{reg}(\mu)} \in \widetilde{\mathbf{V}} \\
&\Leftrightarrow M^+_{reg(\mu)} \in \mathbf{V}.
\end{aligned}$$

□

Separating again Good from Bad. For the rest of this section, let $\mathcal{L} \subseteq A^*$ be a regular language with a neutral letter that lies outside of $\Sigma_2[\mathrm{reg}]$ and let $\mu : A^* \to M$ be its ordered syntactic stamp. Since L is not in $\Sigma_2[\mathrm{reg}]$, Corollary 5.32 gives us that $M^+_{reg}(\mu)$ is not in $\Sigma_2[<]$. Therefore, there are elements $a = (r, e, x, e, r)$ and $b = (q, f, y, f', q')$ in $M^+_{reg}(\mu)$ such that a is an idempotent and b a subword of a, and $a \not\leq aba$. Indeed, thanks to the maximality of the zero, the equations cannot be falsified with either the right or left term being zero.

Let $\mathcal{T}$ be the language of words in M^* whose evaluation belong to the upset of x. Clearly, any word in M^* that evaluates to xyx does *not* belong to $\mathcal{T}$. Indeed, the order on $M^+_{reg}(\mu)$ is inherited from M. Because $\Sigma_2[\mathrm{reg}]$ is a positive lm-variety of languages, either both $\mathcal{T}$ and $\mathcal{L}$ belong to $\Sigma_2[\mathrm{reg}]$, or none does.

By hypothesis, b is thus also a subword of a; this provides us with words that evaluate to a and b of the shape:

$$a_1 b_1 \cdots a_t b_t a_{t+1} \text{ evaluates to } a, \quad b_1 \cdots b_t \text{ evaluates to } b,$$

where each a_i and b_i are letters in $M^+_{reg}(\mu)$. Note that we cannot use anymore an identity element to fill up the space, however we can use idempotents to ensure to have words of these forms. If we had consecutives letters in a without a letter in b in between, we can merge them into a single letter without changing the evaluations. The case with consecutive letters in b is similar. The only remaining cases are when the last letter of a is a letter of b (and the symmetric case). In this case, let $(q_t, f_t, y_t, f_t', q_t')$ be the last letter of b. We set a_{t+1} to be $q_t', f_t', f_t', f_t', q_t'$. Adding this element does not change the evaluation of a. For instance, if we had $a = a_1 b_1 a_2 a_3 b_2$ and $b = b_1 b_2$ with $b_2 = (q, f, y, f', q')$, we end up with:

$$a = a_1 b_1 (a_2 \cdot a_3) b_2 (q', f', f', f', q').$$

Because a, b and aba are not 0, and a is idempotent, we can write for $1 \leq i \leq t$:

$$\begin{aligned} b_i &= (q_i, f_i, y_i, f_i', q_i') \text{ with } q_i' = q_{i+1}, f_i' = f_{i+1} \\ a_i &= (r_i, e_i, x_i, e_i, r_i) \\ a_{t+1} &= (r_1, e_1, x_{t+1}, e_1, r_1) \text{ with } e_1 = f_1, q_1 = r_1 \end{aligned}$$

Let $n \in \mathbb{N}$ be a perfect square. We define $\sqrt{n}+1$ words of length $(t+1)\sqrt{n}+t$ over M:

- For $1 \leq i \leq \sqrt{n}$,

$$x^{(i)} = \left(e_1^{i-1} x_1 e_1^{\sqrt{n}-i} \cdot y_1\right) \cdots \left(e_t^{i-1} x_t e_t^{\sqrt{n}-i} \cdot y_t\right) \left(e_t^{i-1} x_t e_t^{\sqrt{n}-i}\right).$$

  Note that these words evaluate to x, because every x_j is in $e_j M e_j$.

- Additionally, we consider the word $e_1^{\sqrt{n}} y_1 \cdots e_t^{\sqrt{n}} y_t e_{t+1}^{\sqrt{n}}$, which evaluates to y, and we simply write y for it. Note that y can be obtained by replacing all the letters x_j by e_j from any word $x^{(i)}$.

We define $\mathcal{T}$-*good* and $\mathcal{T}$-*bad* words as in Section 5.3.1. This allows to prove an analogue to Lemma 5.14, in the case of Σ_2[reg].

Lemma 5.33.
If $\mathcal{T} \in \Sigma_2$[arb], then there is a Σ_2[arb] language that separates Good from Bad.

Proof. We design a circuit for inputs of length n over $\{a, b\}$ that separates Good from Bad.

Consider the first block of $\sqrt{n}$ letters of the input. We replicate it $t+1$ times, with the i-th replication changing b's to e_i and a's to x_i. This can be done because both x_i and e_i are in the stabilisation monoid of M and therefore can be represented with words of the same size s. We then concatenate these and add y_i between the i-th and $(i+1)$-th replication. In particular, if the block were all b's, we would obtain the word y, which has no letter x_j.

We can do this to each block of $\sqrt{n}$ letters, concatenate the resulting words, then add the word $x^{(1)}$ at the beginning and the end. Note that these operations can be done with only wires,

with no gates involved.

If the input word is in Good, then the word produced is $\mathcal{T}$-good, hence in $\mathcal{T}$. If it was in Bad, then the resulting word would be $\mathcal{T}$-bad, hence would lie outside of $\mathcal{T}$. This shows that the desired circuit can be constructed using the above wiring followed by the circuit for $\mathcal{T}$ for inputs of length $s((t+1)\sqrt{n}+t)(2+\sqrt{n})$. Since t and s are constants and depend solely on L, the resulting circuit is of polynomial size and of the correct shape. □

Reusing Corollary 5.18, we can conclude.

Theorem 5.34 (Straubing Property for Σ_2)**.**

$$\Sigma_2[\mathrm{arb}] \cap \mathrm{Reg} = \Sigma_2[\mathrm{reg}].$$

5.5 Going further

The proof we have developed in this chapter has several components:

i) understanding algebraically $\Sigma_2[<]$,
ii) construct a lower bound against the equivalent circuit model Σ_2,
iii) study the addition of regular predicates into the logic.

We would like to prove Straubing properties for Σ_i and Π_i for $i > 2$. For that, we would need all three previous points extended to higher levels of the alternation hierarchy.

i) Decidability if the membership problem for the logics $\Sigma_i[<]$ for $i \geq 5$ is not even known. Indeed, it is one of the major open problems in automata theory: the decidability of the dot-depth hierarchy. For the levels $\Sigma_3[<]$ and $\Sigma_4[<]$, their membership is decidable thanks to elaborated results of Place and Zeitoun [103] (the algorithm is in fact rather simple, its correctness is not). Even with the decidability in hand, it is not clear how to use it: a better comprehension of the equations needed to define these logics might be useful.
ii) Then one would need to develop lower bounds tailored for the circuit class Σ_i for $i \geq 3$. Indeed, the techniques against Σ_i have to not work against Σ_{i+1}. Top-down approaches, with the usage of finite limits, in the spirit of [58] are the more promising. In fact, they have been recently extended by Göös, Riazanov, Sofronova and Sokolov [46] to work against depth-4 circuits (hence against $\Sigma_3[\mathrm{arb}]$), especially for the PARITY language. It is still unclear how to use these techniques to work against other languages that we know are not in $\Sigma_3[<]$, for instance $(a(ab)^*b)^*$. It is possible to design an attack with limits for every level Σ_i, but the combinatorial problem to find limits become harder to solve as i grows: the property that has to be proved has more and more quantifiers atlernation.
iii) There is probably no particular problem with this item for any level of the alternation hierarchy. Indeed, every logic $\Sigma_i[<]$ can express $(ab)^*$ and therefore Lemma 5.22 can be applied. Also, the algebraic counterpart of each one of these logics are stable under the insertion of a maximal zero and hence we can benefit from the simplification given by Lemma 5.30. So in summary, a stamp belongs to $\Sigma_i[\mathrm{reg}]$ if and only if its enriched monoid belongs to $\Sigma_i[<]$. We thus have a good understanding of the addition of regular predicates into a logic $\Sigma_i[<]$. Lifting a neutral Straubing property to a full Straubing property will obviously depend on the precise structure of the proof, but there is a good chance that it can be achieved without too much trouble.

Bibliography of the current chapter

[3] Jorge Almeida and Ana P. Escada. "The globals of pseudovarieties of ordered semigroups containing B_2 and an application to a problem proposed by Pin". eng. In: *RAIRO - Theoretical Informatics and Applications* 39.1 (2010). DOI: 10.1051/ita:2005001.

[6] Ryan Alweiss, Shachar Lovett, Kewen Wu, and Jiapeng Zhang. "Improved bounds for the sunflower lemma". In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 2020. DOI: 10.4007/annals.2021.194.3.5.

[10] Corentin Barloy, Michael Cadilhac, Charles Paperman, and Thomas Zeume. "The Regular Languages of First-Order Logic with One Alternation". In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '22. Association for Computing Machinery, Aug. 2022. DOI: 10.1145/3531130.3533371.

[26] Laura Chaubard, Jean-Éric Pin, and Howard Straubing. "Actions, wreath products of $\mathcal{C}$-varieties and concatenation product". In: *Theoretical Computer Science*. In honour of Professor Christian Choffrut on the occasion of his 60th birthday 356.1 (May 2006). DOI: 10.1016/j.tcs.2006.01.039.

[30] Luc Dartois and Charles Paperman. "Alternation Hierarchies of First Order Logic with Regular Predicates". en. In: *Fundamentals of Computation Theory*. Ed. by Adrian Kosowski and Igor Walukiewicz. Cham: Springer International Publishing, 2015. DOI: 10.1007/978-3-319-22177-9_13.

[38] R. Erdos P.and Raso. "Intersection theorems for systems of finite sets". In: *journal of the London Mathematical Society* 35.1 (1960).

[46] Mika Göös, Artur Riazanov, Anastasia Sofronova, and Dmitry Sokolov. "Top-Down Lower Bounds for Depth-Four Circuits". In: *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. 2023. DOI: 10.1109/FOCS57990.2023.00063.

[58] J. Håstad, S. Jukna, and P. Pudlàk. "Top-down lower bounds for depth-three circuits". en. In: *Computational Complexity* 5.2 (June 1995). DOI: 10.1007/BF01268140.

[65] Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-24508-4.

[66] Stasys Jukna. *Extremal Combinatorics: With Applications in Computer Science*. 1st. Springer Publishing Company, Incorporated, 2010.

[80] Or Meir and Avi Wigderson. "Prediction from Partial Information and Hindsight, with Application to Circuit Lower Bounds". en. In: *computational complexity* 28.2 (June 2019). DOI: 10.1007/s00037-019-00177-4.

[98] Jean-Eric Pin, Arnaud Pinguet, and Pascal Weil. "Ordered categories and ordered semigroups". en. In: *Communications in Algebra* 30.12 (Dec. 2002). DOI: 10.1081/AGB-120016004.

[103] Thomas Place and Marc Zeitoun. "Going Higher in First-Order Quantifier Alternation Hierarchies on Words". In: *Journal of the ACM* 66.2 (Mar. 2019). DOI: 10.1145/3303991.

[107] Anup Rao. *Coding for Sunflowers*. Feb. 2020. DOI: 10.48550/arXiv.1909.04774.

[120] Michael Sipser. "A topological view of some problems in complexity theory". en. In: *Mathematical Foundations of Computer Science 1984*. Ed. by M. P. Chytil and V. Koubek. Vol. 176. Berlin/Heidelberg: Springer-Verlag, 1984. DOI: 10.1007/BFb0030341.

[127] Howard Straubing. "Finite semigroup varieties of the form V * D". en. In: *Journal of Pure and Applied Algebra* 36 (Jan. 1985). DOI: 10.1016/0022-4049(85)90062-3.

[131] Bret Tilson. "Categories as algebra: An essential ingredient in the theory of monoids". en. In: *Journal of Pure and Applied Algebra* 48.1 (Sept. 1987). DOI: 10.1016/0022-4049(87)90108-3.

Chapter 6

Streaming Complexity: Processing Regular Properties of XML Documents

Outline of the current chapter

This chapter is based on the paper "Stackless Processing of Streamed Trees", which is joint work with Filip Murlak and Charles Paperman [11]. The results of Section 6.5 is new.

Context. While graph is the new black, tree-structured data has not vanished. It is used both as a serialisation format (Wikipedia, Wikidata, DBLP) and as an exchange format (WSDL and SOAP rely on XML, the more recent GraphQL prefers JSON). Querying and validation of tree-structured data continue to be both vital and challenging tasks in data management. Particularly so, when documents grow too large to fit in memory, and it is time to switch to streaming; that is, to read the document sequentially, maintaining a concise internal representation sufficient for the realised task.

According to Palkar, Abuzaid, Bailis, and Zaharia [85], exploratory big-data applications running over data in a semi-structured format, like JSON, can spend 80-90% of their execution

time simply parsing the data. Performance improvements often rely on clever ways to reduce the cost of parsing. In systems research, two main strategies have been proposed. The first one relies on SAX (*Simple API for XML*) parsers: it outsources parsing to the API and deals only with the resulting events [54, 128]. This allows to factor out the cost of parsing, and may lead to significant performance gains when multiple queries are executed over the same document [128]. The second approach is to perform parsing and query execution simultaneously, applying push-down automata as the computation model [84], in the hope that the acquired semantic information would help reduce the cost of parsing. When a single query is executed over a huge document, this may also be highly beneficial [31].

The theoretical take on alleviating the cost of parsing is more radical: since it is so costly, let us assume that it has been already done for us and the input stream is guaranteed to be a well-formed document. This may be the case, for instance, if we trust the source of the document or if we have already processed the document for other purposes. Can this assumption help process the document more efficiently? This setting was introduced as *weak validation* in the seminal work of Segoufin and Vianu [116] on validating a streamed XML document against a DTD by means of a finite automaton. Despite the significant progress made in the initial paper and in the follow-up work [9, 27, 115], the general problem of deciding whether weak validation against a given regular tree language is feasible, remains open. Incidentally, this question is a special—but disturbingly generic—case of an undecidable separation problem [70].

Vectorisation. A recent trend in data processing is to use hardware acceleration to exploit *local parallelism*. Most modern CPU architectures offer SIMD (*single instruction multiple data*) instructions, allowing to perform the same operation on multiple data points in one CPU cycle, leading to what is known as the *Vectorisation* of computation. Vectorisation is used routinely in data-intensive applications like multimedia processing [110] or deep learning [44, 135], and is finding its way to data management, particularly in the sub-field of in-memory databases [142, 105]. Relevant examples from a related field are the performant regular expression engine *Hyperscan* [137] and the competitive engine of the RUST language [53], both relying crucially on Vectorisation. In the context of streaming processing of tree-structured data, an early work on *parabix* by Cameron et al. studies the use of SIMD instructions to accelerate XML parsing [25]. More recently, Langdale and Lemire illustrate how the performance of JSON parsers could be vastly improved by using Vectorisation [75]. Their experiments confirm that the cost of parsing is a large fraction of the total cost of query execution, matching the performance loss with respect to regular expression matching. To get a better feeling of the room for improvement, let us look at some numbers: the experiments had different setups, but the orders of magnitude are still of interest. The standard C function MEMCHR scans memory to find the first occurrence of a given byte; it has been hand-optimised for various architectures and can be assumed to display the best performance one could hope for in a streaming task. On a standard laptop computer, it easily reaches 20Gb/s. The *Hyperscan* regular expression engine reaches performance of 10Gb/s [137]. Langdale et al. get up to 3Gb/s when parsing JSON files and selecting some nodes, but selecting alone reaches 10Gb/s [75]. Palkar et al. explicitly put the blame on the incompatibility of pushdown automata and Vectorisation [85].

To some extent this is explained by theory. An abstract model of exploiting local parallelism in streaming algorithms was proposed in [82]: the stream is read in blocks, each block is processed by a fixed Boolean circuit, and the result is fed back to the circuit together with the next block of the stream. The degree of local parallelism of a language is measured by the complexity of the circuit needed to recognise the language in the above model: the higher the complexity, the less local parallelism. As shown in the paper, the degree of local parallelism of regular languages matches their classical circuit complexity, and it is plausible that the situation is similar for

larger classes of languages. Assuming this is the case, successful Vectorisation of XML or JSON parsers might be more tricky than for regular expression engines: Dyck languages (well-formed multi-bracket expressions) are TC^0-complete [15], but while regular languages may have even higher complexity, the ones appearing in benchmarks are typically much simpler (for instance, all examples in [53] and [141] are in AC^0).

All this evidences that stack-based computation is troublesome. At the same time falling back to finite automata severely limits expressivity, as revealed by the necessary conditions discovered by Segoufin and Vianu [116]. As a middle ground, we propose to relax the computational model just so. We allow one counter for maintaining the current depth in the document, and registers for storing the current depth to be compared with the depths of later tags. In the resulting model, dubbed *depth-register automaton*, transitions are performed at a very low CPU cost with almost no external memory access. The latter depends on the number of registers; if the number is low enough, it is even possible to keep all the values within the CPU's registers and not use external memory at all. Unlike pushdown automata, the model is amenable to Vectorisation and can achieve high throughputs. Indeed, it has been successfully implemented by Gienieczko, Murlak and Paperman [45] with their tool RsonPath. They can query a large fragment of JSONPath with impressive performances, with strategies to skip part of a document.

Regular languages. To understand the power of (weakly) validating and querying streamed tree with the means of finite-states machines, we propose to continue the work of Segoufin and Vianu [116]. The goal is to explicit all the regular tree languages that can be validated in streaming efficiently: ie. by a finite word automaton (or a depth-register automaton) being fed by markup encodings of trees. Unfortunately, this is a hard task and we fall back to subclasses of regular tree languages that are easier to handle. These restrictions are about languages of trees that can check the form of the branches of a tree, but not their relations. It allows to apply techniques from the world of words, which are better understood. In the end, we will extend the results to the less verbose term encoding of trees, and we will draw a bridge with algebraic classes.

6.1 Weak validation

From this point, and until the term encoding is considered in Section 6.4, encoding will be silently referring to markup encoding.

The *validation problem* is the problem parameterised by a regular tree language $\mathcal{L}$ defined by:

- INPUT: a word $w \in A \cup \bar{A}$.
- OUPUT: whether w is the markup encoding of a tree belonging in $\mathcal{L}$.

Note that an algorithm for one of these problems has in particular to check that the word w is a valid encoding of a tree. We want to find efficient algorithms to solve these problems, and identify for which regular tree languages it is easy or complicated. In particular, we are interested in the class of regular tree languages that are validatable in constant space, that is to say the languages $\mathcal{L}$ such that $\langle\mathcal{L}\rangle$ is recognised by a finite automaton. Not many languages are in this class, for instance the regular languages of trees over $\{a\}$ with a single leaf is not validatable in constant space. Indeed, its set of encodings is $\{a^n\bar{a}^n \mid n \in \mathbb{N}\}$, which is well known for its non-regularity. This class have been precisely described by Segoufin and Vianu, using the framework of (specialised) DTDs.

The graph G_d of a DTD d over A is the graph whose set of vertices is A and whose edges are the couples (a, b) such that there is a rule $a \to \mathcal{L}$ where b is a letter of a word in $\mathcal{L}$. A node is

recursive if it belongs to some cycle. A DTD is *non-recursive* if its graph G_d is acyclic. We extend this definition to specialised DTDs $d = (A', d', \mu)$ by asking that d' is non-recursive.

Theorem 6.1 (Segoufin, Vianu [116, Theorem 3.1])**.**
A specialised DTD is validatable in constant space if and only if it is non-recursive.

It implies in particular that a tree language with trees of unbounded depth can never be validated in constant space.

Hence, the problem of *weak validation* is more interesting. It is the validation problem, with the additional assumption that the word w is *valid*, in the sense that it is the encoding of some tree. In the case of constant space, the weak validation problem for $\mathcal{L}$ can be reformulated as a separation problem: is there a regular language disjoint from $\langle\mathcal{L}\rangle$ that contains $\langle\mathcal{L}^c\rangle$? We anticipate the introduction of the stackless model, and call *registerless* the languages that are weakly validatable in constant space. The previous example of trees with a single leaf is now registerless, as a separator is $a^*\bar{a}^*$. Identifying the regular tree languages that are registerless is challenging and still open. This question can though be answered for DTDs of a certain shape. A DTD d is *fully recursive* if all the nodes of G_d that can reach a recursive node are in the same strongly connected component.

Theorem 6.2 (Segoufin, Vianu [116, Theorem 4.2])**.**
It is decidable whether a fully recursive specialised DTD is weakly validatable in constant space.

We give an algebraic characterisation, based on forest algebras defined in Section 2.6, of the regular tree languages that are registerless. Unfortunately, it will not be enough to *a priori* give decidability.

We define the class of forest algebras such that the vertical action can be seen simulated in the horizontal monoid.

Definition 6.3.
We say that a forest algebra (H, V) is *horizontal* if for every $v \in V$, there exists $x_v, y_v \in H$ such that for every $h \in H$

$$v \cdot h = x_v + h + y_v.$$

This class captures precisely the languages that are registerless.

Theorem 6.4.
A tree language $\mathcal{L}$ is weakly validatable in constant space if and only if it is recognised by an horizontal forest algebra.

Proof. Assume that $\mathcal{L}$ is weakly validatable in constant space. Let $\mathcal{A}$ be an automaton that accepts all word in $\langle\mathcal{L}\rangle$ and rejects all words in $\langle\mathcal{L}^c\rangle$. Let $(M, +, 0)$ be the transition monoid of $\mathcal{A}$, that we denote additively here. We will also need the natural morphism $\mu : (A \cup \bar{A})^* \to M$. We

define a forest algebra with $H = M$ and $V = M^2$ with the operation $(x,y)\cdot(x',y') = (x+x', y'+y)$. The action is defined as $h \mapsto x+h+y$, for $(x,y) \in V$. For $h \in H$, the operations $in_l(h)$ and $in_r(h)$ are respectively defined as $((h,0))$ and $(0,h)$. The accepting subset of H is the accepting subset of M. This forest algebra is indeed finite and horizontal by definition. It can recognise $\mathcal{L}$ with the morphism ν that associates $(\mu(a), \mu(\bar{a})) \in V$ to a seen as a context. We want to show by induction that for every forest t, $\nu(t) = \mu(\langle t \rangle)$. For the base case, notice that both the empty forest and the empty word are mapped to the identity of M. Assume that $t = t_1 + t_2$ such that the result is true for t_1 and t_2. Then $\nu(t) = \nu(t_1)+\nu(t_2) = \mu(\langle t_1\rangle)+\mu(\langle t_2\rangle) = \mu(\langle t_1\rangle\langle t_2\rangle) = \mu(\langle t\rangle)$. Assume that $t = a(t')$ such that the result is true for t'. Then $\nu(a(t')) = \nu(a)(\nu(t')) = \mu(a)+\mu(t')+\mu(\bar{a}) = \mu(a\langle t'\rangle\bar{a}) = \mu(\langle t\rangle)$. Finally, t is accepted by (H,V) if and only if $\langle t\rangle$ is in $\mathcal{L}$.

Now assume that $\mathcal{L}$ is recognised by an horizontal forest algebra (H,V) and a morphism ν. For every letter a, by the horizontal property, there exist $x_a, y_a \in H$ such that for every $h \in H$, $a(h) = x_a + h + y_a$. We separate $\langle\mathcal{L}\rangle$ and $\langle\mathcal{L}^c\rangle$ with the monoid H and the morphism $\mu : (A \cup \bar{A})^* \to H$ defined by $\mu(a) = x_a$ and $\mu(\bar{a}) = y_a$. The accepting subset is the same as the one in the forest algebra. With the same computations as before, we prove that for every tree t, $\nu(t) = \mu(\langle t\rangle)$.

□

Notice that we are not assuming regularity, hence the theorem gives that only regular languages can be weakly validated in constant space. Unfortunately, being horizontal is not a property that can be passed to a subalgebra. Indeed, there are no reasons for x_v and y_v to remain in the subalgebra. Hence the set of all horizontal forest algebras is not a variety. This forbid us from having a statement that identifies the registerless languages with those with an horizontal *syntactic* forest algebra. Therefore we have to work with a subclass of regular tree languages. The languages we consider are tied with regular word languages.

Definition 6.5.
Let $\mathcal{L}$ be a regular word language. Then $E\mathcal{L}$ is the regular language of trees that have a path from the root to a leaf labelled with a word in $\mathcal{L}$. Dually $A\mathcal{L}$ is the regular language of trees such that all paths from the root to a leaf are labelled with a word in $\mathcal{L}$.

Note that $(A\mathcal{L})^c = E(\mathcal{L}^c)$. Languages of the form $A\mathcal{L}$ can express useful and nontrivial schema restrictions, as they are able to specify which labels are allowed in the children of a node, depending on regular properties of the path from the root.

So far we used automata as acceptors, defining languages of trees. However, we can also use them as node selectors, defining queries over trees. By a *query Q of arity k* we mean a function mapping each tree T to a set $Q(T)$ of k-tuples of nodes of T. In the streaming setting, higher-arity queries are problematic because a streaming algorithm using memory of size $f(n)$ over inputs of length n cannot return asymptotically more than $f(n)\cdot n$ answers. This means that handling even very simple queries of arity larger than one in sublinear memory is impossible without compromising the semantics by applying restrictive *selection strategies* [140, 50] or heuristics like *load shedding* [60]. Moreover, popular query languages for tree-structured data, like XPath or JSONPath, focus on unary queries. We shall do the same.

Implementations of unary queries over streamed trees come in two distinct flavours, corresponding to the two natural moments when one may wish the selected nodes to be returned: at the opening tag or at the closing tag. Accordingly, we say that an automaton $\mathcal{A}$ *pre-selects* (resp. *post-selects*) a node v of a tree T if $\mathcal{A}$ is in an accepting state directly after reading the opening (resp. closing) tag of v. Both approaches have their merits. Post-selection gives more expressive power, allowing to explore the subtree rooted at the given node. Pre-selection gives

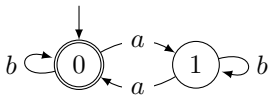


Figure 6.1: A reversible finite automaton.

more flexibility in the subsequent stages of processing, allowing to return the whole subtree rooted at the selected node without additional memory cost. Here we focus on pre-selection, and leave post-selection for the future. Accordingly, we say that an automaton $\mathcal{A}$ *realises* a unary query Q if for every tree T, $\mathcal{A}$ pre-selects exactly those nodes of T that belong to $Q(T)$. We call a unary query Q registerless if it can be realised by a finite automaton, that outputs a node whenever an accepting state is reached.

Definition 6.6.
Let $\mathcal{L}$ be a regular word language. Then $Q\mathcal{L}$ is the query that selects all nodes v such that the path from the root to v is labelled by a word from $\mathcal{L}$.

We call queries of this form *regular path queries* (RPQs). They include all XPath queries built up from downward axes (child, descendent) and label tests, but not those using upward axes (parent, ancestor) or filters.

6.2 Registerless languages

We give here a combinatorial description of the path languages and RPQs that are registerless. We first focus on the more symmetric setting of querying, then we embark on the study of $A\mathcal{L}$ and $E\mathcal{L}$.

6.2.1 Almost-reversibility

How does one go about evaluating an RPQ with a finite automaton reading the markup encoding of a tree? Over the *leftmost* branch this is easy: as long as only opening tags are read, we simulate the automaton underlying the RPQ over the labels in the tags and accept whenever the simulated automaton accepts. When the first closing tag appears, the simulated automaton should revert to the state before reading the corresponding opening tag. Our simulation could store a bounded suffix of the run of the simulated automaton, and use it when closing tags occur, but what shall we do when it is used up? This is clearly not a sustainable strategy. The task does become feasible if we assume that the previous state can be determined based on the current state and the last read letter. Automata that have this property are called *reversible*.

Recall that in a deterministic automaton letters induce functions mapping states to states. A deterministic automaton is *reversible* if every letter induces an injective function (Fig. 6.1). Equivalently, one may assume that letters induce permutations of states, which implies that the associated transition monoid is a group. Reversibility can be studied as a separate notion upon extension to incomplete automata, where letters induce partial functions over states [96].

The simulation above captures RPQs given by reversible automata, but we can do a bit more. Consider the automata depicted in Fig. 6.2. None of them is reversible because the function induced by the letter a is not injective. However, the automaton in Fig. 6.2a defines a registerless

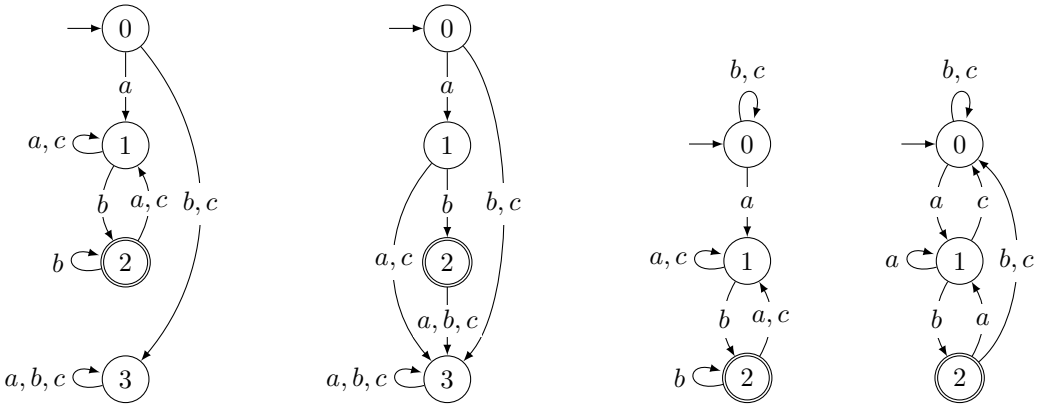


Figure 6.2: Languages of increasing hardness over $A = \{a, b, c\}$.

RPQ. Indeed, the realising finite automaton should check that the first opening tag has label a and then it should accept at each opening tag with label b. Those in Figs. 6.2b to 6.2d are not registerless and it will follow from our characterisation theorems.

In order to capture registerless RPQs precisely, we carefully relax the notion of reversibility. Unlike reversibility itself, its relaxed variant is dependent on which states are accepting. Let us fix a deterministic automaton $\mathcal{A}$. We say that states p and q are *equivalent* if for every word w, $p \cdot w \in F$ iff $q \cdot w \in F$. In a minimal automaton, equivalent states are equal. We say that states p and q are *almost equivalent* if for every non-empty word w, $p \cdot w \in F$ iff $q \cdot w \in F$. That is, non-empty words do not distinguish almost equivalent states; it follows immediately that after reading any letter the states become indistinguishable.

Lemma 6.7.
If states p and q are almost equivalent, then for each letter a, the states $p \cdot a$ and $q \cdot a$ are equivalent.

We shall call a state p of automaton $\mathcal{A}$ *internal* if it is reachable from the initial state via a nonempty word. Note that if all states are reachable, only the initial state can be non-internal, and it happens only iff it has no incoming transitions.

Definition 6.8.
We say that states p and q *meet in state* r if there exists a word u such that $p \cdot u = q \cdot u = r$; we say p and q *meet* if they meet in some state r. A deterministic automaton is *almost-reversible* if every two internal states that meet are almost equivalent. We call a regular language almost-reversible if its minimal automaton is almost-reversible.

As intended, the automaton in Fig. 6.2a is almost-reversible, while those in Figs. 6.2b to 6.2d are not.

It will be useful to have a seemingly restricted notion of almost-reversibility, to have witnesses of non almost-reversibility of a special form.

Lemma 6.9.
A minimal deterministic automaton is almost-reversible if and only if every two internal states p and q that meet in q are almost equivalent.

Proof. This inclusion from left to right is clear. For the other direction, we assume that $\mathcal{A}$ is not almost-reversible: there exists two internal states p, q such that p and q meet and are not almost-equivalent. In particular p and q are distinct.

We call a SCC X in $\mathcal{A}$ a sink if for each $q \in X$ and each $u \in A^*$, $q \cdot u \in X$. We know that p and q meet in a sink SCC X: there exists $r \in X$ and $u \in A^*$ such that $p \cdot u = q \cdot u = r$. We want to find a state s in X such that p and s meet in s, and q and s meet in s. Because X is a sink, $p \cdot u^n \in X$ for all $n > 0$. Consequently, there exist $n, k > 0$ such that $p \cdot u^n = p \cdot u^n \cdot u^k$. Moving n positions backwards in the cyclic list of states $p \cdot u^n, p \cdot u^{n+1}, \ldots, p \cdot u^{n+k-1}$, starting from $p \cdot u^n$, we find a state $s = p \cdot u^{n+k-n \bmod k} \in X$ that meets with p. Because X is a sink, p and s can only meet in X. But then p and s also meet in s. Remark that p and s meet with a word of the form u^k, thus $q \cdot u^k = p \cdot u^k = s \cdot u^k$. Now one of the couple (p, s) or (q, s) has to be non almost-equivalent, otherwise the transitivity of the relation would imply that p and q are almost equivalent. □

We can now state the main theorem of the section.

Theorem 6.10.
Let $\mathcal{L}$ be a regular language.
Then $Q\mathcal{L}$ is a registerless query if and only if $\mathcal{L}$ is almost-reversible.

We will prove both directions.

Lemma 6.11.
If $\mathcal{L}$ is an almost-reversible language, then $Q\mathcal{L}$ is a registerless query.

Proof. Let $\mathcal{A}$ be the minimal automaton of $\mathcal{L}$. The simulating automaton $\mathcal{B}$ will use the same states as $\mathcal{A}$ together with an additional rejecting sink state $\perp$; the initial state and the set of accepting states are also like in $\mathcal{A}$. When reading opening tags, $\mathcal{B}$ follows the transition relation of $\mathcal{A}$. Upon reading a closing tag $\bar{a}$ in a state p, $\mathcal{B}$ moves to some internal p' in $\mathcal{A}$ such that $p' \cdot a$ is almost equivalent to p. To keep $\mathcal{B}$ deterministic, we take the minimal such p' according to an arbitrarily chosen order on the states of $\mathcal{A}$. If such a state p' does not exists, $\mathcal{B}$ moves to $\perp$.

Consider an input tree T. For each prefix w of $\langle T \rangle$, let $\widehat{w}$ be the word obtained from w by successively erasing all two-letter subwords of the form $a\bar{a}$ for $a \in A$. If w ends with the opening tag of a node x in T, then $\widehat{w}$ is the sequence of labels on the shortest path from the root of T to x. If w ends with the closing tag of a node x in T, then $\widehat{w}$ is the sequence of labels on the shortest path from the root of T to the parent of x (if x is the root of T, then the path is empty). We claim that for every proper nonempty prefix w of $\langle T \rangle$, the state p_w of $\mathcal{B}$ after reading w is a an internal state of $\mathcal{A}$ that is almost equivalent to the state $q_{\widehat{w}}$ of $\mathcal{A}$ after reading $\widehat{w}$, and if the last letter of w is an opening tag, then $p_w = q_{\widehat{w}}$. The claim immediately implies that $\mathcal{B}$ realises $Q\mathcal{L}$, because the first and the last state of $\mathcal{B}$ in the run on $\langle T \rangle$ does not matter.

We prove the claim by induction on $|w|$. The automaton $\mathcal{B}$ begins the computation in the

(a) Tree S. (b) Tree S'.

Figure 6.3: Fooling trees in Lemma 6.12.

initial state of $\mathcal{A}$. The first letter of $\langle T \rangle$ is some opening tag a. Because $\widehat{a} = a$, we have $p_a = q_{\widehat{a}}$ and p_a is clearly internal. Suppose now that the claim holds for w. If the next letter after w is an opening tag c, applying Lemma 6.7 to the almost equivalent states p_w and $q_{\widehat{w}}$ of $\mathcal{A}$, we get $p_{wc} = p_w \cdot c = q_{\widehat{w}} \cdot c = q_{\widehat{wc}}$, and we are done because $q_{\widehat{w}} \cdot c$ is clearly internal. Suppose that the next letter read by $\mathcal{B}$ is a closing tag $\bar{c}$. We need to prove that there exists an internal state p' in $\mathcal{A}$ such that $p' \cdot c$ is almost equivalent to p_w, and that every such p' is almost equivalent to $q_{\widehat{w\bar{c}}}$. Consider $p' = q_{\widehat{w\bar{c}}}$. Because $w\bar{c}$ is a proper prefix of $\langle T \rangle$, the word $\widehat{w\bar{c}}$ is nonempty; hence, $q_{\widehat{w\bar{c}}}$ is a internal state of $\mathcal{A}$. We also have $q_{\widehat{w\bar{c}}} \cdot c = q_{\widehat{w}}$, and we have assumed that p_w and $q_{\widehat{w}}$ are almost equivalent; hence, $q_{\widehat{w\bar{c}}} \cdot c$ is almost equivalent to p_w. So, indeed, $q_{\widehat{w\bar{c}}}$ is a correct choice for p'. Let us now take any internal p' with $p' \cdot c$ almost equivalent to p_w, and prove that p' is almost equivalent to $q_{\widehat{w\bar{c}}}$. As p_w and $q_{\widehat{w}}$ are almost equivalent by the induction hypothesis, it follows that so are $p' \cdot c$ and $q_{\widehat{w}}$. By Lemma 6.7, $p' \cdot c \cdot b = q_{\widehat{w}} \cdot b = q_{\widehat{w\bar{c}}} \cdot c \cdot b$ for each $b \in A$. Hence, p' and $q_{\widehat{w\bar{c}}}$ meet. We have already argued that $q_{\widehat{w\bar{c}}}$ is internal, and p' is internal by assumption. Because $\mathcal{A}$ is almost-reversible, we conclude that p' is almost equivalent to $q_{\widehat{w\bar{c}}}$. □

The other direction is proved by pumping simultaneously at the level of trees and their encodings, which resembles pumping arguments for context free grammars. To simplify factorising encodings of trees, for a word $w = a_1 a_2 \cdots a_n \in A^*$ we let $\bar{w} = \bar{a}_n \cdots \bar{a}_2 \bar{a}_1$ (note the reversed order). Consider the tree S shown in Fig. 6.3a, keeping in mind that s, t, u, x are words rather than single letters: each node labelled with a word w represents a chain of $|w|$ nodes whose labels form the word w. Then,

$$\langle S \rangle = s u^{n!} \bar{u}^{n!} t \bar{t} u^{n!} \bar{u}^{n!} \bar{s}.$$

We use S in the proof of the following lemma.

Lemma 6.12.
For a regular language $\mathcal{L}$, if $Q\mathcal{L}$ is a registerless query, then $\mathcal{L}$ is almost-reversible.

Proof. Suppose that the minimal automaton $\mathcal{A}$ of $\mathcal{L} \subseteq A^*$ is not almost-reversible. We use the alternative definition from Lemma 6.9. Let i be the initial state of $\mathcal{A}$. Then, there exist words $s, t, u \in A^+$ and states p, q such that $i \cdot s = p$, $p \cdot u = q \cdot u = q$ and $p \cdot t$ is accepting iff $q \cdot t$ is rejecting. It follows that for each $k > 0$, $st \in \mathcal{L}$ iff $su^k t \in \mathcal{L}^c$.

Consider a deterministic finite automaton $\mathcal{B}$ over $A \cup \bar{A}$ with n states. It is well known that $r \cdot w^{n!} = r \cdot w^{2 \cdot n!}$ for each nonempty word w and each state r of $\mathcal{B}$.

Consider the trees S and S' shown in Fig. 6.3. By the discussion above, the node t is selected in exactly one of those trees by $Q\mathcal{L}$. Consider the runs of $\mathcal{B}$ on $\langle S \rangle$ and $\langle S' \rangle$. Suppose that on $\langle S \rangle$ we have

$$q_0 \xrightarrow{s u^{n!}} q_1 \xrightarrow{\bar{u}^{n!} \cdot t} q_2 \xrightarrow{\bar{t} \cdot u^{n!} \bar{u}^{n!}} q_3 \xrightarrow{\bar{s}} q_4.$$

Then, by the choice of n, we have

$$q_0 \xrightarrow{su^{n!}} q_1 \xrightarrow{u^{n!}} q_1 \xrightarrow{\bar{u}^{n!}\cdot t} q_2 \xrightarrow{\bar{t}\cdot u^{n!}\bar{u}^{n!}} q_3 \xrightarrow{\bar{u}^{n!}} q_3 \xrightarrow{\bar{s}} q_4 .$$

It follows that $\mathcal{B}$ selects t in $\langle S\rangle$ iff it selects t in $\langle S'\rangle$. Consequently, $\mathcal{B}$ does not implement $Q\mathcal{L}$. □

6.2.2 Flatness

Not all finite languages are almost-reversible, as witnessed by the one in Fig. 6.2b. Nevertheless, if $\mathcal{L}$ is finite, then $A\mathcal{L}$ is registerless. Indeed, a finite automaton can simply simulate the stack up to the depth bounded by the length of the longest word in $\mathcal{L}$. If an opening tag is read when the stack is at its maximum depth, the automaton moves to an all-rejecting sink state. Symmetrically, if $\mathcal{L}$ is co-finite (that is, $\mathcal{L}^c$ is finite), then $E\mathcal{L}$ is registerless. This motivates the following dual notions.

Definition 6.13.
We call a state q *acceptive* (resp. *rejective*) if $q\cdot w$ is accepting (resp. rejecting) for some $w \in A^*$. A deterministic automaton is *E-flat* (resp. *A-flat*) if for every internal state p and every rejective (resp. acceptive) state q, if p meets with q in q, then p is almost equivalent to q. A *E-flat* (resp. *A-flat*) *language* is a regular language whose minimal automaton is E-flat (resp. A-flat).

Checking that all finite languages (including the one in Fig. 6.2b) are A-flat, and all co-finite ones are E-flat is an easy exercise. The following lemma, connecting flatness to almost-reversibility is not hard either.

Lemma 6.14.
Let $\mathcal{L} \subseteq A^*$ be a regular language.

i) $\mathcal{L}$ is A-flat iff $\mathcal{L}^c$ is E-flat.
ii) $\mathcal{L}$ is almost-reversible iff it is both A-flat and E-flat.

Proof. Let $\mathcal{A}$ be the minimal automaton of $\mathcal{L}$. Then $\mathcal{A}^c$, obtained from $\mathcal{A}$ by swapping accepting and rejecting states, is the minimal automaton of $\mathcal{L}^c$. A state q is acceptive in $\mathcal{A}$ iff it is rejective in $\mathcal{A}^c$. It follows that $\mathcal{A}$ is A-flat iff $\mathcal{A}^c$ is E-flat.

The second part is immediate with Lemma 6.9. □

We can state our characterisation theorem in the validation setting.

Theorem 6.15.
Let $\mathcal{L}$ be a regular language, then

- $E\mathcal{L}$ is registerless iff $\mathcal{L}$ is E-flat.
- $A\mathcal{L}$ is registerless iff $\mathcal{L}$ is A-flat.

We prove both direction successively. More effort is needed to show that E-flatness of $\mathcal{L}$ is sufficient to simulate its minimal automaton faithfully enough to support recognising $E\mathcal{L}$.

Lemma 6.16.
If $\mathcal{L}$ is an E-flat language, then $E\mathcal{L}$ is a registerless tree language.

Proof. Let $\mathcal{A}$ be the minimal automaton of $\mathcal{L}$. We first construct an automaton $\mathcal{B}$ simulating $\mathcal{A}$ in a certain precise sense, and then we turn $\mathcal{B}$ into an automaton recognising EL.

Like in the simulation of almost-reversible automata, the high-level idea is to maintain the state of $\mathcal{A}$ after processing $\widehat{w}$ up to almost equivalence, except that if at any point the maintained state becomes non-rejective, the simulating automaton moves to an all-accepting sink state $\top$. But because the internal structure of E-flat automata is much richer then that of almost-reversible ones, the simulating automaton $\mathcal{B}$ needs more information.

After reading a prefix w of the encoding of the input tree, the simulating automaton $\mathcal{B}$ will store a *synopsis* of the run of $\mathcal{A}$ on $\widehat{w}$. The goal of the synopsis is to list the transitions that moved the run from one SCC of $\mathcal{A}$ to the next one. However, because the automaton $\mathcal{A}$ is not reversible, taking the transitions backwards when processing closing tags will introduce certain ambiguity into the stored transitions. Namely, the origins of the transitions will be *split states*, defined as pairs (p,q) such that q is rejective and either $p = q$ or p is internal and meets with q in q. E-flatness guarantees that for each split state (p,q), the states p and q are almost equivalent. By minimality, transitions from split states have unambiguous targets.

A *split transition* is a tuple (p,q,a,r) such that (p,q) is a split state and $p \cdot a = q \cdot a = r$. A *synopsis* for $\mathcal{A}$ is an alternating sequence of state triples and letters, written as

$$(r_0,p_0,q_0) \xrightarrow{a_1} (r_1,p_1,q_1) \xrightarrow{a_2} \cdots \xrightarrow{a_\ell} (r_\ell,p_\ell,q_\ell), \tag{6.1}$$

such that r_0 is the initial state of $\mathcal{A}$, each $(p_i,q_i,a_{i+1},r_{i+1})$ is a split transition in $\mathcal{A}$, (p_ℓ,q_ℓ) is a split state in $\mathcal{A}$, and

- for each $i < \ell$, the states q_i and r_{i+1} are in different SCCs;
- for each $i \leq \ell$, q_i belongs to the SCC of r_i and either p_i belongs to the SCC of r_i or $i > 0$ and $p_i = p_{i-1} = q_{i-1}$.

Observe that the states q_i represent a chain of different SCCs, so $\ell + 1$ is bounded by the depth of the DAG of SCCs of $\mathcal{A}$.

The empty word ε is *compatible* only with synopses (r_0,p_0,q_0) with $r_0 \in \{p_0,q_0\}$. For $u \in A^*$ and $a \in A$, the word ua is *compatible* with a synopsis σ of the form (6.1) if $r_0 \cdot ua \in \{p_\ell,q_\ell\}$ and one of the following holds:

(a) $r_0 \cdot u$ is in the SCC of $r_0 \cdot ua$, and u is compatible with the synopsis obtained from σ by replacing (r_ℓ,p_ℓ,q_ℓ) with $(r_\ell, r_0 \cdot u, r_0 \cdot u)$;
(b) $\ell > 0$, $r_0 \cdot u \in \{p_{\ell-1},q_{\ell-1}\}$, $a = a_\ell$, and u is compatible with the synopsis obtained from σ by removing the suffix $\xrightarrow{a_\ell} (r_\ell,p_\ell,q_\ell)$;
(c) $\ell > 0$, $r_0 \cdot ua = p_\ell = p_{\ell-1} = q_{\ell-1}$, and ua is compatible with the synopsis obtained from σ by removing the suffix $\xrightarrow{a_\ell} (r_\ell,p_\ell,q_\ell)$.

Note that if some u is compatible with σ and $r_0 \cdot u = p_\ell$, then u is compatible with every synopsis obtained from σ by replacing q_ℓ with some other state; similarly with p_ℓ and q_ℓ swapped.

The states of $\mathcal{B}$ include all synopses for $\mathcal{A}$ and two sink states: all-accepting $\top$ and all-rejecting $\bot$. The simulation invariant is that after processing a proper prefix w of the encoding of the input tree, either $\mathcal{B}$ is in the state $\top$ and $r_0 \cdot \widehat{v}$ is non-rejective for some prefix v of w, or $\mathcal{B}$ is in a synopsis state σ and $\widehat{w}$ is compatible with σ and if the last symbol of w is an opening tag

then $p_\ell = q_\ell$.

Let r_0 be the initial state of $\mathcal{A}$. If r_0 is rejective, the initial state of $\mathcal{B}$ is (r_0, r_0, r_0); otherwise, it is $\top$. The invariant clearly holds before the first tag is processed. Let us see how to define transitions from a synopsis state σ of the form (6.1) to propagate the invariant.

Suppose that an opening tag a is read and let $s = p_\ell \cdot a = q_\ell \cdot a$. If s is not rejective, move to $\top$. If s is rejective and belongs to the SCC of q_ℓ, continue with (r_ℓ, p_ℓ, q_ℓ) replaced with (r_ℓ, s, s) in σ. If s is rejective but does not belong to the SCC of q_ℓ, continue with $\xrightarrow{a} (s, s, s)$ appended to σ. The invariant propagates.

Suppose a closing tag $\bar{a}$ is read. If p_ℓ is not internal, then $p_\ell = q_\ell = r_0$, which is only possible if $\sigma = (r_0, r_0, r_0)$. The automaton $\mathcal{B}$ then moves to $\bot$. Assume that the invariant holds before $\bar{a}$ is processed. Then, $r_0 \cdot \bar{w} = r_0$. Because r_0 is not internal, it follows that w is empty. Hence, $w\bar{a} = \bar{a}$, which is not a prefix of the encoding of any tree, and the state of $\mathcal{B}$ after processing $w\bar{a}$ does not matter. If p_ℓ is internal, we consider four cases depending on whether p_ℓ and q_ℓ are in the same SCC of $\mathcal{A}$, and whether the shape of the synopsis allows backtracking via a transition that originates outside of the SCC of q_ℓ.

Case A: p_ℓ and q_ℓ are in the same SCC X, and either $r_\ell \notin \{p_\ell, q_\ell\}$ or $a \neq a_\ell$ or $p_{\ell-1}$ is not internal; that is, we can only take (backward) transitions within X. Consider

$$P = \left\{ p \in X \;\middle|\; p \cdot a \in \{p_\ell, q_\ell\} \right\}.$$

Because X contains the internal state p_ℓ and the rejective state q_ℓ, all states in X are internal and rejective. The same holds for $P \subseteq X$. Pick any two $p, q \in P$. Because p_ℓ and q_ℓ meet inside X, so do p and q. It follows that p and q meet in q. Hence, (p, q) is a split state, and p and q are almost equivalent. In a minimal automaton there can be at most two different almost equivalent states, so $|P| \leq 2$. If $P = \emptyset$, then $\mathcal{B}$ moves to $\bot$. Otherwise, $P = \{p', q'\}$ for some p' and q', and $\mathcal{B}$ continues, replacing (r_ℓ, p_ℓ, q_ℓ) with (r_ℓ, p', q'). Suppose that the invariant holds before $\bar{a}$ is processed. If it holds by (6.2.2), then $r_0 \cdot \widehat{w\bar{a}} \in P = \{p', q'\}$ and $\widehat{w\bar{a}}$ is compatible with the synopsis obtained from σ by replacing (r_ℓ, p_ℓ, q_ℓ) with $(r_\ell, r_0 \cdot \widehat{w\bar{a}}, r_0 \cdot \widehat{w\bar{a}})$. Suppose that the invariant holds by (6.2.2). This implies that $r_\ell \in \{p_\ell, q_\ell\}$ and $a = a_\ell$, so it must be the case that p_ℓ is not internal. Then q_ℓ is equal to p_ℓ, so not internal either. By (6.2.2), $r_0 \cdot \widehat{w\bar{a}} \in \{p_{\ell-1}, q_{\ell-1}\}$, so it is non-internal too. Consequently, $\widehat{w\bar{a}}$ is the empty word, which is possible only if $w\bar{a}$ is the complete encoding of the input tree. But then the invariant is not required to hold. Finally, the invariant cannot hold by (6.2.2), because it would imply that $q_{\ell-1}$ and q_ℓ are in the same SCC, which is forbidden by the definition of synopsis.

Case B: p_ℓ and q_ℓ are in the same SCC X, and also $r_\ell \in \{p_\ell, q_\ell\}$, $a = a_\ell$, and $p_{\ell-1}$ is internal; that is, we can also take (backward) transitions that leave X. Note that this is possible only if $\ell > 0$. Consider again the set $P \subseteq X$ introduced above. If $P = \emptyset$, then $\mathcal{B}$ continues, removing the suffix $\xrightarrow{a_\ell} (r_\ell, p_\ell, q_\ell)$ from the synopsis. In this case, only the condition (6.2.2) of the invariant might hold before processing $\bar{a}$, so $\widehat{w\bar{a}}$ is compatible with the modified synopsis, and the invariant propagates. Assume that P is nonempty. Let $p' \in \{p_{\ell-1}, q_{\ell-1}\}$ and $q' \in P$. We know that $p' \cdot a$ and $q' \cdot a$ belong to $\{p_\ell, q_\ell\}$, and that p_ℓ and q_ℓ meet in X, so we also have that p' and q' meet in X. Because $q' \in P \subseteq X$, it follows that p' and q' meet in q'. As $p_{\ell-1}$ is assumed to be internal, so is $q_{\ell-1}$, and consequently also p'. The state q' is rejective because all states in P are. It follows that (p', q') is a split state, so p' and q' are almost equivalent. Because $p' \in \{p_{\ell-1}, q_{\ell-1}\} \subseteq X^{\mathrm{c}}$ and $q' \in P \subseteq X$, we conclude that $p' \neq q'$. Using again the fact that there are at most two different almost equivalent states in every minimal automaton, we get that $p' = p_{\ell-1} = q_{\ell-1}$ and $\{q'\} = P$. The automaton $\mathcal{B}$ continues, replacing (r_ℓ, p_ℓ, q_ℓ) with (r_ℓ, p', q') in

the synopsis σ. If the invariant holds before processing $\bar{a}$, then either (6.2.2) or (6.2.2) holds. If (6.2.2) holds, then $r_0 \cdot \widehat{w\bar{a}} = q'$, and the invariant propagates like before. If (6.2.2) holds, then $r_0 \cdot \widehat{w\bar{a}} = p' = p_{\ell-1} = q_{\ell-1}$, and after processing $\bar{a}$, (6.2.2) will hold.

Case C: q_ℓ is in SCC X but $p_\ell \notin X$, and either $r_\ell \notin \{p_\ell, q_\ell\}$ or $a \neq a_\ell$. We then have $p_\ell = p_{\ell-1} = q_{\ell-1}$. Suppose $p \cdot a = p_\ell$ for some internal p and $q \cdot a = q_\ell$ for some $q \in X$. Then it easily follows that p meets with q in q, and so p and q are almost equivalent. Consequently, $p \cdot a = p_\ell$ and $q \cdot a = q_\ell$ are equal, which is impossible because $p_\ell \notin X$. Thus, p and q cannot both exist.

If p does not exist, $\mathcal{B}$ moves to the state it would take from the synopsis σ' obtained from the current one by replacing (r_ℓ, p_ℓ, q_ℓ) with (r_ℓ, q_ℓ, q_ℓ) in σ. Note that σ' falls into Case A. Suppose that the invariant holds before processing $\bar{a}$. If it is by (6.2.2), then $r_0 \cdot \widehat{w} = q_\ell$, so $\widehat{w}$ will also be compatible with σ' and the invariant will propagate as shown in Case A. The invariant cannot hold by (6.2.2), because this would imply that $r_\ell \in \{p_\ell, q_\ell\}$ and $a = a_\ell$, and we have assumed the contrary. Suppose that the invariant holds by (6.2.2). Then $(r_0 \cdot \widehat{w\bar{a}}) \cdot a = r_0 \cdot \widehat{w} = p_\ell$. But, as we have shown, there are no internal states p such that $p \cdot a = p_\ell$. Hence, $r_0 \cdot \widehat{w\bar{a}}$ is a noninternal state. This is possible only if $\widehat{w\bar{a}}$ is empty. Then, $w\bar{a}$ is the whole encoding of the input tree, and the invariant is not required to hold any more.

If q does not exist, the state is chosen similarly, but this time we obtain σ' by removing the suffix $\xrightarrow{a_\ell}(r_\ell, p_\ell, q_\ell)$ from σ. Note that σ' falls into Case A or Case B: $p_{\ell-1}$ is internal because it is equal to p_ℓ, and $p_{\ell-1}$ and $q_{\ell-1}$ are in the same SCC because they are equal. If the invariant holds before processing $\bar{a}$, then it must be by (6.2.2). Then, $\widehat{w}$ will also be compatible with σ', and the invariant will propagate as shown in Cases A and B.

Case D: q_ℓ is in SCC X but $p_\ell \notin X$, and both $r_\ell \in \{p_\ell, q_\ell\}$ and $a = a_\ell$. It then follows that $p_\ell = p_{\ell-1} = q_{\ell-1}$ and $r_\ell = q_\ell$. Consequently, $p_\ell \cdot a = q_\ell$ and, because p_ℓ and q_ℓ are almost equivalent, $q_\ell \cdot a = q_\ell$. Suppose that $p \cdot a = p_\ell$ for some internal state p. Then, we have $p \cdot aa = q_\ell \cdot aa = q_\ell$; that is, p meets with q_ℓ in q_ℓ. Since q_ℓ is rejective, it follows that p and q_ℓ are almost equivalent. But that means that $p_\ell = p \cdot a = q_\ell \cdot a = q_\ell$, which is impossible because $p_\ell \notin X$. Hence, no such p exists. Suppose that $q \cdot a = q_\ell$ for some $q \in X \setminus \{q_\ell\}$. Then $q \cdot a = q_\ell \cdot a = q_\ell$ and it follows that q is almost equivalent to q_ℓ. But this is impossible because together with $p_\ell \notin X$ this would give three different almost equivalent states. Hence, such q also does not exist. We let $\mathcal{B}$ continue with the same synopsis. Suppose that the invariant holds before processing $\bar{a}$. If it is by (6.2.2), then $r_0 \cdot \widehat{w\bar{a}} = q_\ell$, because it is the only state in X from which the transition over a leads to $\{p_\ell, q_\ell\}$, and the invariant propagates. If the invariant holds by (6.2.2), then $r_0 \cdot \widehat{w\bar{a}} \in \{p_{\ell-1}, q_{\ell-1}\}$, but $p_{\ell-1} = q_{\ell-1} = p_\ell$, so for $\widehat{w\bar{a}}$ and σ we will have (6.2.2).

Finally, if the invariant holds by (6.2.2), it follows that $w\bar{a}$ is the whole encoding of the input tree, like in the first subcase of Case C, and the invariant is not required to hold any more.

This completes the construction of $\mathcal{B}$ and the proof that every run of $\mathcal{B}$ over the encoding of a tree T satisfies the invariant. Directly from the invariant it follows that after reading a prefix wa of $\langle T \rangle$ for a opening tag a, we have $p_\ell = q_\ell = r_0 \cdot \widehat{wa}$. To recognise EL it suffices to enrich the synopsis states of $\mathcal{B}$ with the information about the most recently read tag, and move directly to $\top$ whenever a closing tag $\bar{a}$ is read in a state storing the opening tag a and a synopsis with $p_\ell = q_\ell$ accepting in $\mathcal{A}$. The resulting automaton $\mathcal{B}'$ enters $\top$ in the situation described above or if it encounters a prefix v of the encoding such that $r_0 \cdot v$ is not rejective. In the first case, the automaton $\mathcal{B}'$ has detected a leaf such that the branch leading to it is labelled by a word from $\mathcal{L}$. In the second case, $\mathcal{B}'$ has detected a node such that each branch containing this node is labelled by a word from $\mathcal{L}$. Correctness of $\mathcal{B}'$ follows. □

The other direction is very similar to the proof of Lemma 6.12.

(a) Tree S. (b) Tree S'.

Figure 6.4: Fooling trees in Lemma 6.17.

Lemma 6.17.
For a regular language $\mathcal{L}$, if $E\mathcal{L}$ is registerless then $\mathcal{L}$ is E-flat.

Proof. Suppose that the minimal automaton $\mathcal{A}$ of $\mathcal{L}$ is not E-flat. Let i be the initial state of $\mathcal{A}$. Then, there exist words $s, t, u \in A^+$, $x \in A^*$ and states p, q such that $i \cdot s = p$, $p \cdot u = q \cdot u = q$, $q \cdot x$ is rejecting, and $p \cdot t$ is accepting iff $q \cdot t$ is rejecting. It follows that for each $k > 0$, $su^k x \in \mathcal{L}^c$, and $st \in \mathcal{L}$ iff $su^k t \in \mathcal{L}^c$.

Now the pumping is exactly the same as in the proof of Lemma 6.12, with the trees of Fig. 6.4 that have two more leaves labelled with x.

□

6.3 Stackless model

Under the markup encoding, finite automata are unable to check even the simplest properties of the input document: for instance, determining if one marked node is a child, descendant, or sibling of another marked node requires a stack—or at least a counter, used to compare depths of nodes. Realising multiple such tasks simultaneously seems to lead to multi-counter automata, which are notoriously hard to analyse. We take a different path: we allow only one counter, used exclusively to maintain the current depth in the tree, but additionally equip the automaton with a bounded number of registers, which can be used to store depths of critical nodes, and compare them later with the current depth. To keep our automata executable efficiently, we assume that they are deterministic. Thus we arrive at *deterministic input-driven 1-counter automata with registers*. 'Input-driven' is the standard terminology for counters or stacks that evolve independently of the state [20, 36]. Here it means that the counter increases by one with each opening tag read, and decreases by one with each closing tag read; such automata (without registers) are also called *visibly counter automata* [9]. Importantly, the only tests allowed on the values stored in registers are order comparisons with the current depth.

6.3.1 Depth-register automata

We can formally defined the announced model of computation.

Definition 6.18.

A *depth-register automaton* $\mathcal{A}$ is a tuple

$$(A, Q, q_{init}, F, \Xi, \delta),$$

where

- A is a finite alphabet,
- Q is a finite set of states,
- $q_{init} \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting (final) states,
- Ξ is a finite set of registers,
- $\delta : Q \times (A \cup \bar{A}) \times 2^{\Xi} \times 2^{\Xi} \to 2^{\Xi} \times Q$ is the transition function.

A *configuration* of $\mathcal{A}$ is a tuple $(q, d, \eta) \in Q \times \mathbb{Z} \times \mathbb{Z}^{\Xi}$, whose components specify the state, the current depth, and the values stored in the registers, respectively. We call a configuration (q, d, η) *accepting* if $q \in F$. The *initial configuration* is $c_{init} = (q_{init}, 0, \eta_{init})$ where $\eta_{init}(\xi) = 0$ for all $\xi \in \Xi$.

The *run of $\mathcal{A}$ over a word $a_1 a_2 \dots a_n \in (A \cup \bar{A})^*$ from a configuration* (q_0, d_0, η_0) is the unique sequence of configurations

$$(q_0, d_0, \eta_0)(q_1, d_1, \eta_1)\dots(q_n, d_n, \eta_n) \in \left(Q \times \mathbb{Z} \times \mathbb{Z}^{\Xi}\right)^*$$

such that for each $i \in \{1, 2, \dots, n\}$, there exists $Y_i \subseteq \Xi$ such that

- $d_i = \begin{cases} d_{i-1} + 1 & \text{if } a_i \in A, \\ d_{i-1} - 1 & \text{if } a_i \in \bar{A}; \end{cases}$
- $\delta(q_{i-1}, a_i, X_i^{\leq}, X_i^{\geq}) = \left(Y_i, q_i\right)$ where

$$X_i^{\leq} = \{\xi \in \Xi \mid \eta_{i-1}(\xi) \leq d_i\},$$
$$X_i^{\geq} = \{\xi \in \Xi \mid \eta_{i-1}(\xi) \geq d_i\};$$

- for each $\xi \in \Xi$,

$$\eta_i(\xi) = \begin{cases} d_i & \text{if } \xi \in Y_i, \\ \eta_{i-1}(\xi) & \text{if } \xi \notin Y_i. \end{cases}$$

We write $c \cdot w$ for the last configuration of the run on w from c. If $c \cdot w = c'$, we also write $c \xrightarrow{w} c'$. By the *run of $\mathcal{A}$ on w* we understand the run on w from c_{init}. We say that w is *accepted* by $\mathcal{A}$ if $c_{init} \cdot w$ is accepting. The *language recognised by $\mathcal{A}$* is the set of words accepted by $\mathcal{A}$.

Depth-register automata without registers (that is, with $\Xi = \emptyset$) are a notational variant of deterministic finite automata over the alphabet $A \cup \bar{A}$.

The languages that are weakly validatable with a dept-register automata are said to be *stackless*. A query is stackless if it can be implemented by a depth-register automaton.

To conclude the discussion of the automata model, let us point out that the kind of tests allowed on registers is a natural parameter of the definition. For instance, one could allow testing if the current depth differs from the content of a given register by a specified constant; this kind of test can be simulated in our model at the cost of using additional registers. An interesting proper extension is to allow semilinear conditions, like testing equality modulo a specified constant. Finally, forsaking any hope of decidability of emptiness (which might be tolerable), one could go up to full arithmetics. Owing to their determinism, depth-register automata in all these variants would be efficiently executable in practice, using only a constant

number of variables (possibly just CPU registers).

We give several example of the power of such automata with registers.

First note that stackless languages need not to be regular.

Example 6.19.
The set of trees over the alphabet $\{a, b\}$ in which all a-labelled nodes are at the same depth, can be recognised by a depth-register automaton. The first time the automaton sees a, it stores the current depth in its only register. Then, every time it sees a it checks if the current depth is equal to the stored value, and if it is not, it moves to a rejecting sink state.

How far do stackless tree languages go beyond registerless? Let us see how depth-register automata can deal with sequences of siblings and the descendent relation.

Example 6.20.
Consider a regular language $\mathcal{L} \subseteq A^*$ and the set $H\mathcal{L}$ of trees over A such that the sequence of labels read from the children of the root forms a word in $\mathcal{L}$. Depending on $\mathcal{L}$, the tree language $H\mathcal{L}$ may be registerless or not. For instance, for $\mathcal{L} = A^*aA^*$, $H\mathcal{L}$ is not registerless, because a finite automaton cannot determine whether the current tag with label a belongs to a child of the root. This follows form our general result Theorem 6.15 applied to the set of trees that contain a branch labelled by a word from AaA^*.

In contrast, $H\mathcal{L}$ is stackless for all regular $\mathcal{L}$. Indeed, after reading the first tag (which must be an opening tag in a valid encoding), the automaton stores the current depth (which is 1) in its only register, and then simulates the finite automaton recognising $\mathcal{L}$ over all closing tags for which the current depth is equal to the value stored in the register. This is correct, because in each valid encoding all closing tags with current depth 1 belong to the children of the root.

Example 6.21.
Consider the set of trees over the alphabet $\{a, b, c\}$ where the first a-labelled node (in the document order) has a b-labelled descendent. To recognise this language, the automaton should read the input word until it sees a, load the current depth to its only register, and accept iff it sees the letter b before the current depth drops strictly below the stored value (this will indicate, that the corresponding closing tag has been read). Now, consider the set of trees over $\{a, b, c\}$ where *some* a-labelled node has a b-labelled descendant. It suffices to test this property for minimal a-labelled nodes (that is, those without a-labelled ancestors): if a node has a b-labelled descendent, so do all its ancestors. Hence, to recognise the described language it suffices to run the automaton described above in a loop, returning to the initial state whenever the current depth drops strictly below the stored value, until it accepts.

The main weakness of depth-register automata when applied to processing trees is their limited ability to handle the child relation, as revealed by the following example.

Example 6.22.
Consider the language of trees over the alphabet $\{a, b, c\}$ where some a-labelled node has a b-labelled child. It might appear that this language is stackless because it is easy to identify an a-labelled node and a single register is sufficient to identify the tags of its children in the encoding. Indeed, this idea can be used to recognise the language of trees where some *minimal* a-labelled node has a b-labelled child, just like we did for b-labelled descendants in Example 6.21. Without the minimality assumption, however, the subautomaton searching for b-labelled children needs to be relaunched whenever the opening tag a is read, which may well happen before the previous instance of the subautomaton terminates. Each launch requires a new register to store the return point. Because the input tree may contain arbitrarily long chains of a-labelled nodes, this does not seem feasible with any fixed number of registers. That it is indeed infeasible follows from the general characterisation result we establish in the next subsection.

The method from Example 6.21 can be extended to test the existence of multiple nodes with specified labels and descendent relationships between them. By a *descendent pattern* we shall understand a finite tree over A. A tree T *contains* a descendent pattern π if there exists a matching function h that maps nodes of π to nodes of T such that for all nodes u, v of π:

- the label of u coincides with the label of $h(u)$;
- if v is a child of u, then $h(v)$ is a descendent of $h(u)$.

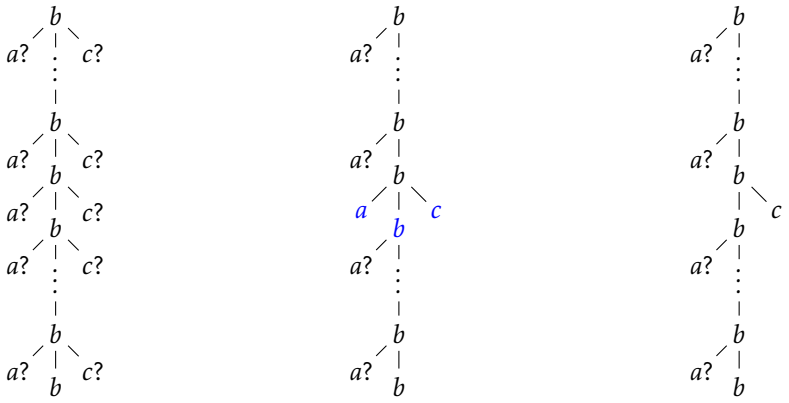
Fact 6.23.
For each descendent pattern π, the set of trees containing π is stackless.

Proof. By a slight abuse of the definition of depth-register automata, we shall allow automata that can stop; that is, in some configurations there may be no transition to take. We prove by induction on the height of π that there is an automaton $\mathcal{A}_\pi$ that recognises trees that contain π and stops upon reading the closing tag corresponding to the first opening tag of its input.

If π consists of a single node, the automaton loads into its only register the current depth before reading any tags, scans the input until the current depth again becomes equal to the stored value. Then it moves to a state without outgoing transitions that is accepting or not, depending on whether the automaton has detected a tag with the label from the root of π or not.

Suppose that the root of π has some children. By the inductive hypothesis, there is an automaton $\mathcal{A}_{\pi'}$ for each descendent pattern π' corresponding to an immediate subtree of π. Let $\mathcal{A}$ be the synchronous product of all these automata, recognising the intersection of the languages recognised by its components. Like in Example 6.21, we can assume that the root of π is matched to a minimal element with the desired label. The automaton $\mathcal{A}_\pi$ loads the current depth before reading any tags into its first register, and then processes the input looking for the first opening tag with the same label as the root of π. If $\mathcal{A}_\pi$ does not see one before the current depth is again equal to the stored value, it rejects. If it does find one, it calls the automaton $\mathcal{A}$ using a set of registers excluding the first one and waits until $\mathcal{A}$ stops. If $\mathcal{A}$ accepts, $\mathcal{A}_\pi$ waits until the current depth becomes equal to the value stored in the first register, and accepts. If $\mathcal{A}$ rejects, $\mathcal{A}_\pi$ moves on to the next opening tag with the same label as the root of π. □

Finally, let us point out that the ability to deal with sequences of siblings, demonstrated in Example 6.20, is limited to nodes that are close to the root. The following example shows why.



(a) Trees from K_n. (b) Match. (c) No match.

Figure 6.5: Fooling trees for several siblings.

Example 6.24.
Even a finite automaton can check if the streamed tree contains two consecutive siblings with labels a and b: it suffices to check if the read encoding contains the closing tag $\bar{a}$ followed immediately by the opening tag b. Consider, however, the set of trees that contain three consecutive siblings with labels a, b, c. This language is not stackless. Indeed assume that this language is recognised by a depth-register automaton $\mathcal{B}$ with m states and l registers. For a size n, we define K_n to be the set of trees that have the main branch labelled by the word b^n, and additionally each b-labelled node may have a c-labelled child to the right of the main branch, and each b-labelled node may have an a-labelled child to the left of the main branch except for the last one. Such trees are pictured in Fig. 6.5a. For a tree T like this, let w_T be the prefix of $\langle T \rangle$ ending at the opening tag of the deepest b-labelled node. Let c_{init} be the initial configuration of $\mathcal{B}$.

A configuration $c_{init} \cdot w_T$ is composed by a state, the depth n, and a value in $\{0, \dots, n\}$ for every register. That is, $m \cdot (n+1)^l$ configurations are possible. But there are 2^{n-1} ways to choose which b-labelled non-leaf nodes have an a-labelled child, so $\left|\left\{w_T \;\middle|\; T \in K_n\right\}\right| = 2^{n-1}$. Consequently, for sufficiently large n, there exist two different words u and v in $\left\{w_T \;\middle|\; T \in K_n\right\}$, such that $c_{init} \cdot u = c_{init} \cdot v$. Because $u \neq v$, there exists $i \in \{1, 2, 3, \dots, n\}$ such that for all $S, T \in K_n$, if $u = w_S$ and $v = w_T$, then the ith b-labelled node has an a-labelled child in S iff it does not have one in T. Let us choose S and T such that in both of them, the ith b-labelled node has a c-labelled child and there are no other c-labelled nodes, as shown in Figs. 6.5b and 6.5c. Clearly, the tree in Fig. 6.5b contains three sibling labelled by a, b and c. It is not difficult to verify that the one in Fig. 6.5c does not. However, from the definition of S and T it follows that $\langle S \rangle = uw'$ and $\langle T \rangle = vw'$ for some w', and because $c_{init} \cdot uw' = c_{init} \cdot vw'$, we conclude that S and T are indistinguishable to $\mathcal{B}$.

Thus, depth-register automata are able to express involved global properties of trees (Fact 6.23), far out of reach of finite automata, yet they cannot handle many properties that appear local but lose their locality when seen as properties of the encodings (Examples 6.22 and 6.24). Characterising stackless tree languages seems to be challenging, but in the following we solve the special

case of tree languages defined in terms of properties of branches.

6.3.2 Hierarchical almost-reversibility

We have already developed intuitions on evaluating RPQs over markup encodings using finite automata. Can we do more using the depth information and the (limited) ability to process it offered by the registers? Using one register and an additional component in the state, we can store the configuration of the simulated automaton in one node on the path from the root to the current node: we store the depth of this node in the register and the state of the simulated automaton in the additional component of the state of the simulating automaton. When the simulation climbs up to this depth again, we know to which state the simulated automaton should be reverted, regardless of the reversibility assumptions.

Using this feature we can simulate automata whose strongly connected components (SCCs) are singletons (Fig. 6.2b). Recall that an SCC is a maximal subset X of the state-space such that every state in X is reachable from every other state in X. If each SCC is a singleton, then a run may loop in some states it visits, but it never revisits a state it has left. Hence, in each run there is a bounded number of state changes. The simulating automaton can then represent the whole run of the simulated automaton over the path from the root to the current node by means of the list of state changes and depths at which these changes occurred. Automata with only singleton SCCs capture exactly the class of $\mathcal{R}$-*trivial* languages. As we shall see, the potential of register automata is exhausted by the combination of the above simulation method with the full power of finite automata to simulate a run inside a single SCC. The class of automata that can be simulated this way is captured by the following definition.

Definition 6.25.
A deterministic automaton is *hierarchically almost-reversible*, abbreviated as HAR, if every two states from the same SCC that meet inside this SCC are almost equivalent. A regular language is HAR if its minimal automaton is HAR.

By design, HAR languages include all almost-reversible languages (Fig. 6.2a), and all $\mathcal{R}$-trivial languages (Fig. 6.2b), but also the language in Fig. 6.2c which is neither almost-reversible nor $\mathcal{R}$-trivial. The language in Fig. 6.2d, is not HAR.

As Definition 6.25 is invariant under the complementation of the automaton, we obtain the following.

Lemma 6.26.
The complement of a HAR language is HAR.

We can state our main characterisation theorem.

Theorem 6.27.
Let $\mathcal{L}$ be a regular language. The following are equivalent:

1. $Q\mathcal{L}$ is a register query,
2. $E\mathcal{L}$ is a registerless language,
3. $A\mathcal{L}$ is a registerless language,

4. $\mathcal{L}$ is hierarchically almost-reversible.

Proof. The rest of the section is devoted to prove (2) implies (4), and (4) implies (1). If we assume that they are correct, we can prove the remaining implications. (1) implies (2) because an automaton $\mathcal{A}$ realising QL can be easily turned into an automaton $\mathcal{A}'$ recognising EL. $\mathcal{A}'$ behaves like $\mathcal{A}$, but it additionally remembers the previously read symbol; if the previous symbol was an opening tag, the state is accepting in $\mathcal{A}$, and the current letter is a closing tag, then $\mathcal{A}'$ moves to an all-accepting sink state. It follows that (2) and (3) are equivalent. Indeed, we use the equality that $(A\mathcal{L})^{\mathrm{c}} = E(\mathcal{L}^{\mathrm{c}})$ and the fact that both stackless and HAR languages are closed under complement. □

We first give a stackless algorithm for HAR languages ((4) implies (1) in Theorem 6.27).

Lemma 6.28.
If $\mathcal{L}$ is a HAR language, then $Q\mathcal{L}$ is a stackless query.

Proof. Let $\mathcal{L}$ be a HAR language and $\mathcal{A}$ its minimal automaton. Like before, we construct a depth-register automaton $\mathcal{B}$ that evaluates $Q\mathcal{L}$ by maintaining a simulation of the run of $\mathcal{A}$ on the word $\widehat{w}$ labelling the path π from the root to the current node. It applies the method used for $\mathcal{R}$-trivial languages to keep track of the changes of SCCs of $\mathcal{A}$ during the simulated run, and an adaptation of the method for almost-reversible languages to deal with the segments of the simulated run within a single SCC. After processing a prefix w of the encoding of the input tree, for each SCC X of $\mathcal{A}$ visited during the run on $\widehat{w}$, except the current one, the automaton $\mathcal{B}$ stores

- the depth of the deepest node on the path π whose label was read in a state from X during the run on $\widehat{w}$; and
- some state from X that meets in X with the last state from X visited by $\mathcal{A}$ in the run on $\widehat{w}$.

Additionally, if q is the current state of $\mathcal{A}$ after processing $\widehat{w}$ and Y is the SCC of $\mathcal{A}$ that contains q, the automaton $\mathcal{B}$ stores some state $p \in Y$ that meets with q in Y, and $p = q$ after reading each opening tag. Initially, p is the initial state i of $\mathcal{A}$, and nothing else is stored.

Suppose that $\mathcal{B}$ reads an opening tag a and the current depth is d. Because $\mathcal{A}$ is HAR, the states p and q mentioned above are almost equivalent. As $\mathcal{A}$ is minimal, it follows from Lemma 6.7 that $p \cdot a = q \cdot a$. Consequently, $p \cdot a$ is the next state of $\mathcal{A}$. If $p \cdot a \in Y$, we just replace p with $p \cdot a$ and proceed to the next tag. If $p \cdot a$ belongs to some SCC $Z \neq Y$, we also add Y to the list of remembered SCCs, with depth d (loaded to some unused register) and state p, and continue with Z as the current SCC.

Suppose now that $\mathcal{B}$ reads a closing tag $\bar{a}$ and the current depth d is greater than or equal to the maximal recorded depth d'. This indicates that the previous state of $\mathcal{A}$ also belongs to Y. We should now revert $\mathcal{A}$ to some state $q' \in Y$ such that $q' \cdot a = q$, but we do not know which one. Even worse, we do not have access to q, but only to some state $p \in Y$ that meets with q in Y. Nevertheless, we can maintain the invariant by picking any state $p' \in Y$ such that $p' \cdot a \in Y$ is almost equivalent to p. Note first that such states p' exist because q' is one of them: $q' \cdot a = q$ and from the previous case we know that q and p are almost equivalent. To keep $\mathcal{B}$ deterministic we pick the minimal such p' according to some arbitrarily fixed order on the states of $\mathcal{A}$. To prove that every p' is suitable it suffices to show that p' meets with q' in Y. We know that $p \cdot u = q \cdot u \in Y$ for some word u. Because $p' \cdot a$ is almost equivalent to p and $\mathcal{A}$ is

minimal, we get $p' \cdot a \cdot u = p \cdot u = q \cdot u = q' \cdot a \cdot u$, and we are done. Hence, $\mathcal{B}$ can replace p with p' and proceed to the next tag.

Finally, suppose $\mathcal{B}$ reads a closing tag $\bar{a}$ and the current depth is strictly smaller than the greatest recorded depth d'. This indicates that the previous state of $\mathcal{A}$ belongs to the SCC $X \neq Y$, associated with depth d'. The automaton $\mathcal{A}$ should be reverted to the last state q' from X visited during the run. The simulation does not have access to q', but it has the state p' recorded for X, and we know that p' meets with q' in X. This is sufficient to maintain the invariant: the automaton $\mathcal{B}$ simply replaces p with p', removes X from the list of remembered SCCs marking the register storing the associated depth d' as unused, and proceeds to the next tag with X as the current SCC. □

We now prove the inexpressibility direction of Theorem 6.27 ((2) implies (4)). We use pumping as in Lemma 6.12 and Lemma 6.17, but requires considerably more effort because this time we need to fool a depth-register automaton. Before we dive into it, we prepare some simple tools helping to analyse runs of such automata.

For configurations $c = (q, d, \eta)$ and $c' = (q', d', \eta')$ of a depth-register automaton $\mathcal{B}$ we write $c \sim c'$ if $q = q'$.

For $-\infty \leq i \leq j \leq \infty$, we write $c \approx_{i,j} c'$ if $c \sim c'$ and for each register ξ one of the following conditions holds:

- $\eta'(\xi) - d' = \eta(\xi) - d$;
- $\eta(\xi) - d < i$ and $\eta'(\xi) - d' < i$ and $\eta(\xi) = \eta'(\xi)$;
- $\eta(\xi) - d > j$ and $\eta'(\xi) - d' > j$.

We let $\|\varepsilon\| = 0$ and inductively $\|wa\| = \|w\| + 1$ and $\|w\bar{a}\| = \|w\| - 1$ for all $a \in A$ and $w \in (A \cup \bar{A})^*$. For nonempty w we also define

$$\lfloor w \rfloor = \min_{\varepsilon \neq u \leq w} \|u\|, \qquad \lceil w \rceil = \max_{\varepsilon \neq u \leq w} \|u\|,$$

where $u \leq w$ means that u is a prefix of w. Note that for all w,

$$\lfloor w \rfloor \leq \|w\| \leq \lceil w \rceil.$$

Lemma 6.29.
Suppose that $c_1 \approx_{i,j} c_2$. For every word w such that $i \leq \lfloor w \rfloor \leq \lceil w \rceil \leq j$, it holds that $c_1 \cdot w \approx_{i-\|w\|, j-\|w\|} c_2 \cdot w$.

Proof. It suffices to show the lemma for the case when w is a single letter; the general claim follows by straightforward induction on the length of w. Suppose that $w = a \in A$. Then, $\lfloor w \rfloor = \lceil w \rceil = 1$. Because $c_1 \approx_{i,j} c_2$ and $i \leq 1 \leq j$, it follows the same transition over a will be taken from c_1 and c_2. After the transition is taken, the absolute thresholds between the three kinds of behaviour of registers listed in the definition of $\approx$ do not change, but because the current depth increases by one, the relative thresholds have to be adjusted. This gives precisely $c_1 \cdot a \approx_{i-1,j-1} c_2 \cdot a$. For $w = \bar{a}$ the argument is entirely analogous.

□

A word $x \in (A \cup \bar{A})^+$ is *descending* if $1 = \lfloor x \rfloor \leq \lceil x \rceil = \|x\|$ and it is *ascending* if $-1 = \lceil x \rceil \geq \lfloor x \rfloor = \|x\|$. Descending words generalise words from A^+, and ascending words generalise words from $\bar{A}^+$.

For $i, j \in \mathbb{Z}_\infty = \mathbb{Z} \cup \{-\infty, \infty\}$ we let

$$[i,j] = \{k \in \mathbb{Z}_\infty \mid i \leq k \leq j\}, \quad (i,j] = \{k \in \mathbb{Z}_\infty \mid i < k \leq j\},$$

and analogously for $[i,j)$ and (i,j).

Lemma 6.30.
Let $c_i = (q_i, d_i, \eta_i)$ with $i \in [1;4]$ be configurations of a depth-register automaton $\mathcal{B}$ and let $y, z \in (A \cup \bar{A})^+$ be descending words such that $c_1 \xrightarrow{y} c_2 \xrightarrow{z} c_3 \xrightarrow{y} c_4$. If $\mathsf{img}(\eta_1) \subseteq (-\infty; d_1]$ and $c_1 \sim c_3$, then $\mathsf{img}(\eta_4) \cap (d_1; d_2] = \emptyset$.

Proof. Because y and z are descending, from $\mathsf{img}(\eta_1) \subseteq (-\infty; d_1]$ it follows that $\mathsf{img}(\eta_3) \subseteq (-\infty; d_3]$. Combining this with $c_1 \sim c_3$, we conclude that from configurations c_1 and c_3 the same sequence of transitions will be taken while processing y. But this implies that if a depth $d \in (d_1; d_2]$ was stored in some register ξ while processing y from c_1, the corresponding depth $d' \in (d_3; d_4]$ will be stored in ξ while processing y from c_3. That is, each depth stored when the first copy of y was processed, is overwritten when the second copy of y is processed. Because $\mathsf{img}(\eta_1) \subseteq (-\infty; d_1]$, and both y and z are descending, there is no other way of putting a value from the segment $(d_1; d_2]$ into registers.

□

Lemma 6.31.
Let $\mathcal{B}$ be a depth-register automaton with k states and ℓ registers, and let $n \geq k \cdot (\ell + 1)$. For every configuration $c = (q, d, \eta)$ of the automaton $\mathcal{B}$ and every descending or ascending word $x \in (A \cup \bar{A})^+$, if

$$\mathsf{img}(\eta) \cap \left[d + \left\lfloor x^{3 \cdot n!} \right\rfloor; d + \left\lceil x^{3 \cdot n!} \right\rceil\right] = \emptyset,$$

then

1. $c \cdot x^{n!} \sim c \cdot x^{n!} \cdot x^{n!}$; and
2. $c \cdot x^{n!} \cdot x^{n!} \approx_{\lfloor x^{n!} \rfloor - \|x^{n!}\|, \lceil x^{n!} \rceil - \|x^{n!}\|} c \cdot x^{n!} \cdot x^{n!} \cdot x^{n!}$.

Proof. It is well known that for every deterministic finite automaton $\mathcal{A}$ over $A \cup \bar{A}$ with at most n states, $p \cdot w^{n!} = p \cdot w^{n!} \cdot w^{n!}$ for every state p and every word w. To see why this is the case, let us analyse the evolution of the state after processing successive copies of w. Already after processing at most n copies a state will repeat, and because $\mathcal{A}$ is deterministic, we will start looping around a cycle in $\mathcal{A}$. After processing all $n!$ copies we are still on the cycle, of course. After processing any number of copies that is divisible by the length of the cycle (measured in the number of w-steps, not single letters), we return to the same state. Because the length of the cycle is at most n, and $n!$ is divisible by every number between 1 and n, the claim follows.

The lemma is proved in a similar fashion. Suppose x is descending; the argument for ascending x is entirely analogous. Throughout the run on $x^{n!} \cdot x^{n!} \cdot x^{n!}$ from c, the current depth stays within $\left[d + \left\lfloor x^{3 \cdot n!} \right\rfloor; d + \left\lceil x^{3 \cdot n!} \right\rceil\right]$. Consequently, comparisons with values from $\mathsf{img}(\eta)$ give the same result at every step of this run. Moreover, because x is descending, depths stored

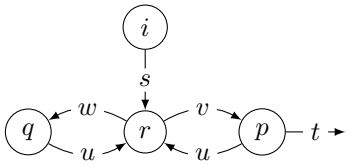


Figure 6.6: Non-HAR gadget Lemma 6.32.

when processing the ith copy of x are all strictly smaller than every depth that occurs when processing the jth copy of x for all $j > i$. Consequently, the behaviour of $\mathcal{B}$ when processing the $(i+1)$st copy of x is determined by the state and the set of registers storing values not greater than the current depth—after processing the ith copy of x. Because the set of registers can only grow as the successive copies of x are processed, after processing at most $k \cdot (\ell + 1)$ copies of x a state-set pair will repeat. Because the sets only grow, all state-pairs in between share the same set. It follows that when processing subsequent copies of x, this sequence of state-pairs will repeat in a cyclic fashion. Because the length of this sequence is at most $k \cdot (\ell + 1)$, it follows like before that the state-set pairs corresponding to $c \cdot x^{n!}$ and $c \cdot x^{n!} \cdot x^{n!}$ coincide. This implies item (1) of the lemma. In configuration $c \cdot x^{n!} \cdot x^{n!}$ some registers store the same value from

$$\left(-\infty; d + \left\lceil x^{n!} \right\rceil\right] \cup \left(d + \left\lceil x^{3 \cdot n!} \right\rceil; \infty\right)$$

that they stored in configuration $c \cdot x^{n!}$, and into the remaining registers some values from

$$\left(d + \left\lceil x^{n!} \right\rceil; d + \left\lceil x^{2 \cdot n!} \right\rceil\right]$$

were loaded when the second copy of $x^{n!}$ was being processed. Because the state-set pairs corresponding to $c \cdot x^{n!}$ and $c \cdot x^{n!} \cdot x^{n!}$ coincide, processing the third copy of $x^{n!}$ will load into the same registers the corresponding (that is, shifted by $\|x^{n!}\|$) values, and no other load operations will be performed. This implies item (2) of the lemma.

□

Lemma 6.32.
For each regular language $\mathcal{L}$, if $E\mathcal{L}$ is a stackless tree language, then $\mathcal{L}$ is HAR.

Proof. Again, we prove the contrapositive. Suppose $\mathcal{L} \subseteq A^*$ is not HAR. Then, its minimal automaton $\mathcal{A}$ admits states p, q, and r in the same SCC Y such that for some word u and some non-empty word t, we have $r = p \cdot u = q \cdot u$ and $p \cdot t$ is accepting and $q \cdot t$ is non-accepting (in particular, $p \neq q$). Then, there exist v and w such that $r \cdot v = p$ and $r \cdot w = q$. Finally, by minimality, all states are reachable from the initial state, so there exists a word s such that $i \cdot s = r$. Because Y contains two different states, it is a non-trivial SCC. Consequently, for each state $p' \in Y$ there exists a nonempty looping word; that is, a word $w' \neq \varepsilon$ such that $p' \cdot w' = p'$. By appending suitable looping words if necessary, we can assume that the words s, u, v, w are nonempty as well. Additionally, it will be convenient to assume that $|u| \geq |t|$; this can be ensured by appending $|t|$ copies of the appropriate looping word to u. The resulting fragment

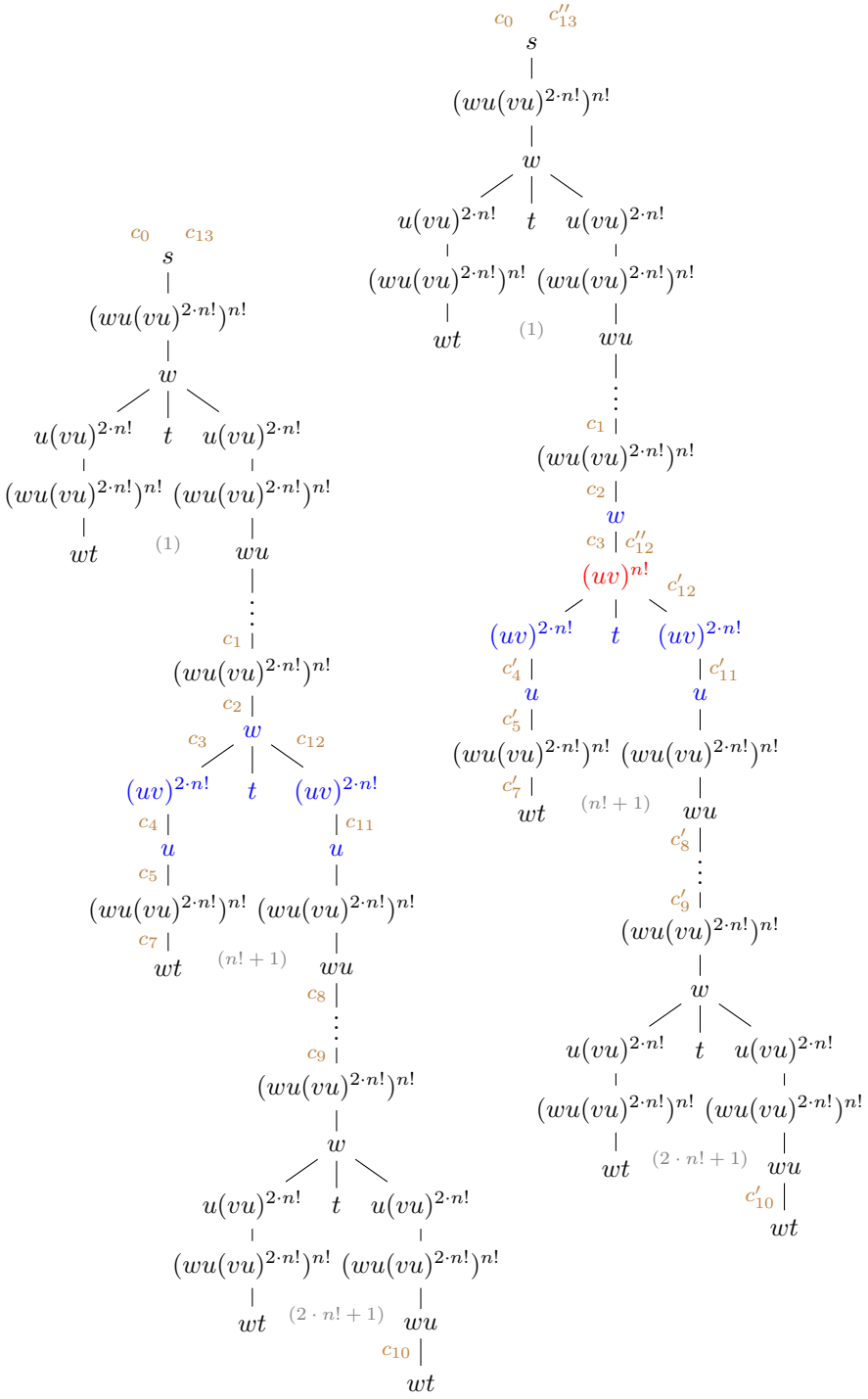


Figure 6.7: Fooling trees in Lemma 6.32.

of the automaton $\mathcal{A}$ is shown in the top left corner of Fig. 6.6. We have

$$s(wu + vu)^* vt \subseteq \mathcal{L}, \qquad s(wu + vu)^* wt \subseteq \mathcal{L}^{\mathrm{c}}.$$

Consider a depth-register automaton $\mathcal{B}$ over $A \cup \overline{A}$ with k states and ℓ registers. Let $n = k \cdot (\ell + 1)$. We shall construct a fooling pair of trees by unravelling the fooling gadget.

The trees are shown in Fig. 6.7. The original tree R, is build from: (i) a tree R_0 consisting of a single branch labelled by the word s, (ii) trees $R_1, \ldots, R_{2 \cdot n!+1}$ that are isomorphic copies of the same tree, and (iii) a tree $R_{2 \cdot n!+2}$ consisting of a single branch labelled by the word wt. Each branch of R is labelled by a word from $s(wu + vu)^* wt \subseteq compl\mathcal{L}$, which means that $R \notin EL$. The pumped tree R' is obtained by inserting an additional segment labelled by $(uv)^{n!}$ in $R_{n!+1}$, just before the branching; we will write $R'_{n!+1}$ for thus modified $R_{n!+1}$. The modification introduces a branch labelled by a word from $s(wu + vu)^* vt \subseteq \mathcal{L}$, which means that $R' \in E\mathcal{L}$. We will show that the automaton $\mathcal{B}$ cannot distinguish $\langle R \rangle$ from $\langle R' \rangle$, by analysing the respective runs in parallel. The crucial moments of the analysis will be configurations $c_i = (q_i, d_i, \eta_i)$, $c'_i = (q'_i, d'_i, \eta'_i)$, and $c''_i = (q''_i, d''_i, \eta''_i)$, depicted (with the exception of c_6) in brown in Fig. 6.7: configurations to the left of edges are visited when going down and those to the right when going up.

Let x be the prefix of $\langle R_1 \rangle$ ending at the opening tag of the rightmost leaf of R_1. Because $|t| \leq |u|$, the rightmost branch of R_1 is at least as long as both other branches, which implies that x is descending. Clearly, so is $y = wu(vu)^{2 \cdot n!} \in A^+$. Consider the following initial segments of the runs of $\mathcal{B}$ over $\langle R \rangle$ and $\langle R' \rangle$:

$$c_0 \xrightarrow{sx^{n!}} c_1 \xrightarrow{y^{n!}} c_2 \xrightarrow{w} c_3 \xrightarrow{(uv)^{2 \cdot n!}} c_4 \xrightarrow{u} c_5 \xrightarrow{y^{n!-1}} c_6 \xrightarrow{y} c_7,$$
$$c_0 \xrightarrow{sx^{n!}} c_1 \xrightarrow{y^{n!}} c_2 \xrightarrow{w} c_3 \xrightarrow{(uv)^{3 \cdot n!}} c'_4 \xrightarrow{u} c'_5 \xrightarrow{y^{n!-1}} c_6 \xrightarrow{y} c'_7.$$

Let $\delta = |(uv)^{n!}|$. As all words over the arrows are descending, we have

$$\mathsf{img}(\eta_i) \subseteq [-\infty; d_i], \quad \mathsf{img}(\eta'_j) \subseteq [-\infty; d'_j], \quad d'_j = d_j + \delta \tag{6.2}$$

for all $i \in [0;7]$ and $j \in [4;7]$. Condition (6.2) allows us to apply Lemma 6.31 to configuration c_3 and the descending word uv, and conclude that $c_4 \approx_{1-\delta,0} c'_4$. By (6.2), this can be strengthened to $c_4 \approx_{1-\delta,\infty} c'_4$. By Lemma 6.29, we get

$$c_7 \approx_{1-\|(uv)^{n!}u \cdot y^{n!}\|,\infty} c'_7. \tag{6.3}$$

Applying Lemma 6.31 to c_2 and y, we get $c_2 \sim c_6$. Hence, we can apply Lemma 6.30 to configurations c_2, c_5, c_6, c_7 and descending words y and $y^{n!-1}$. Combining the result with (6.3), we get

$$\mathsf{img}(\eta_7) \cap \left(d_2; d_5\right] = \emptyset, \qquad \mathsf{img}(\eta'_7) \cap \left(d_2; d'_5\right] = \emptyset. \tag{6.4}$$

Consequently, from (6.3) we can also conclude

$$c_7 \approx_{1-\|y^{n!+1}\|,\infty} c_7. \tag{6.5}$$

Let $y' = wu(vu)^{3\cdot n!}$ and take x_0 such that $y^{2\cdot n!+1} \cdot x_0 = x$. Consider

$$c_0 \xrightarrow{sx^{n!}} c_1 \xrightarrow{y^{n!}\cdot y\cdot y^{n!}} c_7 \xrightarrow{x_0} c_8 \xrightarrow{x^{n!-1}} c_9 \xrightarrow{x} c_{10},$$
$$c_0 \xrightarrow{sx^{n!}} c_1 \xrightarrow{y^{n!}\cdot y'\cdot y^{n!}} c'_7 \xrightarrow{x_0} c'_8 \xrightarrow{x^{n!-1}} c'_9 \xrightarrow{x} c'_{10}.$$

Note that condition (6.2) holds for all $i, j \in [8;10]$. From (6.5) via Lemma 6.29 we get

$$c_{10} \approx_{1-\|y^{n!+1}wux^{n!}\|,\infty} c'_{10}. \tag{6.6}$$

Applying Lemma 6.31 to configuration $c_0\cdot s$ and the descending word x, we get $c_1 \sim c_9$. Applying Lemma 6.30 to configurations c_1, c_8, c_9, c_{10} and the descending words x and $x^{n!}$, and combining the result with (6.6), we get

$$\mathrm{img}(\eta_{10}) \cap (d_1; d_8] = \emptyset, \qquad \mathrm{img}(\eta'_{10}) \cap (d_1; d'_8] = \emptyset. \tag{6.7}$$

Hence, we can strengthen (6.6) to

$$c_{10} \approx_{1-\|x^{n!+1}\|,\infty} c'_{10}. \tag{6.8}$$

Let $\bar{x} = \bar{u}\bar{w}\bar{y}^{2\cdot n!+1}$; that is, $x\bar{x} = \langle R_1 \rangle$. Consider

$$c_{10} \xrightarrow{wt\bar{t}\bar{w}\cdot\bar{x}^{n!}\cdot\bar{u}\bar{w}\bar{y}^{n!}\cdot\bar{u}} c_{11} \xrightarrow{(\bar{v}\bar{u})^{2\cdot n!}} c_{12} \xrightarrow{\bar{w}\cdot\bar{y}^{n!}\cdot\bar{x}^{n!}\cdot\bar{s}} c_{13},$$
$$c'_{10} \xrightarrow{wt\bar{t}\bar{w}\cdot\bar{x}^{n!}\cdot\bar{u}\bar{w}\bar{y}^{n!}\cdot\bar{u}} c'_{11} \xrightarrow{(\bar{v}\bar{u})^{2\cdot n!}} c'_{12} \xrightarrow{(\bar{v}\bar{u})^{n!}} c''_{12} \xrightarrow{\bar{w}\cdot\bar{y}^{n!}\cdot\bar{x}^{n!}\cdot\bar{s}} c''_{13}.$$

We have $d'_i = d_i + \delta$ for $i \in [10;12]$ and $d''_i = d_i$ for $i \in [12;13]$. By Lemma 6.29, we have $c_{12} \approx_{1-\|w\|,\infty} c'_{12}$. As from (6.7) it follows that $\mathrm{img}(\eta_{12}) \cap (d_1; d_{12}) = \mathrm{img}(\eta'_{12}) \cap (d_1; d'_{12}) = \emptyset$, we also have

$$c_{12} \approx_{0,\infty} c'_{12}. \tag{6.9}$$

Applying Lemma 6.31 to configuration c'_{11} and the ascending word $\bar{v}\bar{u}$, we get $c'_{12} \approx_{0,\delta-1} c''_{12}$. In combination with (6.9) this implies $c_{12} \approx_{0,\delta-1} c''_{12}$. Because $d_{12} = d''_{12}$, it follows that $c_{12} \approx_{-\infty,\delta-1} c''_{12}$. By Lemma 6.29, this implies $c_{13} \sim c''_{13}$. □

6.4 Term encoding

An alternative way to serialize tree-structured data, used for instance in JSON, is the *term encoding*, in which the information about the label is included only in opening tags. Streaming processing under this encoding is harder, but analyzing it is easier. An effective characterization of regular tree languages that are registerless under the term encoding is given in [9].

A tree language $\mathcal{L}$ over A is *term-registerless* (resp. *term-stackless*) if there exists a finite automaton (resp. depth-register automaton) over $A \cup \{\lhd\}$ that accepts all words from $[\mathcal{L}]$ and rejects all words from $[\mathcal{L}^c]$. A unary query Q is *term-registerless* (resp. *term-stackless*) if there exists a finite automaton (resp. depth-register automaton) over $A \cup \{\lhd\}$ that pre-selects nodes in $Q(T)$ when running over $[T]$.

Our treatment can be easily adapted to the term encoding by adjusting the definition of when two states meet: we say that states p and q *blindly meet* in state r if there exist words $u_1, u_2 \in A^*$ such that $|u_1| = |u_2|$ and $p\cdot u_1 = q\cdot u_2 = r$. By replacing 'meet' with 'blindly meet' in Definitions 6.8, 6.13 and 6.25, we get the definitions of the syntactic classes of *blindly almost-reversible*, *blindly*

HAR, *blindly A-flat*, and *blindly E-flat* word languages. All the main theorems then hold for the term encoding with all syntactic classes of word languages replaced by their blind analogues.

Nevertheless, 'blind' classes are much more restricted than their originals: all $\mathcal{R}$-trivial languages are blindly HAR, but the possibilities of backtracking inside an SCC are very limited. For example, the minimal automaton shown in Fig. 6.1 is reversible, but not blindly-HAR; this means that the language $(b^*ab^*ab^*)^*$ this automaton recognizes is registerless under the markup encoding, but not even stackless under the term encoding. This is the cost of succinctness.

Theorem 6.33.
Let $\mathcal{L}$ be a regular language.

1. $E\mathcal{L}$ is a term-registerless tree language iff $\mathcal{L}$ is blindly E-flat.
2. $A\mathcal{L}$ is a term-registerless tree language iff $\mathcal{L}$ is blindly A-flat.
3. The following conditions are equivalent:
   (a) $Q\mathcal{L}$ is a term-registerless unary query;
   (b) $E\mathcal{L}$ and $A\mathcal{L}$ are term-registerless tree languages;
   (c) $\mathcal{L}$ is blindly E-flat and blindly A-flat;
   (d) $\mathcal{L}$ is blindly almost-reversible.

Proof. The argument is fully analogous to that in Theorem 6.10 and Theorem 6.15, with Lemma 6.16,Lemma 6.12, Lemma 6.17, and Lemma 6.11. replaced by their analogues for term-registerless, blindly E-flat, blindly A-flat, and blindly almost-reversible languages.

The analogue of Lemma 6.11 states that if $\mathcal{L}$ is a blindly almost-reversible language, then $Q\mathcal{L}$ is a term-registerless query. The proof is almost identical, except that when the closing tag $\triangleleft$ is read in state p, we pick any state p' such that $p' \cdot a$ is almost equivalent to p for some $a \in A$; because $\mathcal{L}$ is blindly almost-reversible, the original argument now shows also that the choice of a does not matter.

The analogue of Lemma 6.14 states that a regular language is blindly A-flat iff its complement is blindly E-flat, and that it is blindly almost-reversible iff it is both blindly A-flat and blindly E-flat; it is proved just like the original.

The analogue of Lemma 6.16 states that if $\mathcal{L}$ is blindly E-flat, then $E\mathcal{L}$ is term-registerless. The proof is an adaptation of the original one to the blind setting. The states of the simulating finite automaton, the simulation invariant, the transitions over opening tags, and the transformation into an automaton recognizing $E\mathcal{L}$ are entirely analogous, with 'meet' replaced everywhere with 'blindly meet'; in particular, we keep the labels $a_1, \ldots, a_\ell$ in the synopsis. However, the behaviour of the simulating automaton over the closing tag needs to be adjusted so that it does not rely on the label of the current node. We begin by dropping all references to the current label in the conditions defining Cases A–D, which gives

Case A': $p_\ell, q_\ell \in X$ but either $r_\ell \notin \{p_\ell, q_\ell\}$ or $p_{\ell-1}$ is not internal;

Case B': $p_\ell, q_\ell \in X$, $r_\ell \in \{p_\ell, q_\ell\}$, and $p_{\ell-1}$ is internal;

Case C': $q_\ell \in X$, $p_\ell \notin X$, and $r_\ell \notin \{p_\ell, q_\ell\}$;

Case D': $q_\ell \in X$, $p_\ell \notin X$, and $r_\ell \in \{p_\ell, q_\ell\}$.

In each of these cases the simulating automaton needs to consider all possible values of the current label. That is, in Cases A' and B', the set P is now defined as

$$P = \left\{ p \in X \;\middle|\; p \cdot a \in \{p_\ell, q_\ell\}, a \in A \right\},$$

and in Case C' we look at $p \cdot a_1 = p_\ell$ and $q \cdot a_2 = q_\ell$ for arbitrary $a_1, a_2 \in A$. Apart from these differences, the arguments in Cases A'–C' are analogous to the original ones. Let us have a closer look at Case D'. Like before we have $p_\ell = p_{\ell-1} = q_{\ell-1}$ and $r_\ell = q_\ell$. Consequently, $p_\ell \cdot a_\ell = q_\ell$ and, because p_ℓ and q_ℓ are almost equivalent, $q_\ell \cdot a_\ell = q_\ell$. Suppose that $p \cdot a = p_\ell$ for some internal state p and some $a \in A$. Then, we have $p \cdot aa_\ell = q_\ell \cdot a_\ell a_\ell = q_\ell$; that is, p blindly meets with q_ℓ in q_ℓ. Since q_ℓ is rejective, it follows from blind E-flatness that p and q_ℓ are almost equivalent. Consequently, $q_\ell \cdot a = p \cdot a = p_\ell$. Because we also have that $p_\ell \cdot a_\ell = q_\ell$, it follows that $p_\ell \in X$ which is a contradiction. Hence, such p cannot exist. One then argues, like in the markup case, that there is no $q \in X \setminus \{q_\ell\}$ for which there exists $a \in A$ such that $q \cdot a = q_\ell$, and that letting the simulating automaton continue with the same synopsis preserves the invariant.

Finally, the analogue of Lemma 6.12 states that for each regular language $\mathcal{L}$, if $E\mathcal{L}$ is term-registerless, then $\mathcal{L}$ is blindly E-flat. This time there are important differences in the proof; we sketch it below.

We show that if $\mathcal{L}$ is not blindly E-flat, then $[E\mathcal{L}]$ cannot be separated from $[(E\mathcal{L})^{\mathrm{c}}]$ by a finite automaton. Suppose that the minimal automaton $\mathcal{A}$ of $\mathcal{L} \subseteq A^*$ is not E-flat. Let i be the initial state of $\mathcal{A}$. Then, there exist words $s, t, u_1, u_2 \in A^+$, $x \in A^*$ and states p, q such that $|u_1| = |u_2|$, $i \cdot s = p$, $p \cdot u_1 = q \cdot u_2 = q$, $q \cdot x$ is rejecting, and $p \cdot t$ is accepting iff $q \cdot t$ is rejecting. It follows that for each $k > 0$, $su_1(u_2)^k x \in \mathcal{L}^{\mathrm{c}}$, and $st \in \mathcal{L}$ iff $s(u_1)(u_2)^k t \in \mathcal{L}^{\mathrm{c}}$. Unlike for the markup encoding, the construction of the fooling trees depends on whether $st \in \mathcal{L}$ or $st \in \mathcal{L}^{\mathrm{c}}$

Suppose first that $st \in \mathcal{L}^{\mathrm{c}}$. Then, the trees S, S' used in Lemma 6.12 should be replaced with the ones in Fig. 6.8a. We have $S \notin E\mathcal{L}$ and $S' \in E\mathcal{L}$. Note that we have no control on whether the rightmost branch of S' is labelled by a word from $\mathcal{L}$ or not, but it is irrelevant, because we know that the middle branch is. The term encodings of S and S' satisfy the following:

$$\begin{aligned}
[S] &= s \cdot u_1(u_2)^{n!}\, x\bar{x}\,(\bar{u}_2)^{n!}\,\bar{u}_1\, t\bar{t}\, u_1(u_2)^{n!}\, x\bar{x}\,(\bar{u}_2)^{n!}\,\bar{u}_1\,\bar{s}\,,\\
[S'] &= s\, u_1(u_2)^{2\cdot n!}\, x\bar{x}\,(\bar{u}_2)^{n!}\,\bar{u}_2\, t\bar{t}\, u_1(u_2)^{n!}\, x\bar{x}\,(\bar{u}_2)^{n!}\,\bar{u}_1(\bar{u}_2)^{n!-1}\,\bar{u}_1\,\bar{s}\\
&= s\, u_1(u_2)^{2\cdot n!}\, x\bar{x}\,(\bar{u}_2)^{n!}\,\bar{u}_1\, t\bar{t}\, u_1(u_2)^{n!}\, x\bar{x}\,(\bar{u}_2)^{n!}\,\bar{u}_2(\bar{u}_2)^{n!-1}\,\bar{u}_1\,\bar{s}\\
&= s\, u_1(u_2)^{2\cdot n!}\, x\bar{x}\,(\bar{u}_2)^{n!}\,\bar{u}_1\, t\bar{t}\, u_1(u_2)^{n!}\, x\bar{x}\,(\bar{u}_2)^{2\cdot n!}\,\bar{u}_1\,\bar{s}\,,
\end{aligned}$$

because $|u_1| = |u_2|$ implies $\bar{u}_1 = \bar{u}_2$. The rest of the proof is identical.

If $st \in \mathcal{L}$, in S we replace u_1 on the rightmost branch with u_2, and we modify S' accordingly. It then holds that $S \in E\mathcal{L}$ regardless of whether $su_2(u_2)^{n!}x$ belongs to $\mathcal{L}$ or not, and $S' \notin E\mathcal{L}$; the proof again continues like in Lemma 6.12. □

Theorem 6.34.
For each regular language $\mathcal{L}$, the following conditions are equivalent:

- $Q\mathcal{L}$ is a term-stackless unary query;
- $E\mathcal{L}$ is a term-stackless tree language;
- $A\mathcal{L}$ is a term-stackless tree language;

(a) Tree S. (b) Tree S'.

Figure 6.8: Blind variants of fooling trees in Lemma 6.12.

- $\mathcal{L}$ is blindly HAR.

Proof. The argument is fully analogous to that in Theorem 6.27, with Lemmas 6.26, 6.28 and 6.32 replaced by their analogues for term-stackless and blindly HAR languages.

The analogue of Lemma 6.26 states that the class of blindly HAR languages is closed under complement, which is immediate from the definition just like for HAR languages.

The analogue of Lemma 6.28 states that if $\mathcal{L}$ blindly HAR then $Q\mathcal{L}$ is term-stackless. The proof is analogous, with the only modification being what we did with Lemma 6.11 in the proof of Theorem 6.33: when the closing tag $\triangleleft$ is read in state p and the current depth is greater than or equal to the maximal stored depth, we pick any state p' such that $p' \cdot a$ is almost equivalent to p for some $a \in A$. Because $\mathcal{L}$ is blindly HAR, the original argument now shows also that the choice of a does not matter.

Finally, the analogue of Lemma 6.32 states that for each regular language $\mathcal{L}$, if $E\mathcal{L}$ is a term-stackless tree language then $\mathcal{L}$ is blindly HAR. The proof is obtained by adjusting the proof of Lemma 6.32 just like the proof of Lemma 6.12 was adjusted in Theorem 6.33. This time there is only one case because we know that $s(wu_1 + vu_2)^* wt \subseteq \mathcal{L}^c$ and $s(wu_1 + vu_2)^* vt \subseteq \mathcal{L}$, and not the other way around. In the tree R shown in Fig. 6.7, the copies of u immediately following copies of w should be replaced by u_1 and those immediately following v should be replaced by u_2. From there, the proof continues like before. □

6.5 Algebraic characterisations

The definitions of almost-reversible, E-flat, A-flat and hierarchically almost-reversible are combinatorial descriptions of automata. However, the classes of languages they described are closed under many operations, giving them the structure of different types of varieties. This time, we stick with stamps instead of semigroups as algebraic objects for *ne*-varieties, as the descriptions are easier in this setting.

Theorem 6.35.
The following classes are varieties:

- the class of almost-reversible languages is an *ne*-variety,
- the class of E-flat languages is a positive *ne*-variety,
- the class of A-flat languages is a positive *ne*-variety,

- the class of hierarchically almost-reversible languages is an *ne*-variety.
- the class of blindly almost-reversible languages is an *lm*-variety,
- the class of blindly E-flat languages is a positive *lm*-variety,
- the class of blindly A-flat languages is a positive *lm*-variety,
- the class of blindly hierarchically almost-reversible languages is an *lm*-variety.

Proof. All items will be immediate after proving their equivalence with algebraic classes that will form different varieties of stamps. Note that the class of (blindly or not) E-flat and A-flat are not closed under complementation (with, for example, the co-finite and finite languages), hence the need for ordered structures.

However, we give the proof for the closure under non-erasing inverse morphisms of almost-reversible languages to stress that they do not correspond to a variety of monoids. Let $\mathcal{L}$ be a regular language over A recognised by an almost-reversible $\mathcal{A}$, and $\mu : B^* \to A^*$ a non-erasing morphism. It is standard that the language $\mathcal{L}' = \mu^{-1}(\mathcal{L})$ is recognised by the automaton $\mathcal{A}'$ defined as $\mathcal{A}$ with the different transition function $\delta'_a = \delta_{\mu(a)}$. We will show that $\mathcal{A}'$ is almost-reversible. Let p and q be two states and $u \in B^*$ such that $p \cdot u = q \cdot u$. Thus p and q also meet in the automaton $\mathcal{A}$, with the word $\mu(u)$. Therefore p and q are almost-equivalent in $\mathcal{A}$. Let $v \in B^+$, $p \cdot v$ and $q \cdot v$ reach respectively the states that correspond in $\mathcal{A}$ to $p \cdot \mu(v)$ and $q \cdot \mu(v)$. Here we use the non-erasing assumption on μ: $\mu(v)$ is non-empty and then $p \cdot \mu(v) = p \cdot v$ and $q \cdot \mu(v) = q \cdot v$ are equivalent. Therefore p and q are almost equivalent in $\mathcal{A}'$.

For blind classes, we moreover need to preserve the length of words that make two states blindly meet, hence the closure only stands for length-multiplying morphisms. □

6.5.1 Checking first and last letter

A salient ability of our classes is that they can check easily the first and last letter of a word (corresponding to the root and leaves of the corresponding tree). This is captured by the idea of almost-**V** stamps, for **V** any kind of variety. It was first introduced by Grosshans, McKenzie and Segoufin in their study of programs over monoids [52]. It has later been used again by Grosshans in [51] in a general study of the join of varieties with **LI**. This operation is the algebraic counterpart of freely checking finite prefixes and suffixes of words. We adapt this operation to catch the ability of checking only the first and last letters of a word. Additionally, note that the terminology used therein is "essentially-**V**" stamps. We chose to change it to fit with our notations, and to avoid a conflict with the already existing class **EV** of monoids whose idempotents generate a monoid in **V**.

Definition 6.36.
Let **V** be a positive *lm*-variety of stamps. A stamp $\mu : A^* \to M$ is said to be *almost*-**V** whenever there exists a stamp $\eta : A^* \to N$ in **V** such that for every $u, v \in A^*$ we have

$$\eta(u) \leq \eta(v) \Rightarrow (\forall a, b \in A,\ \mu(aub) \leq \mu(avb)).$$

The class of all such stamps is denoted by **AV**. It is a (positive) *ne*-variety of stamps when **V** is a (positive) *ne*-variety of stamps (even if it is a variety of monoids), and a (positive) *lm*-variety of stamps whenever **V** is (see [52]). It naturally has a description in term of quotient of languages.

Lemma 6.37.
Let **V** be a positive *lm*-variety of stamps and let $\mathcal{L}$ be a regular language. We have that $\mathcal{L}$ is in **AV** if and only if $\forall a, b \in A$, $a^{-1}\mathcal{L}b^{-1}$ is in **V**.

Proof. Throughout the proof, let μ be the syntactic stamp of $\mathcal{L}$.

Firstly, assume that the condition on the quotients is satisfied. Consider the collection of stamps in **V** that recognise the languages of the form $a^{-1}\mathcal{L}b^{-1}$, and write η for their product. Since a variety is stable by product, η is in **V**. Let $u, v \in A^*$ such that $\eta(u) \leq \eta(v)$ and $a, b \in A$. We want to show that aub and avb are ordered for the syntactic order of $\mathcal{L}$. Let $x, y \in A^*$, we have to prove $xauby \in \mathcal{L} \Rightarrow xavby \in \mathcal{L}$. Let c and d be the first and last letter of both words: there exist x' and y' in A^* such that $xa = cx'$ and $by = y'd$. The assumption on η gives that $\eta(x'uy') \leq \eta(x'vy')$. Because η recognises $c^{-1}\mathcal{L}d^{-1}$, we know that $x'uy' \in c^{-1}\mathcal{L}d^{-1} \Rightarrow x'vy' \in c^{-1}\mathcal{L}d^{-1}$. This is equivalent to $cx'uy'd \in \mathcal{L} \Rightarrow cx'vy'd \in \mathcal{L}$. Hence $aub \leq_{\mathcal{L}} avb$ and $\mu(aub) \leq \mu(avb)$. This is exactly the definition of μ being almost-**V**.

Secondly, assume that $\mathcal{L}$ is in **AV**. Let $\eta : A^* \to M$ in **V**, given by the condition of almost-**V**. Let a, b be two letters. Let P be the upset of $\eta(a^{-1}\mathcal{L}b^{-1})$, we will show that $\eta^{-1}(P) = a^{-1}\mathcal{L}b^{-1}$ to deduce that $a^{-1}\mathcal{L}b^{-1}$ is recognised by M. The right to left inclusion is true, as $\eta^{-1} \circ \eta$ is always increasing for inclusion. Let $u \in \eta^{-1}(P)$, we have directly that there exists $v \in a^{-1}\mathcal{L}b^{-1}$ such that $\eta(u) \geq \eta(v)$. By definition of almost-**V**, we have that $\mu(aub) \geq \mu(avb)$. We know that $avb \in \mathcal{L}$ and μ recognises $\mathcal{L}$, therefore $aub \in \mathcal{L}$. Hence $u \in a^{-1}\mathcal{L}b^{-1}$. □

We also need a generic way to go from automata definitions based on almost-equivalence, to better-known ones that only use equivalence (that is to say equality whenever we have a minimal automaton).

Let P be a property on automata that takes two states p and q and return a Boolean. We will exclusively consider properties among the following, called *meet properties*:

- P_m: p and q meet.
- P_m^+: p and q meet in q and q is rejective.
- P_m^-: p and q meet in q and q is acceptive.
- P_{ms}: p and q are in a same SCC and meet in this SCC.
- P_{bm}: p and q blindly meet.
- P_m^+: p and q blindly meet in q and q is rejective.
- P_m^-: p and q blindly meet in q and q is acceptive.
- P_{bms}: p and q are in a same SCC blindly and meet in this SCC.

An automaton is *P-collapsing* if every pair of states p, q that has the property P is such that p and q are equivalent. It is *P-almost-collapsing* if every pair of internal states p, q that has the property P is such that p and q are almost-equivalent. The eight classes under study in the complexity of streaming tree (the ones in Theorem 6.35) correspond to the eight classes of P-almost-collapsing automata introduced. We extend these definitions to languages by asking that its minimal automaton is P-collapsing or P-almost-collapsing.

Lemma 6.38.
Let $\mathcal{L}$ be a regular language and P be a meet property. Then $\mathcal{L}$ is P-almost-collapsing if and only if for every $a, b \in A$, $a^{-1}\mathcal{L}b^{-1}$ is P-collapsing.

Proof. First, assume that $\mathcal{L}$ has a P-almost-collapsing minimal automaton $\mathcal{A}$. Let $a, b \in A$. The language $a^{-1}\mathcal{L}b^{-1}$ is recognised by the automaton $\mathcal{A}'$ defined as $\mathcal{A}$ with a different initial and final states. The new initial state is $i' = i \cdot a$, and the new final states are the state q such that $q \cdot b \in F$. We denote this last set F'. Moreover, we trim $\mathcal{A}'$ by removing states that cannot be reached from the initial state. It means in particular that every state in $\mathcal{A}'$ is internal in $\mathcal{A}$. Take two states p and q of $\mathcal{A}'$ such that $P(p, q)$ stands and a letter c such that $p \cdot c = q \cdot c$. It is clear that P is also satisfied for p and q seen as states of $\mathcal{A}$. Thus both p and q are internal and satisfy P in $\mathcal{A}$ and are therefore almost-equivalent in $\mathcal{A}$. To show that p and q are equivalent, we have to see that both or none of them are final. This is indeed the case: by minimality of $\mathcal{A}$ and the almost-equivalence of p and q we have $p \cdot b = q \cdot b$. Thus for both p and q, they are final in $\mathcal{A}'$ if and only if $p \cdot b$ is final in $\mathcal{A}$. Thus $a^{-1}\mathcal{L}b^{-1}$ is a P-collapsing language.

Second, assume that the conditions on quotients is fulfilled. For $a, b \in A$, let $\mathcal{A}_{a,b}$ be a P-collapsing automaton that computes $a^{-1}\mathcal{L}b^{-1}$. By doing a product of them, we can assume that they all have the same states Q and transition function δ, but with different final states. Indeed a product of P-collapsing automata is itself P-collapsing. We construct $\mathcal{A}$ to be the union of all of these automata with a fresh initial state i, and a bunch of states j_a for $a \in A$. We denote the other states $p_{a,b}$ for $p \in Q$ and $a, b \in A$. In particular, $i_{a,b}$ is the initial state of $\mathcal{A}_{a,b}$. For each letter a, b, we add a transition $i \xrightarrow{a} j_a$, and a transition $j_a \xrightarrow{b} (i_{a,b} \cdot b)$. The other transitions are $p_{a,b} \xrightarrow{c} q_{a,c}$ where $q = \delta(p)$. The final states are i iff $\varepsilon \in \mathcal{L}$; j_a iff $a \in \mathcal{L}$; and $p_{a,b}$ iff $\delta_b^{-1}(p)$ is final in $\mathcal{A}_{a,b}$. It is clear that $\mathcal{A}$ recognises $\mathcal{L}$.

It remains to see that $\mathcal{A}$ is P-almost-collapsing. Two important properties of this automaton are:

i) The image of any $p_{a,b}$ and $q_{a,c}$, with p and q equivalent, under the transition function for any letter is two equivalent states, hence they are almost-equivalent.
ii) If $p_{a,b}$ and $q_{a,c}$ have the property P, then p and q have the property P in $\mathcal{A}_{a,b}$, for any possibility for P.

Consider two internal nodes that meet. We can make several other observations: the only non-internal node is i; two distinct nodes j_a and j_b never have P; and two nodes $p_{a,b}$ and $q_{c,d}$ with $a \neq c$ never have P. Hence there are two cases to consider.

- The nodes $p_{a,b}$ and $q_{a,c}$ have P. This gives that p and q have P in $\mathcal{A}_{a,b}$ thanks to ii), and are then equivalent. Then by i) they are almost-equivalent.
- The nodes $p_{a,b}$ and j_a have P. The same reasoning stands with the definition of the outgoing transitions from j_a.

□

Therefore, to characterise the classes of interest, we apply both Lemma 6.37 and Lemma 6.38. All is left to do is algebraically characterise the classes of P-collapsing languages. It alleviates the problem of dealing with the ability to check freely the first and last letters. Plus, these classes are already known in some cases.

Remark 6.39.
Similar results could be obtained for more than one letter, and for only checking first or last letters.

(a) −-reversible automaton for $\mathcal{L}$ (b) Minimal automaton of $\mathcal{L}$

Figure 6.9: Automata recognising $\mathcal{L} = \{a, ab, ac\}$.

6.5.2 Equivalences

Almost-reversible languages. We first tackle the class of regular languages $\mathcal{L}$ such that $Q\mathcal{L}$ is registerless. The class of almost-reversible languages is syntactically the same as the class of P_m-almost-collapsing languages. By Lemma 6.38, it is equivalent to asking that for every letter a, b, the quotients $a^{-1}\mathcal{L}b^{-1}$ are all P_m-collapsing. A minimal P_m-collapsing automaton is such that no two distinct states meet. This amounts to requiring that the transition functions are injective (and therefore bijective). These automata with such transitions are well known.

Fact 6.40 (Folklore)**.**
Let $\mathcal{L}$ be a regular language and M its syntactic monoid. Then $\mathcal{L}$ is P_m-collapsing iff M is a group.

Recall that **G** is the variety of monoids that are groups. Using Lemma 6.37, we have that $\mathcal{L}$ is almost-reversible if and only if its syntactic monoid is in **AG**. For the other equivalences, we will only give the characterisation of P-collapsing automata, without refering to Lemma 6.37. The results are summarised in Theorem 6.45.

E-flat and A-flat languages. We then tackle the class of regular languages $\mathcal{L}$ such that $E\mathcal{L}$ (resp. $A\mathcal{L}$) is registerless. This time, we consider the meet properties P_m^+ and P_m^-. An automaton is +-reversible (resp. −-reversible) if after removing all accepting (resp. rejecting) sink states all the (partial) transition functions are injective. This notion has been studied by Pin [96]. Let $\mathbf{G}^+$ be the positive variety of all ordered monoids whose idempotents commute and such that

$$\forall e \in M \text{ idempotent }, \ e \geq 1.$$

The positive variety $\mathbf{G}^-$ is defined similarly with $e \leq 1$. Pin showed that a language is recognised by a +-reversible (resp. −-reversible) automaton if and only if its syntactic ordered monoid is in $\mathbf{G}^+$ (resp. $\mathbf{G}^-$). Note that this is not equivalent to having its minimal automaton that is +-reversible (resp. −-reversible). For instance, there is the example of $\mathcal{L} = \{a, ab, ac\}$ in [96]. A −-reversible automaton recognising $\mathcal{L}$ and its minimal automaton are given in Fig. 6.9. The rejecting sink state is not drawn. Remark that the minimal automaton is not −-reversible.

Fact 6.41.
Let $\mathcal{L}$ be a regular language and M its syntactic ordered monoid. We have the following

equivalences:

- $\mathcal{L}$ is P_m^+-collapsing iff M is in $\mathbf{G}^+$.
- $\mathcal{L}$ is P_m^--collapsing iff M is in $\mathbf{G}^-$.

Proof. We prove the claim for P_m^+-collapsing automata. First, assume that the minimal automaton $\mathcal{A}$ of $\mathcal{L}$ is P_m^+-collapsing.

- Let x, y be two words such that δ_x and δ_y are idempotents, and p a state. This implies that $(p \cdot x) \cdot x = p \cdot x$ and $(p \cdot y) \cdot y = p \cdot y$. If $p \cdot x$ and $p \cdot y$ are the sink states, then $p \cdot xy = p \cdot yx$. If only $p \cdot x$ is the accepting sink state, then $p \cdot y$ and p meet in $p \cdot y$ and therefore $p = p \cdot y$ by the collapsing property. So $p \cdot yx = p \cdot x = p \cdot xy$. Last, if both are not the sink state then with the same reasoning we have that $p \cdot xy = p = p \cdot yx$. So $\delta_x \delta_y = \delta_y \delta_x$ and the idempotents of M commute.
- Let e, z be two words such that δ_e is idempotent. Let p be a state such that $p \cdot z \in F$. If $p \cdot e$ is the accepting sink state then $p \cdot e \cdot z$ is in F as well. If not, then $p \cdot e = p$ and therefore $p \cdot e \cdot z = p \cdot z$ is in F. So $\delta_e \geq \delta_\varepsilon$, and 1 is smaller than every idempotent in M.

So M satisfies all the equations of $\mathbf{G}^+$.

Second, assume that M is in $\mathbf{G}^+$. This means that $\mathcal{L}$ is recognised by an automaton $\mathcal{A}$ which is +-reversible. Let p, q be two rejective states such that p and q meet in q. With the facts that q is rejective and that the partial transition function in $\mathcal{A}$ without the accepting sink states is injective, we have that p and q are equivalent. They will therefore be merged during the minimisation. Hence the minimal automaton of $\mathcal{L}$ is P_m^+-collapsing. □

Hierarchically almost-reversible languages. We continue our algebraic investigation with the study of regular languages $\mathcal{L}$ such that $Q\mathcal{L}$ (resp. $A\mathcal{L}$, $E\mathcal{L}$) is stackless. The meet property under consideration is P_{ms}. The class of P_{ms}-collapsing automata is exactly the class of automata whose transition (partial) functions restricted to any SCC are injective.

We denote by **ER** the class of monoids such that their idempotents generate an $\mathcal{R}$-trivial monoid. It is defined by the equations (see Exercice 5.2.8 in [2]): for any two idempotents e and f we have that

$$(ef)^\omega e = (ef)^\omega.$$

Fact 6.42.
Let $\mathcal{L}$ be a regular language and M be its syntactic monoid. Then $\mathcal{L}$ is P_{ms}-collapsing iff M is in **ER**.

Proof. First, assume that $\mathcal{A}$, the minimal automaton of $\mathcal{L}$, is not P_{ms}-collapsing. Then there exists two distinct states p and q that meet in their SCC. This means there are three words u, x and y, and a state r such that $p \cdot u = q \cdot u = r$ and $r \cdot x = p$ and $r \cdot y = q$. We consider the idempotent in the syntactic monoid: $e = (\delta_u \delta_x)^\omega$ and $f = (\delta_u \delta_y)^\omega$. Their key properties is that e maps both p and q to p, and f maps both p and q to q. This implies that $(ef)^\omega e$ maps p to p and that $(ef)^\omega$ maps p to q. Hence $(ef)^\omega e \neq (ef)^\omega$, from which $M \notin$ **ER** follows.

Second, assume that $\mathcal{A}$ is P_{ms}-collapsing. Let e and f be two idempotents of M, with δ_x and δ_y the corresponding transition functions. Let p be a state of $\mathcal{A}$. Consider the sequence of states $(\delta_x \delta_y)^i(p)$. By the pigeonhole principle, there are i and j such that $(\delta_x \delta_y)^i(p) = (\delta_x \delta_y)^j(p) = p_0$.

We give a name to all states in between:

$$p_0 \xrightarrow{x} q_0 \xrightarrow{y} p_1 \xrightarrow{x} \cdots \xrightarrow{x} q_{j-i-1} \xrightarrow{y} p_{j-i} = p_0.$$

Of course, all of those states are in the same SCC. However, δ_x is idempotent therefore for any k, $\delta_x(q_k) = \delta_x^2(p_k) = \delta_x(p_k) = q_k$. We obtain $\delta_y(p_k) = p_k$ as well. We can reach q_k with x from p_k and q_k, thus by the injectivity of the transition functions in a SCC: $p_k = q_k$. We obtain $q_{k-1} = p_k$ as well. In the end, we have the collapse $p_0 = q_0 = p_1 = \cdots = q_{j-i-1}$. Hence, $(ef)^\omega$ maps p to p_0 and $(ef)^\omega e$ maps p to p_0 too. This stands for all states p, so $(ef)^\omega$ and $(ef)^\omega e$ are the same functions. We conclude that $\mathcal{A}$ satisfies the equations of **ER**. □

Blindly almost-reversible languages. Next in line is the study of languages $\mathcal{L}$ such that $Q\mathcal{L}$ is term-registerless. The meet property is P_{bm}. Recall that **I** is the trivial variety consisting only of the monoid of size 1, and hence **QI** is the variety of stamps whose stable monoid is trivial.

Fact 6.43.
Let $\mathcal{L}$ be a regular language and μ be its syntactic stamp. Then $\mathcal{L}$ is blindly reversible if and only if μ is in **QI**.

Proof. Let $\mathcal{A}$ be the minimal automaton of $\mathcal{L}$. Let s be the stability index of μ. It follows from the definition of the operator **Q** that μ is in **QI** iff for all word x of size s, δ_x is the identity function.

First, assume that $\mathcal{A}$ is P_{bm}-collapsing. Let x be a word of size s and p a state. Let $q = p \cdot x$. By the definition of the stable monoid, there exist a word y of size s such that $\delta_{x^2} = \delta_y$. We have $p \cdot y = p \cdot x \cdot x = q \cdot x$, and thus p and q blindly meet. Then, by blind reversibility and minimality of $\mathcal{A}$, $p = q$. Thus $p \cdot x = p$, and δ_x is the identity. We conclude that $\mu \in$ **QI**.

Second, assume that $\mathcal{A}$ is not P_{bm}-collapsing. It means that there exists two distinct states p, q and two words x, y of the same length such that $p \cdot x = q \cdot y$. By adding any same suffix after x and y we can assume that x and y are of size s. Moreover, at least one of δ_x or δ_y is not the identity function. Hence the stable monoid of μ is not trivial, and $\mu \notin$ **QI**. □

Blindly E-flat and A-flat. The study of regular languages $\mathcal{L}$ such that $E\mathcal{L}$ (resp. $A\mathcal{L}$) is term-registerless is more complicated. Understanding what are blindly E-flat and A-flat languages requires to understand partial automata that have their transition functions globally injective, in the spirit of the work of Pin. This will not be done in this work.

Blindly hierarchically almost-reversible languages. Last, we go on with regular languages $\mathcal{L}$ such that $Q\mathcal{L}$ (resp. $A\mathcal{L}$, $E\mathcal{L}$) is term-stackless. The meet property is P_{bms}. We recall that **R** is the variety of $\mathcal{R}$-trivial monoids and is defined by the equations $(xy)^\omega x = (xy)^\omega$.

Fact 6.44.
Let $\mathcal{L}$ be a regular language and μ be its syntactic stamp. Then $\mathcal{L}$ is P_{bms}-collapsing if and only if μ is in **QR**.

Proof. Let $\mathcal{A}$ be the minimal automaton of $\mathcal{L}$, s be the stability index of μ and M_s the stable monoid of μ.

First assume that $\mathcal{A}$ is P_{bms}-collapsing. Let x and y be two words of size s and p be a state. We consider the sequence of states $p, p \cdot xy, p \cdot (xy)^2, \cdots$. By the pigeonhole principle, there are two indices i and j such that $p \cdot (xy)^i = p \cdot (xy)^j$. We give a name to all states in between:

$$p_0 \xrightarrow{x} q_0 \xrightarrow{y} p_1 \xrightarrow{x} \cdots \xrightarrow{x} q_{j-i-1} \xrightarrow{y} p_{j-i} = p_0.$$

All of those states are in the same SCC. Let k be an integer, $(xy)^{j-i}$ is a word of size $(j-i)s$ and therefore there exists some z of size s with the same transition function. Hence $p_k \cdot z = p_k$ and thus q_{k-1} and p_k blindly meet and are in the same SCC: they are equal by the P_{bms}-collapsing property. With the same argument we have that $q_k = q_{k-1}$. So $p_0 = q_0 = \ldots = q_{j-i-1}$. In the end, $(xy)^\omega$ and $(xy)^\omega x$ both map p to p_0. So $(xy)^\omega x$ and $(xy)^\omega$ define the same functions, and M_s satisfies the equations of **R**.

Second, assume that $\mathcal{A}$ is not P_{bms}-collapsing. Then there exist three states in the same SCC $p \neq q$ and r and two words x, y of the same size such that $p \cdot x = q \cdot y = r$. By adding any same letters after x and y, we can assume that x and y are both of size s. Let also z, t be two words such that $r \cdot z = p$ and $r \cdot t = q$. We can reach q from p with the word $x(ty)^{s-1}t$. Because the sizes of x and y are multiples of s we have that $(ty)^{s-1}t$ is of size a multiple of s as well. Hence there is a word x' of size s such that $p \cdot x' = q$. Symmetrically, there is a word y' of size s such that $q \cdot y' = p$. Thus $(x'y')^\omega$ maps p to itself, and $(x'y')^\omega x'$ maps p to q. They therefore do not define the same function, and M_s does not satisfies the equations of **R**.

□

Summary. We can now wrap everything together, and state the theorem that characterises algebraically all of our classes. It gives another way, directly on the syntactic algebraic objects, to know whether a path language of tree can be weakly validated (resp. queried) in constant and logarithmic memory.

Theorem 6.45.
Let $\mathcal{L}$ be a regular language and let μ be its syntactic (ordered if needed) stamp. Then:

- $\mathcal{L}$ is almost reversible iff $\mu \in \mathbf{AG}$.
- $\mathcal{L}$ is E-flat iff $\mu \in \mathbf{AG}^+$.
- $\mathcal{L}$ is A-flat iff $\mu \in \mathbf{AG}^-$.
- $\mathcal{L}$ is hierarchically almost reversible iff $\mu \in \mathbf{AER}$.
- $\mathcal{L}$ is blindly almost reversible iff $\mu \in \mathbf{AQI}$.
- $\mathcal{L}$ is blindly hierarchically almost reversible iff $\mu \in \mathbf{AQR}$.

6.6 Going further

There are several directions for future research. A straightforward line of work is to study automata with globally injective transition functions. This would allow to give an algebraic characterisation of blindly E-flat and blindly A-flat languages, and to complete the picture given by Theorem 6.45. It will very probably be of the shape **AQV** for some variety of monoids **V** that has to be found.

As we have seen, both registerless and stackless models are efficient but their expressive power is limited. For instance, they cannot compute queries that are about children. However,

the documents that are processed usually have some additional structure, in particular if there is available information on their sources. The next question is to use this knowledge to process documents more efficiently. The general problem is *validation against* $\mathcal{S}$: given two regular tree languages $\mathcal{L}$ (the target) and $\mathcal{S}$ (the schema), determine the complexity of deciding that a given tree is in $\mathcal{L}$ assuming that it belongs to $\mathcal{S}$. At first, it would be nice to characterise for a fixed schema all regular word languages $\mathcal{L}$ such that $Q\mathcal{L}$, $E\mathcal{L}$ or $A\mathcal{L}$ can be validated against $\mathcal{S}$ in constant space. This chapter obtained this result for the schema of all trees, in which case it is equivalent to weak validation. It is also clear that when $\mathcal{S}$ only has trees of bounded depth then all path languages can be validated against $\mathcal{S}$ in constant space. Schemas with limited amount of branching appear too to allow more languages to be processed efficiently. A crucial schema is the set of trees that satisfy constraints demanded for XML documents, and understanding which languages can be validated against this schema would shed some light on the processing of real-life documents. Nevertheless, studying schemas is a complicated task and not much is yet known.

Another line of research, already properly identified, is to have a decidable characterisation of the regular tree languages that are weakly validatable in constant space. Our Theorem 6.4 on horizontal forest algebras, while not known to give decidability, allows to use algebraic techniques. Indeed, we can decide if a forest algebra (H, V) is horizontal but not if it admits an expansion (that is to say that (H, V) is a submonoid of the expansion) which is horizontal. For example, thanks to the lower bound, we know that every language that is weakly validatable in constant space must satisfies the equations: $v^\omega(x) + y + v^\omega(z) = v^\omega(v^\omega(x) + y + v^\omega(z))$ for every context v and forests x, y, z. Can these equations, possibly with some others, help to construct an horizontal expansion? Answering this question would yield an answer to the longstanding open question of Segoufin and Vianu. This algebraic approach can help as well when documents are encoded with the term encoding, that is to say the closing tag does not carry any label information. In this case, the horizontal property becomes the blind horizontal property: there exists $y \in H$, such that for all $v \in V$ there exists $x_v \in H$ such that $v(h) = x_v + h + y$. This uniformity on y makes the analysis simpler and yields decidability of the problem (when a neutral letter is present). Note that this was already known in [9], but our approach seems more generalisable to the markup encoding setting.

To go even further, the ultimate goal would be to have a trichotomy on the space complexity of weakly validating documents (both for markup and term encoding). It would have the following form:

Conjecture 6.46.
Let $\mathcal{L}$ be a regular tree language. Then one and only one of the following is true, where h is the height of a tree:

- $\mathcal{L}$ is weakly validatable in constant space.
- $\mathcal{L}$ is weakly validatable in $\Theta(\log(h))$ space.
- $\mathcal{L}$ is weakly validatable in $\Theta(h)$ space.

A stronger result would be to have a decidable characterisation of all these classes. In this work, we made progress on this conjecture for path languages of trees, and strongly suspect that the stackless model captures exactly what can be done with a logarithmic amount of space available.

Finally, continuing this study can lead to optimisation of actual computations, notably with the help of vectorisation. As pointed out in the introduction, it has already be done in RsonPath [45]. But it can be even improved further with the advance of theoretical results. For

instance, having results on schemas could give better algorithms amenable to vectorisation.

Bibliography of the current chapter

[2] Jorge Almeida. *Finite Semigroups and Universal Algebra*. World Scientific, 1995. DOI: 10.1142/2481.

[9] Vince Bárány, Christof Löding, and Olivier Serre. "Regularity Problems for Visibly Pushdown Languages". In: *Proc. STACS*. Springer, 2006. DOI: 10.1007/11672142_34.

[11] Corentin Barloy, Filip Murlak, and Charles Paperman. "Stackless Processing of Streamed Trees". In: *PODS*. June 2021. DOI: 10.4230/LIPIcs.

[15] David A. Mix Barrington and James C. Corbett. "On the Relative Complexity of Some Languages in NC_1". In: *Inf. Process. Lett.* 32.5 (1989). DOI: 10.1016/0020-0190(89)90052-5.

[20] Burchard von Braunmühl and Rutger Verbeek. "Input-Driven Languages are Recognized in log n Space". In: *Proc. FCT 1983*. Springer, 1983. DOI: 10.1007/3-540-12689-9_92.

[25] Robert D. Cameron, Ehsan Amiri, Kenneth S. Herdy, Dan Lin, Thomas C. Shermer, and Fred Popowich. "Parallel Scanning with Bitstream Addition: An XML Case Study". In: *Proc. Euro-Par 2011*. Springer, 2011. DOI: 10.1007/978-3-642-23397-5_2.

[27] Cristiana Chitic and Daniela Rosu. "On Validation of XML Streams Using Finite State Machines". In: *Proc. WebDB 2004*. ACM, 2004. DOI: 10.1145/1017074.1017096.

[31] Denis Debarbieux, Olivier Gauwin, Joachim Niehren, Tom Sebastian, and Mohamed Zergaoui. "Early nested word automata for XPath query answering on XML streams". In: *Theor. Comput. Sci.* 578 (2015). DOI: 10.1016/j.tcs.2015.01.017.

[36] Patrick Dymond. "Input-driven Languages Are in Log N Depth". In: *Inf. Process. Lett.* 26.5 (Jan. 1988). DOI: 10.1016/0020-0190(88)90148-2.

[44] Evangelos Georganas, Sasikanth Avancha, Kunal Banerjee, Dhiraj D. Kalamkar, Greg Henry, Hans Pabst, and Alexander Heinecke. "Anatomy of high-performance deep learning convolutions on SIMD architectures". In: *Proc. SC 2018*. IEEE / ACM, 2018. DOI: 10.5555/3291656.3291744.

[45] Mateusz Gienieczko, Filip Murlak, and Charles Paperman. "Supporting Descendants in SIMD-Accelerated JSONPath". In: *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*. ASPLOS '23. , Vancouver, BC, Canada, Association for Computing Machinery, 2024. DOI: 10.1145/3623278.3624754.

[50] Alejandro Grez, Cristian Riveros, and Martín Ugarte. "A Formal Framework for Complex Event Processing". In: *Proc. ICDT 2019*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. DOI: 10.4230/LIPIcs.ICDT.2019.5.

[51] Nathan Grosshans. "A Note on the Join of Varieties of Monoids with LI". In: *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*. 2021. DOI: 10.4230/LIPIcs.MFCS.2021.51.

[52] Nathan Grosshans, Pierre Mckenzie, and Luc Segoufin. "The Power of Programs over Monoids in DA". In: *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*. 2017. DOI: 10.4230/LIPIcs.MFCS.2017.2.

[53] Sascha Grunert and Daniel Schmidt. *A comparison of regex engines*. 2017. URL: https://rust-leipzig.github.io/regex/2017/03/28/comparison-of-regex-engines/.

[54] Ashish Kumar Gupta and Dan Suciu. "Stream Processing of XPath Queries with Predicates". In: *Proc. SIGMOD 2003*. ACM, 2003. DOI: 10.1145/872757.872809.

[60] Yeye He, Siddharth Barman, and Jeffrey F. Naughton. "On Load Shedding in Complex Event Processing". In: *Proc. ICDT 2014*. OpenProceedings.org, 2014. DOI: 10.5441/002/icdt.2014.23.

[70] Eryk Kopczynski. "Invisible Pushdown Languages". In: *Proc. LICS 2016*. ACM, 2016. DOI: 10.1145/2933575.2933579.

[75] Geoff Langdale and Daniel Lemire. "Parsing gigabytes of JSON per second". In: *VLDB J.* 28.6 (2019). DOI: 10.1007/s00778-019-00578-5.

[82] Filip Murlak, Charles Paperman, and Michal Pilipczuk. "Schema Validation via Streaming Circuits". In: *Proc. PODS 2016*. ACM, 2016. DOI: 10.1145/2902251.2902299.

[84] Dan Olteanu. "SPEX: Streamed and Progressive Evaluation of XPath". In: *IEEE Trans. Knowl. Data Eng.* 19.7 (2007). DOI: 10.1109/TKDE.2007.1063.

[85] Shoumik Palkar, Firas Abuzaid, Peter Bailis, and Matei Zaharia. "Filter Before You Parse: Faster Analytics on Raw Data with Sparser". In: *Proc. VLDB Endow.* 11.11 (2018). DOI: 10.14778/3236187.3236207.

[96] Jean-Eric Pin. "On reversible automata". In: *Proc. LATIN 1992*. Springer, 1992.

[105] Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. "Rethinking SIMD Vectorization for In-Memory Databases". In: *Proc. SIGMOD 2015*. ACM, 2015. DOI: 10.1145/2723372.2747645.

[110] Gang Ren, Peng Wu, and David A. Padua. "An Empirical Study On the Vectorization of Multimedia Applications for Multimedia Extensions". In: *Proc. IPDPS 2005*. IEEE, 2005. DOI: 10.1109/IPDPS.2005.94.

[115] Luc Segoufin and Cristina Sirangelo. "Constant-Memory Validation of Streaming XML Documents Against DTDs". In: *Proc. ICDT 2007*. Springer, 2007. DOI: 10.1007/11965893_21.

[116] Luc Segoufin and Victor Vianu. "Validating Streaming XML Documents". In: *Proc. PODS 2002*. ACM, 2002. DOI: 10.1145/543613.543622.

[128] Dan Suciu. "From searching text to querying XML streams". In: *J. Discrete Algorithms* 2.1 (2004). DOI: 10.1007/3-540-45735-6_2.

[135] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. *Improving the speed of neural networks on CPUs*. 2011.

[137] Xiang Wang, Yang Hong, Harry Chang, KyoungSoo Park, Geoff Langdale, Jiayu Hu, and Heqing Zhu. "Hyperscan: A Fast Multi-pattern Regex Matcher for Modern CPUs". In: *Proc. NSDI 2019*. USENIX Association, 2019.

[140] Haopeng Zhang, Yanlei Diao, and Neil Immerman. "On complexity and optimization of expensive queries in complex event processing". In: *Proc. SIGMOD 2014*. ACM, 2014. DOI: 10.1145/2588555.2593671.

[141] Yichun Zhang. *Regex Engine Matching Speed Benchmark*. 2015. URL: http://openresty.org/misc/re/bench/.

[142] Jingren Zhou and Kenneth A. Ross. "Implementing database operations using SIMD instructions". In: *Proc. SIGMOD 2002*. ACM, 2002. DOI: 10.1145/564691.564709.

Chapter 7

Incremental Complexity: Maintaining Regular Languages under Small Changes

Outline of the current chapter

The section on the RAM model is based on unpublished work with Antoine Amarilli, Louis Jachiet and Charles Paperman. The section on dynamic first-order logic is based on unpublished work with Felix Tschirbs and Nils Vortmeier.

Complexity questions usually consider worst-case (or average-case) of the execution of an algorithm. However, it is frequent that a same algorithm has to be run several times on instances that are close. In this case, we can often do better than the naive approach that consists in recomputing everything from scratch every time. Indeed, we can store the intermediate state of computation, and potentially more, to help future computations. We assume that the input is changed one bit at a time, and that the auxiliary data has to be maintained efficiently. This

approach to complexity has been formalised by Miltersen, Subramanian, Vitter and Tamassia [81]. They introduced in particular the class incr-POLYLOGTIME of problems that can be dynamically maintained with updates in polylogarithmic time, and studied its relation with the class P.

Another point of view, coming from database theory, comes from Dong, Su and Topor [34, 35] and Patnaik and Immerman [90]. Here, we want to evaluate a query on a database (in the relational model) with the help of auxiliary data that are updated whenever there is a change in the database. For the updates, motivated by the success of logic in the relational model, the choice is to use first-order formulae: giving the class known as Dyn-FO. See Schwentick and Zeume [114] for a survey on Dyn-FO.

Moreover, these incremental complexity classes seem to be related to circuit complexity classes. Cohen and Tamassia [28] discussed incremental algorithm to maintain the ouput of special classes of circuits. In Miltersen et al. [81, Section 6], it is shown that some problem that is hard for circuit classes can be solved efficiently by a dynamic algorithm, exhibiting differences between the two paradigms. Also, first-order logic is known to correspond to a uniform version of AC^0. Finally, in the framework of Dyn-FO, some lower bounds against circuit classes gave lower bounds against dynamic classes, showing a transfer of results from the parallel world to the dynamic world. Indeed Dong and Su [33] (giving credit to Miltersen) used a result of Cai [24] on the PARITY language and AC^0 to show the existence of an arity hierarchy inside Dyn-FO.

Regarding the success of looking at regular languages inside circuit classes, an interesting question is to look at regular languages inside dynamic classes. Maintaining regular language in the RAM model has been studied by Skovbjerg Frandsen, Miltersen and Skyum [122]. Their study has been completed with a complete classification of the dynamic complexity of regular languages by Amarilli, Jachiet and Paperman [7]. Regular languages inside Dyn-FO have been investigated by Patnaik and Immerman [90], and continued by Hesse [61]. Gelade, Marquardt and Schwentick [43] proved later that the regular languages are exactly the languages in Dyn-Prop, the quantifier-free restriction of Dyn-FO. Tschirbs, Vortmeier and Zeume looked at the maintenance of regular languages under large batch changes.

In this chapter, we propose in a first time to extend the result of Amarilli, Jachiet and Paperman [7] to the setting of regular languages of trees. Then we continue by looking at the fine-grained complexity of regular languages inside Dyn-FO, by restricting the available memory size that the auxiliary structures can access.

7.1 Two models

In this chapter, we study the incremental maintenance of regular languages, both for words and for trees. The general problem is the following, and is parametrised by a language $\mathcal{L}$ over an alphabet A. At first, there is a word $w = \bot^n$ for n an integer and $\bot$ a fresh symbol. There is then an infinite stream of operations on w of the two following sorts:

- $\mathbf{SET}_a(i)$ for $a \in A \cup \{\bot\}$ and $1 \leq i \leq n$: sets the i^{th} position of w to the value a.
- **MEMBER** returns whether w is in $\mathcal{L}$, by ignoring the letter $\bot$.

This is the *dynamic problem for* $\mathcal{L}$.

We wish to study data structures that can be maintained to answer this problem efficiently. By that, we mean that the structures can be modified efficiently after each **SET** operation, such that **MEMBER** operations are also efficient. In the problem, n is given as part of the input but is never changed by the stream of operations.

An important class of dynamic problems is related to monoids. Let M be a monoid. We say that a word $w \in M^*$ evaluates to $x \in M$ if the product of its letters is x. In this case, the alphabet is $A = M$ and the target language is the set of words of M that evaluate to some fixed $x \in M$.

This problem is called the *dynamic problem for M at x*. We extend the notation for x replaced by a subset P of M. If we can solve the dynamic problem for M at every x, we just say that we can solve the dynamic problem for M. Roughly speaking, if the set of regular languages for which the dynamic problem is in a given complexity class forms a variety of languages, then it is enough to look at dynamic problems for monoids instead of languages.

This setting can be extended for regular languages of trees. We start with any tree with only $\perp$ labels, and there are operations $\mathbf{SET}_a(i)$ for every node i. The algebraic counterpart problem becomes the dynamic problem for (H, V) at $x \in H$ where (H, V) is a forest algebra, that takes a tree in V^Δ and accepts if its evaluation is x. To evaluate a leaf, it feeds the corresponding element of V with the identity of H.

7.1.1 The RAM model

In the literature, the RAM (Random Access Machine) model is defined in many different ways. To uniformise this model, Grandjean and Jachiet [48] recently proposed a unified and robust definition for the RAM model. We stick to this notion. This model has registers that can store positive integers, and instructions that can be used to manipulate integers and to manage the memory. The input is stored as a size n, and an array of integers of size n. Both words and trees can be represented in this fashion. In the case of trees, we assume that each node has a set of pointers to its children. The instructions available consist in a basic set of assembly-like instructions (with loading of a register value and conditional structures), and with the addition as only arithmetic operation. The key features of the RAM model are:

- **logarithmic word length:** the registers can store an integer of value at most polynomial in the input size n. This is equivalent to requiring that the registers have logarithmic size,
- **unit cost:** every instruction, in particular the addition, takes a unit time to be executed.

The time of execution of an algorithm in the RAM model is the number of executed unit cost operations. In [48], it is shown that with the addition only it is possible to emulate many other arithmetic operations as successor, multiplication, division, modulo, squaring, ...

Definition 7.1.
A *dynamic RAM program* $\mathcal{P}$ consists in:

- an initialisation algorithm that runs in linear time,
- an update algorithm that takes as parameters an integer and a letter,
- a membership algorithm that returns a Boolean.

All algorithms run in the RAM model and also take the maintained tree as a parameter.

Given a dynamic problem for a regular language $\mathcal{L}$ and a dynamic RAM program $\mathcal{P}$, we initialise data structures in memory with the initialisation algorithm. Afterwards, when an operation $\mathbf{SET}_a(i)$ is seen, we run the update algorithm on (i, a) to update the memory. Finally, when an operation **MEMBER** is seen, we run the membership algorithm to have a Boolean, with the help of the auxilliary data structures. We say that $\mathcal{P}$ is a dynamic RAM program for $\mathcal{L}$ if the returned Boolean is 1 if and only if the maintained word is in $\mathcal{L}$. We are interested in the execution time of the update and membership algorithms. For $f : \mathbb{N} \to \mathbb{N}$ a function, we say that $\mathcal{P}$ is in $O(f)$ (resp. $o(f)$) if the worst case execution of the update and membership algorithm is bounded by a function in $O(f)$ (resp. $o(f)$).

We especially consider constant time executions: $f : n \mapsto 1$. Limiting the preprocessing time is important, as otherwise we could compute all possible executions in exponential time (and

memory) and perform the updates and membership queries in constant time for any language.

As usual lower bounds are hard to find. There are only a few of them, all proven in the *cell probe model* that only considers the number of registers accessed during a computation. For an integer d, let $\mathbb{Z}_d$ be the cyclic group over d elements $\{0,\dots,d-1\}$ with the addition modulo d. One of the problems we know hard is the *prefix-$\mathbb{Z}_d$* problem: given a maintained word w in $\mathbb{Z}_d^*$, returns the evaluation of the prefix $w_1\cdots w_i$ for a membership query with a parameter $i \in \mathbb{N}$.

Theorem 7.2 (Fredman and Saks [40, Theorem 3])**.**
For any $d \geq 2$, there is no dynamic RAM program in $o(\log(n)/\log(\log(n)))$ for the prefix-$\mathbb{Z}_d$ problem.

One last interesting problem in *prefix-U_1*. The monoid U_1 is defined as the syntactic monoid $(a+b)^*a(a+b)^*$. It has two elements: one is neutral and the other a zero. This time, the maintained word has two elements and we want to return the evaluation in U_1 of a prefix given a position. This problem is only conjectured to be hard.

Conjecture 7.3 (Amarilli, Jachiet and Paperman [7, Conjecture 2.3])**.**
There is no dynamic RAM program in $O(1)$ for the prefix-U_1 problem.

7.1.2 Dynamic first-order logic

This setting will only be studied for words, and will therefore be formulated to fit the word case. This time, the auxiliary data structures that can be maintained are relationaldata bases.

A *relational schema* is a finite set of symbols called *relations*. Each relation comes with a fixed arity. A relation of arity 1 (resp. 2) is called unary (resp. binary). In this work, a *domain* is a set of integers of the form $\{1,\dots,n\}$. With an abuse of language, we say in this case that the domain is the integer n. A *database instance* $\mathcal{D}$ of a schema $\mathcal{S}$ over a domain n is a mapping from $\mathcal{S}$ to set of tuples. Namely, a relation R is mapped to a set of tuples of $\{1,\dots,n\}^k$ for k the arity of R. We will usually denote tuples with the typography $\bar{x}$, and $\bar{x}_i$ for its i^{th} component. For $\bar{x}$ a tuple, we denote by $R(\bar{x})$ the fact that $\bar{x}$ is in the image of R.

The *alphabet schema* is $\mathcal{S}_A = \{W_a \mid a \in A \cup \{\bot\}\}$, where every relation is of arity one. A finite word w of size n is represented as a database instance of $\mathcal{S}_A$ over n with, for $a \in A \cup \{\bot\}$ and $1 \leq i \leq n$:

$$W_a(i) \text{ if and only if } w_i = a.$$

The data structures that are allowed in this setting are databases instances. We want to update our auxiliary tables with first-order formulae. To work well with words, we choose the signature consisting of the order $<$ and local letter predicates loc_a. A first-order formula over a schema $\mathcal{S}$ is a first-order formula constructed with atomic formulae that are of the form $R(\bar{x})$ for R a relation of $\mathcal{S}$ and $\bar{x}$ a tuple of variables of size the arity of R. We also add the atomic formulae for the letter predicates and the chosen numerical predicates: $x \leq y$ and $R(\bar{x})$ where $\bar{x}$ is a tuple of variables, shifted variables $x+k$ or $x-k$, or constants k or $\max-k$. We interpret these formulae over databases $\mathcal{D}$. Like in Chapter 1, for $\mathcal{V}$ a set of variables, a $\mathcal{V}$-structure of size n is a database over n with a function ν from $\mathcal{V}$ to $\{1,\dots,n\}$ that induces a partition of $\mathcal{V}$. The satisfaction relation is analogous, we only give a few cases for the example. We fixed the signature, so there is no need for an interpretation. Let $\mathcal{D}$ be a $\mathcal{V}$-structure of size n:

- $\mathcal{D} \models R(\bar{x})$ whenever $R(\bar{i})$ stands in $\mathcal{D}$ where $\bar{i} = \nu(\bar{x})$ (with ν applied component-wise),
- $\mathcal{D} \models \exists x \varphi$ whenever $\mathcal{D}_i \models \varphi$ for some i where $\mathcal{D}_i$ is $\mathcal{D}$ with $\nu(x) = i$.

Definition 7.4.
A *dynamic logic program* $\mathcal{P}$ is a tuple $(\mathcal{S}, (\varphi^a_R)_{R,a}, \varphi_M, (\varphi^{init}_R)_R)$ where:

- $\mathcal{S}$ is a relational schema that contains the alphabet schema,
- for every relation R and letter a, $\varphi^a_R(i, \bar{j})$ is a first-order formula over $\mathcal{S}$ with the arity of R plus one free variables,
- φ_M is a first-order sentence over $\mathcal{S}$,
- for every relation R, $\varphi^{init}_R(\bar{j})$ is a first-order formula over $\mathcal{S}$ with the arity of R free variables.

It will be implicit that $\mathcal{S}$ contains the alphabet schema. We now describe the action of a dynamic logic program $\mathcal{P} = (\mathcal{S}, (\varphi^a_R)_{R,a}, \varphi_M, (\varphi^{init}_R)_R)$ on a dynamic problem for $\mathcal{L}$. Initially, we have a word $w = \bot^n$ and an empty database instance $\mathcal{D}$ of $\mathcal{S}$ over n that is initialised with the formulae φ^{init}_R. To be precise, at the beginning, a relation R contains the tuple $\bar{j}$ if and only if $\mathcal{D} \models \varphi^{init}_R(\bar{j})$. When there is an operation $\mathbf{SET}_a(i)$, we update $\mathcal{D}$. After the update, a relation R contains the tuple $\bar{j}$ if and only if $\mathcal{D} \models \varphi^a_R(i, \bar{j})$. Note that in this case i and $\bar{j}$ are fixed, and thus $\varphi^a_R(i, \bar{j})$ is a sentence. On an operation **MEMBER**, the formula φ_M is applied on the current state of the database. We say that $\mathcal{P}$ is a dynamic logic program for $\mathcal{L}$ if φ_M is true on a **MEMBER** operation exactly when $w \in \mathcal{L}$.

Example 7.5.
We give an example of first-order formulae used to maintain a word. That is to say that we explicit the formula used to maintain the database instance of $\mathcal{S}_A$ that represents a word w. In this case, for a, b two letters and $1 \leq i, j \leq n$:

$$\varphi^a_{W_b}(i, j) = (i = j \wedge a = b) \vee (i \neq j \wedge W_b(j)).$$

These are very simple formulae, and we always assume that the relations W_a are maintained in this fashion. This allows to remove every letter predicate $\mathbf{a}(i)$ in a formula to replace them with the predicate $W_a(i)$. In this case, we have to assume that the table $W_\bot$ is initialised to contain all integers between 1 and n at the beginning.

We define our principal classes of dynamic complexity. We will in particular look at restrictions on the arity of the auxiliary schema.

Definition 7.6.
We define the following complexity classes:

- Dyn-FO is the class of languages for which there is a dynamic logic program,
- UDyn-FO is the class of languages for which there is a dynamic logic program with a schema with only unary relations,
- BDyn-FO is the class of languages for which there is a dynamic logic program with a schema with only unary and binary relations.

We will also look at restriction on the formulae used by dynamic logic programs. For F a fragment of first-order logic, we denote by Dyn-F the class of languages for which there is a dynamic logic program with only formulae in F. There are the analogous classes UDyn-F and BDyn-F. To fit with the classes that already exist in the literature, we denote by Prop the set of first-order formulae without quantifiers. We extend the notation of all these classes for dynamic problems for monoids.

7.2 Regular languages maintainable in RAM

The study of regular word languages maintainable in RAM, and in particular dynamic problems for monoids, has been started by Skovbjerg Frandsen, Miltersen and Skyum [122] who obtained a partial classification of monoids based on their complexity. Later, this classification has been completed by Amarilli, Jachiet and Paperman [7] (assuming Conjecture 7.3). It is also based on algebraic properties of the languages.

We call **ZG** the variety of monoids that satisfy the equation $x^{\omega+1}y = yx^{\omega+1}$ for every elements x and y. It means that the elements that are part of a group can commute with every other element. It generalises both commutative monoids and syntactic monoids of finite languages.

The second variety of monoids is **SG** and is defined by the equation $x^{\omega}yx^{\omega+1} = x^{\omega+1}yx^{\omega}$ for every elements x and y. It means that two elements that are part of a same group can commute. We can now state the classification theorem for the classification of regular languages with regards to the complexity of dynamic RAM program for their dynamic problems.

Theorem 7.7 (Amarilli, Jachiet, Paperman [7])**.**
Let $\mathcal{L}$ be a regular word language. Assuming Conjecture 7.3:

- if $\mathcal{L}$ is in **ZG**, then it has a dynamic RAM program in $O(1)$,
- if $\mathcal{L}$ is in **SG** but not in **ZG**, then it has a dynamic RAM program in $O(\log(\log(n)))$ but none in $O(1)$,
- if $\mathcal{L}$ is not in **SG**, then it has a dynamic RAM program in $O(\log(n)/\log(\log(n)))$, but none more efficient.

Their result is even more general: it stands in the case where there is no letter $\perp$ in the alphabet, and the maintained word is initialised arbitrary. In this case, the methods of Section 2.4 apply and the theorem is true with **ZG** and **SG** replaced respectively by **QLZG** and **QSG**.

We continue this study for the case of the regular tree languages, by giving a decidable characterisation of the regular tree languages that have a dynamic RAM program in $O(1)$.

7.2.1 Almost-commutative languages

First of all, to apply the algebraic method, we need to check that the complexity class under study is a variety of forest algebras.

Lemma 7.8.
The regular languages that have a dynamic RAM program in $O(1)$ form a variety of tree languages.

Proof. Let $\mathcal{L}$ and $\mathcal{L}'$ be two regular tree languages, and let $\mathcal{P}$ and $\mathcal{P}'$ be two dynamic RAM programs for them. We describe a dynamic RAM program that maintains a tree t.

- **Boolean operations.** A program for $\mathcal{L} \cap \mathcal{L}'$ initialises the memory with both the data structures from $\mathcal{P}$ and $\mathcal{P}'$, and update them both when needed. The initialisation takes a linear time, and the updates take a constant time. Finally, to answer an operation **MEMBER**, the program queries if the word is in $\mathcal{L}$ with $\mathcal{P}$, and in $\mathcal{L}'$ wiht $\mathcal{P}'$, and performs a bitwise and of the answers. The ideas for $\mathcal{L} \cup \mathcal{L}'$ and $\mathcal{L}^c$ are similar.
- **Quotient.** Let c be a context. The new program uses the algorithms of $\mathcal{P}$ on input $c \cdot t$. We use the initialisation of $\mathcal{P}$, then update the auxiliary memory by setting the values $\bot$ on top to c. Thanks to c being finite, it takes a time linear plus a constant, which is still linear. Then we use the update and membership algorithm without modifying them.
- **Inverse morphism.** Let $(\mu, \nu) : B^\Delta \to A^\Delta$ be a morphism such that $\mathcal{L}' = \mu^{-1}(\mathcal{L})$. We assume that $\mu(a)$ has the same shape for every letter a. This is done by plugging some letter $\bot$ where needed. The new program uses the algorithms of cP on input $\mu(t)$. We use the initialisation of $\mathcal{P}$: it takes a linear time as the size of $\mu(t)$ is proportional to the size of t. Then for every update, we use the update algorithm of $\mathcal{P}$ a constant number of times to update every letter in $\mu(a)$. The membership algorithm remains untouched.

□

Corollary 7.9.
Let $\mathcal{L}$ be a regular language and (H, V) be its syntactic forest algebra. Then $\mathcal{L}$ has a dynamic RAM program in $O(1)$ if and only if (H, V) has.

Proof. Assume that (H, V) has a dynamic RAM program in $O(1)$, and that $\mathcal{L}$ is recognised by the subset P of H. There is a dynamic RAM program $\mathcal{P}$ for (H, V) at x for every $x \in P$. Therefore there is a dynamic logic program for (H, V) at P, because languages recognised by RAM program in $O(1)$ are stable by union. Then $\mathcal{L}$ is maintained by $\mathcal{P}$ on the subalphabet $\mu(A)$.

Conversely, assume that $\mathcal{L}$ has a dynamic RAM program in $O(1)$. Let $x \in H$ and $(\mu, \nu) : V^\Delta \to (H, V)$ be the morphism that evaluates a tree in V^Δ. Then $\mu^{-1}(x)$ has a dynamic RAM program because it is a variety and it is recognised by the syntactic morphism of $\mathcal{L}$. This is exactly solving the dynamic problem for (H, V) at x. □

The goal is to find an equivalent of the algebraic class **ZG** that describes the class of regular languages of words that have a dynamic RAM program in $O(1)$. We first give a combinatorial description on languages, dubbed almost-commutative languages, that helps with giving an efficient dynamic program for every language inside this class. We then do an algebraic study of a counterpart of **ZG** to derive the lower bounds as well.

Let d be an integer, we say that a set $S \subseteq \mathbb{N}^d$ is *ultimately periodic* if there are two vectors $c, p \in \mathbb{N}^d$ such that

$$\forall x \in S,\ x \geq c \Rightarrow (x \in S \Leftrightarrow x + p \in S).$$

Such a set is completely described by c, p and the set T of vectors (component-wise) smaller than $c + p$ in S. Given such a description, a vector x is in S if and only if

$$(x \leq c \wedge x \in T) \vee (x > c \wedge ((x - c) \bmod p) + c \in T).$$

The *Parikh image* of a tree t over an alphabet A is the vector $v \in \mathbb{N}^A$ such that for every letter a, v_a is the number of nodes labelled by a in t. For B a subalphabet of A and a tree t over A, the

projection of t over B is the tree obtained from t by removing every letter not in B. We denote it by $\pi_B(t)$.

Definition 7.10.
A tree language $\mathcal{L}$ is *regular-commutative* if there exists an ultimately periodic set $S \subseteq \mathbb{N}^A$ such that $\mathcal{L}$ is the set of trees whose Parikh image is in S.

A tree language $\mathcal{L}$ is *virtually-singleton* if there exists a subalphabet $B \subseteq A$ and a tree t such that $\mathcal{L}$ is the set of trees whose projection over B is t.

A tree language $\mathcal{L}$ is *almost-commutative* if it is a finite Boolean combination of regular-commutative and virtually-singleton languages.

First, we easily check that every language under consideration is regular.

Fact 7.11.
Every almost-commutative language is regular.

Proof. By the closure of regular languages under Boolean operations, we only need to desribe a tree automaton for regular-commutative and virtually-singleton languages.

- Let S be an ultimately periodic set of $\mathbb{N}^A$. Let c and p given by the definition of ultimately periodic. The set of states of the tree automaton is the set of vectors smaller than $c + p$. There is a transition $a(\mathcal{M}) \to q$ for every $\mathcal{M}$ that performs the addition of its letters, removes p_a to the component a whenever a value reaches $c_a + p_a$, and checks that it is q. It is indeed regular.
- Let B be a subalphabet of A and s be a tree. The tree automaton ignores letters not in B, and remembers all the trees seen in its states, and reaches a rejecting sink state when the size is depassed.

□

We give algorithms to maintain both regular-commutative languages and virtually-singleton languages.

Lemma 7.12.
There is a dynamic RAM program for every regular-commutative language.

Proof. Let $\mathcal{L}$ be a regular commutative language with ultimately periodic set S. Let c, p and T that completely describe S, with T a finite set of vectors smaller than $c + p$. We describe the program. The initialisation stores c, p and T in memory, and instanciates a vector with only zeroes to the size of A. It takes a constant time. The new vector in memory stores the Parikh image of the maintained tree. It implies that it stores values up to size n, every value therefore fits into a single memory cell. On operation $\mathbf{SET}_a(i)$, it retreives the previous value b at position i. Then it incremenents the Parikh image for the letter a by 1, and decrements the one for b by 1. It takes constant time. When a query **MEMBER** is seen, it checks if the Parikh image is smaller than c. If it is the case, it checks if it is in T, if not it substracts c, does an operation modulo p, adds c again and checks if it is in T. Every single operation can be done in constant

time, and there are constantly many operations. □

Lemma 7.13.
There is a dynamic RAM program for every virtually-singleton language.

Proof. Let $\mathcal{L}$ be a virtually-singleton language, with subalphabet B and tree t of size k. We describe the program. We initialise several structures:

- an array M that contains the markup encoding of the tree with two set of pointers from the positions of the input word to the corresponding opening and closing positions in M,
- a doubly-linked list L that contains a couple of pointers to M for all positions with a letter in B,
- and an array T such that the i^{th} cell contains a pointer to the element of L associated to the position i if it exists.

This is done in linear time.

Assume an operation $\mathbf{SET}_a(i)$ is seen. We update M by updating to a and $\bar{a}$ the images of the two pointers. We retreive the previous value b at position i. If a and b are both in B or none are, then we do nothing. If a is in B and b is not, then we add at the beginning of L the couple of pointers that directs towards the opening and closing positions of i in M, and we add a pointer to this list element in T. If a is not in B and b is, then we use T to delete the list element stored at the adress $T[i]$. All of this takes a constant amount of time.

Assume an operation **MEMBER** is seen. We check if L has strictly more than k elements. This is done by following k pointers, hence takes a constant number of steps. If it is the case, we know that the projection of the maintained tree over B is not t. Otherwise, we sort all the pointers in L according to the markup encoding (in constant time, as k is still a constant). Finally, we use M to know the markup encoding of the projection of the tree onto B. It can be compared with the markup encoding of t to answer the query. □

Corollary 7.14.
There is a dynamic RAM program for every almost-commutative language.

Proof. This is direct with Lemma 7.12, Lemma 7.13 and the closure of languages that have a dynamic RAM program in $O(1)$ under Boolean operations. □

7.2.2 Equations

We embark on the algebraic characterisation of almost-commutative languages. We will describe them by a simple equation satisfied by their syntactic forest algebras. We generalise the class **ZG** on words to a class **ZG** on trees. This is the class of forest algebras such that the vertical monoid is in **ZG**.

Definition 7.15.

We call **ZG** the set of forest algebras (H, V) such that for all $v, w \in V$:

$$v^{\omega+1}w = wv^{\omega+1}, \tag{ZGv}$$

Unsurprisingly, every almost-commutative language satisfies this equation.

Lemma 7.16.
Let $\mathcal{L}$ be an almost-commutative language and (H, V) be its syntactic forest algebra. We have that (H, V) satisfies the equation (ZGv).

Proof. We denote by $(\mu, \nu) : A^\Delta \to (H, V)$ the syntactic morphism of $\mathcal{L}$.

First assume that $\mathcal{L}$ is a regular-commutative language. Let S be the associated subset of $\mathbb{N}^A$. Let $v, w \in V$, and c, d two contexts mapped respectively to v and w by ν. We want to show that $c \cdot d \sim_{\mathcal{L}} d \cdot c$. Let e be a context and f a tree, it amounts to show that $e \cdot c \cdot d(f) \in \mathcal{L}$ iff $e \cdot d \cdot c(f) \in \mathcal{L}$. These two trees having the same Parikh image we have the equivalence between $c \cdot d$ and $d \cdot c$. It implies that $vw = wv$. In particular, it stands that $v^{\omega+1}w = wv^{\omega+1}$.

Next, assume that $\mathcal{L}$ is a virtually-singleton language. Let B and t be the associated subalphabet and tree. Let $v \in V$ and c be a context mapped to v by ν. If c has no letter in B, then it is clear that it is equivalent to the neutral context. Thus $v = 1$ and $v^{\omega+1}w = w = wv^{\omega+1}$. If c has a letter in B, then, for n strictly greater than the size of t, the projection of c^n over B cannot be t. Thus for every context d and tree f, $d \cdot c^\omega(f)$ is not in $\mathcal{L}$. This implies that v^ω is a zero x of V. So $v^{\omega+1}w = x = w^{\omega+1}$.

To conclude, every almost-commutative language, which is a Boolean combination of the previous two types of languages, also satisfies (ZGv). □

We prove now that this equation (ZGv) is in fact rather powerful and implies a bunch of other equations. First, for any $k \in \mathbb{N}$ and $v \in V$, every element of the form $v^{\omega+k}$ can be written as an $(\omega+1)$ power, and thefore can be used in the equation (ZGv). It works in particular with idempotents. Indeed,

$$(v^{\omega+k})^{\omega+1} = v^\omega \cdot v^{\omega+k} = v^{\omega+k}.$$

The first equation that we obtain comes from the fact that the horizontal monoid of a forest algebra in **ZG** in is **ZG** as well. Indeed, by Fact 2.32, H is a submonoid of V. So for $(H, V) \in \mathbf{ZG}$ and $h, g \in H$:

$$(\omega+1) \cdot h + g = g \cdot (\omega+1) \cdot h. \tag{ZGh}$$

The study on **ZG** for words ([8, Lemma 3.8]) makes it possible to distribute the idempotent powers: for every $h, g \in H$ and $v, w \in V$ in a forest algebra in **ZG** we have:

$$(vw)^\omega = v^\omega w^\omega, \tag{DISTv}$$

$$\omega \cdot (h + g) = \omega \cdot h + \omega \cdot g. \tag{DISTh}$$

The equation (ZGv) also gives interesting interactions between the vertical and horizontal monoids.

Lemma 7.17.
Let (H, V) be a forest algebra in **ZG**. It satisfies the following equations, for every $h \in H$ and

$v \in V$:

$$v((\omega+1)\cdot h) = v(0) + (\omega+1)\cdot h, \tag{OUTh}$$
$$v^{\omega+1}(h) = v^{\omega+1}(0) + h. \tag{OUTv}$$

Proof. Let $w = 1 + h$ be an element of V. For every $n \in \mathbb{N}$, $w^n = 1 + n \cdot h$. If w^n is idempotent, then $w^{2n}(0) = w^n(0)$ and so $n \cdot h$ is idempotent as well. It implies that $w^\omega = 1 + \omega \cdot h$, and thus $w^{\omega+1} = 1 + (\omega+1)\cdot h$. We can apply (ZGv) on v and w: $vw^{\omega+1} = w^{\omega+1}v$. This rewrites into $v(1 + (\omega+1)\cdot h) = v + (\omega+1)\cdot h$. Applying the neutral forest 0 to both sides gives (OUTh).

Now, let $w = 1+h$. We apply (ZGv) to v and w: $v^{\omega+1}w = wv^{\omega+1}$. This rewrites to $v^{\omega+1}(1+h) = v^{\omega+1} + h$. Applying the neutral forest 0 to both sides gives (OUTv). □

We moreover obtain an equation that says that vertical idempotents are horizontal idempotents as well.

Lemma 7.18.
Let (H, V) be a forest algebra in **ZG**. For every $v \in V$ and $i, j \in \mathbb{N}$, we have that

$$v^{\omega+i}(0) + v^{\omega+j}(0) = v^{\omega+i+j}(0).$$

In particular,

$$v^{\omega}(0) + v^{\omega}(0) = v^{\omega}(0). \tag{IDv}$$

Proof. Let $v \in V$. We write $v^{\omega+i+j}(0) = v^{\omega+i}(v^{\omega+j}(0))$ and we apply (OUTv) (with $v^{\omega+j}(0)$ playing the role of h). This gives $v^{\omega+i}(v^{\omega+j}(0)) = v^{\omega+i}(0) + v^{\omega+j}(0)$. The "in particular" part comes from the special case $i = j = 0$. □

The last important equation is an equation that draws a brige between horizontal and vertical idempotent powers.

Lemma 7.19.
Let (H, V) be a forest algebra in **ZG**. For every $h \in H$ and $v \in V$, we have that:

$$\omega \cdot v(h) = v^{\omega}(0) + \omega \cdot h. \tag{FLAT}$$

Proof. We denote by w the element $h + 1 \in V$. It stands that $w^\omega = \omega \cdot h + 1$ and $v(h) = vw(0)$. We start with a chain of equality:

$$\begin{aligned} v^{\omega}(0) + \omega \cdot h &= v^{\omega}(\omega \cdot h) && \text{(by (OUTh))} \\ &= v^{\omega} \cdot w^{\omega}(0) \\ &= (vw)^{\omega}(0) && \text{(by (DISTv))} \end{aligned}$$

We now prove that both sides of the equation (FLAT) are equal to $v^{\omega}(0) + \omega \cdot h + \omega \cdot v(h)$.

On one hand,

$$\begin{aligned} v^\omega(0) + \omega \cdot h &= (vw)^\omega(0) \\ &= (vw)^{\omega-1}(v(h)) \\ &= (vw)^{\omega-1}(0) + v(h). && \text{(by (OUTv))} \end{aligned}$$

Repeating the process, for all $k \in \mathbb{N}$, $v^\omega(0) + \omega \cdot h = (vw)^{\omega-k}(0) + k \cdot v(h)$. Let n be a multiple of the idempotent powers of both H and V. With this value, $n \cdot v(h) = \omega \cdot v(h)$ and $(vw)^{\omega-n} = (vw)^\omega$. Hence $v^\omega(0) + \omega \cdot h = (vw)^\omega(0) + \omega \cdot v(h) = v^\omega(0) + \omega \cdot h + \omega \cdot v(h)$.

On the other hand,

$$\begin{aligned} \omega \cdot v(h) &= vw(0) + (\omega - 1) \cdot v(h) \\ &= vw((\omega - 1) \cdot v(h)). && \text{(by (OUTh))} \end{aligned}$$

Repeating the process, for all $k \in \mathbb{N}$, $\omega \cdot v(h) = (vw)^k((\omega - k) \cdot v(h))$. As previously, with k set to a multiple of the idempotent powers of H and V, we have that

$$\begin{aligned} \omega \cdot v(h) &= (vw)^\omega(\omega \cdot v(h)) \\ &= (vw)^\omega(0) + \omega \cdot v(h) && \text{(by (OUTh))} \\ &= v^\omega(0) + \omega \cdot h + \omega \cdot v(h). \end{aligned}$$

This concludes the proof. □

We now need a normal form on trees that are mapped on an idempotent through a morphism into a forest algebra in **ZG**.

Lemma 7.20.
Let $(\mu, \nu) : A^\Delta \to (H, V)$ with $(H, V) \in \mathbf{ZG}$ and n be the idempotent power of V. Let f be a forest mapped to an idempotent of H. For $a_1, \ldots, a_k$ an enumeration of the letters in f, we define the forest

$$C_f = a_1^n + \cdots + a_k^n.$$

It stands that:

$$\mu(f) = \mu(C_f).$$

Proof. We proceed by induction on f: we prove that for every forest f, $\omega \cdot \mu(f) = \mu(C_f)$. It will conclude as $\mu(f) = \omega \cdot \mu(f)$.

If f is empty then $f = C_f$ and thus $\omega \cdot \mu(f) = \mu(f) = \mu(C_f)$.

If $f = f_1 + f_2$, then let $b_1, \ldots, b_{k_1}$ and $c_1, \ldots, c_{k_2}$ be the letters in f_1 and f_2. We have:

$$\begin{aligned} \omega \cdot \mu(f) &= \omega \cdot \mu(f_1) + \omega \cdot \mu(f_2) && \text{(by (DISTh))} \\ &= \mu(C_{f_1}) + \mu(C_{f_2}) && \text{(by induction hypothesis)} \\ &= \mu(b_1^n) + \cdots + \mu(b_{k_1}^n) + \mu(c_1^n) + \cdots + \mu(c_{k_2}^n). \end{aligned}$$

Each of the term in the sum is a vertical idempotent, so by Lemma 7.18 also an horizontal idempotent, hence we can apply (DISTh) to commute them. Hence we can put them in any

order and use idempotency to obtain only one copy of each letter. Finally, we have to see that the set of letters in f is the union of the letters in f_1 and in f_2.

If $f = a(g)$, then let $b_1, \cdots, b_k$ be the letters in g. We have:

$$\begin{aligned}
\omega \cdot \mu(f) &= \omega \cdot \nu(a(\Box))(\mu(f)) \\
&= \nu(a(\Box))^\omega(0) + \omega \cdot \mu(f) && \text{(by (FLAT))} \\
&= \mu(a^n) + \mu(C_f) && \text{(by induction hypothesis)} \\
&= \mu(a^n) + \mu(b_1^n) + \cdots + \mu(b_k^n)
\end{aligned}$$

We conclude exactly like in the previous case. □

Let $(\mu, \nu) : A^\Delta \to (H, V)$ be a morphism and f be a forest. We say that a non-empty context c is an *idempotent factor* of f if there exists a context d and a forest g such that:

$$f = d(c(g)) \text{ and } \nu(c) \text{ is an idempotent.}$$

We need the ability to find idempotent factors in forests.

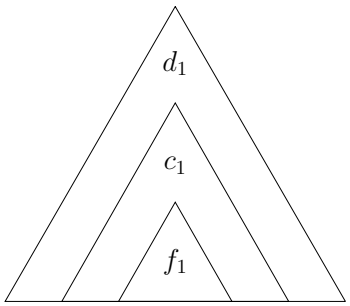
Fact 7.21.
Let $(\mu, \nu) : A^\Delta \to (H, V)$ be a morphism and f be a forest. If f is of size greater than $|V|^{5|V|^{6|V|}}$, then it is possible to find an idempotent factor in f.

Proof. We use a well known fact that for a monoid M and a word $w \in M^n$ with $n \geq |M|^{5|M|}$, there is a subword of w that is mapped to an idempotent. See [64, Theorem 1] for fine bounds on n. There are three cases.

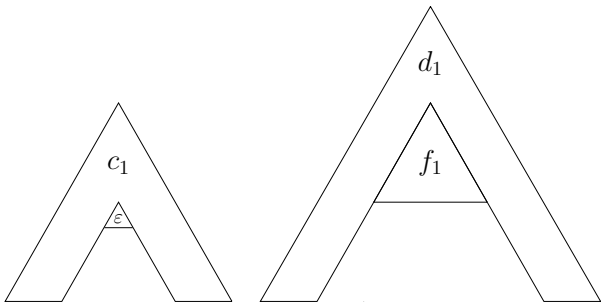
- There is a node with more than $|V|^{5|V|}$ children (or there are that many roots). In this case, we denote by $g_1, \ldots, g_m$ the forest rooted at every child, and by c_i the context $g_i + \Box$. We construct a word $w \in V^m$ by setting $w_i = \nu(c_i)$. Because m is big enough, this word contains an idempotent $\nu(c_i \cdots c_j)$. Let d be the context that consists of f with $g_i + \cdots + g_j$ identified in a single node $\Box$, and $c = c_i(\cdots(c_j))$. Thus $f = d(c(0))$ and $\nu(c)$ is an idempotent.
- There is a path of length greater than $|V|^{5|V|}$, that we assume being from the root to a leaf. In this case, let $a_1, \ldots, a_m$ be the nodes along the path. For $1 \leq i < m$, let l_i (resp. r_i) be the forest with the left (resp. right) siblings of a_{i+1}. We define $c_i = a_i(l_i + \Box + r_i)$. For $i = 1$, we also add the other roots, and $c_m = a_m(\Box)$. With these definitions, we have that $f = c_1(c_2(\cdots(c_m(0))))$. We construct a word $w \in V^m$ by setting $w_i = \nu(c_i)$. Because m is big enough, we can find an idempotent $\nu(c_i \cdots c_j)$. Let $d = c_1(\cdots(c_{i-1}))$, $c = c_i(\cdots(c_j))$ and $g = c_{j+1}(\cdots(c_m(0)))$. We have $f = d(c(g))$ and $\nu(c)$ is an idempotent.
- Every node has less than $|V|^{5|V|}$ children and every path is of length less than $|V|^{5|V|}$. In this case, f has size less than $|V|^{5|V||V|^{5|V|}}$. This is a contradiction with the assumption of the size of f.

□

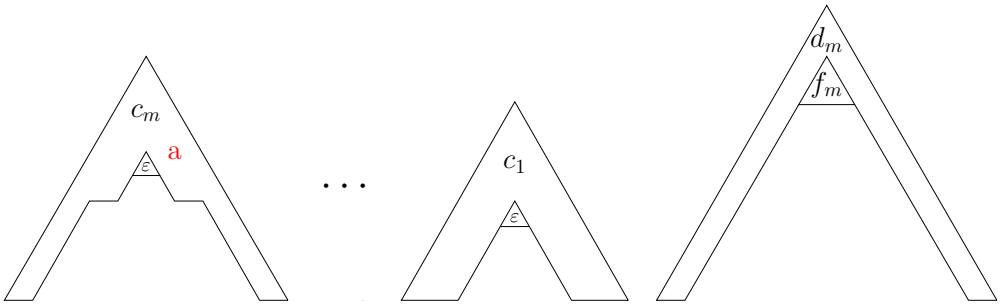
We can deduce from that a normal form for every tree.



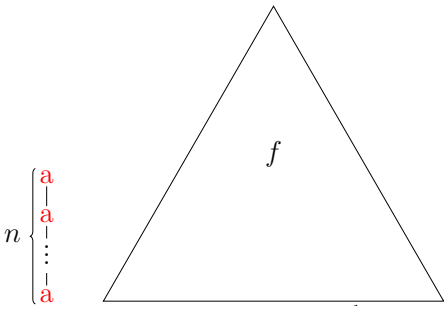
(a) Finding of an idempotent.



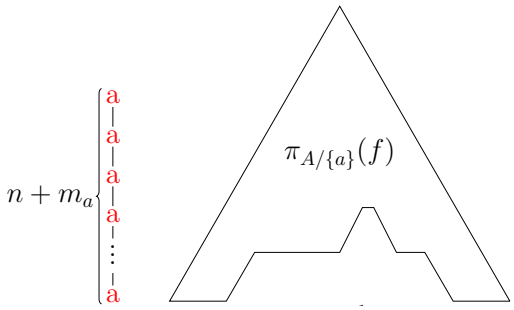
(b) Extraction with (OUTh) and (OUTv).



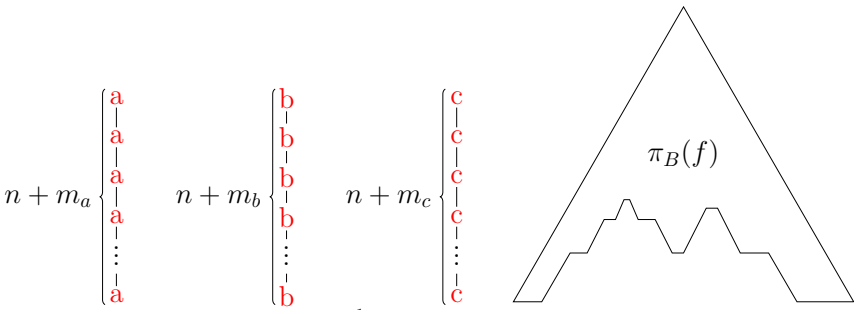
(c) Repetition until an idempotent with a letter in C is found.



(d) Extraction of many as with Lemma 7.20, and reconstruction of f.



(e) Extraction of every a from f with (OUTh) and (OUTv).



(f) Extraction of every letter in C from f.

Figure 7.1: Proof of Lemma 7.22

Lemma 7.22.
Let $(\mu, \nu) : A^\Delta \to (H, V)$ with $(H, V) \in \mathbf{ZG}$ and n be the idempotent power of V. Let $N = |V|^{5|V|^{6|V|}}$. Let f be a forest. Let $(m_1, \dots, m_k)$ be the Parikh image of t for an enumeration $A = \{a_1, \dots, a_k\}$, and $r_i = m_i \bmod n$. We partition $A = B \cup C$ with

$$B = \{a_i \mid m_i < N\},$$
$$C = \{a_i \mid m_i \geq N\}.$$

We define the forest:

$$K_f = \sum_{a_i \in C} a_i^{n+r_i} + \pi_B(f).$$

It stands that:

$$\mu(f) = \mu(K_f).$$

Proof. The proof is graphically represented in Fig. 7.1. We proceed by induction on the number of letters in C. If $C = \varnothing$, then $K_f = f$ and they have the same images under μ. Now assume that C has at least one letter.

First, we want to find an idempotent factor in f that contains a letter in C. Let $f_0 = f$. Assume we have constructed a sequence of forests $f_0, \dots, f_i$ and a sequence of contexts $c_1, \dots, c_i$, such that every f_j has size greater than N. Thanks to that, we can apply Fact 7.21 to f_i. This give the decomposition in Fig. 7.1a. Hence we can write $f_i = d_{i+1}(c_{i+1}(g_{i+1}))$ with $\nu(c_{i+1})$ an idempotent of V. Let $f_{i+1} = d_{i+1}(g_{i+1})$. We claim that $\mu(f_i) = \mu(c_{i+1}(0) + f_{i+1})$, as represented in Fig. 7.1b. Indeed, with $v = \nu(d_{i+1})$, $w = \nu(c_{i+1})$ an idempotent and $h = \mu(g_{i+1})$:

$$\begin{aligned}
\mu(f_i) &= \mu(d_{i+1}(c_{i+1}(g_{i+1}))) \\
&= v \cdot w^\omega \cdot h \\
&= v \cdot (w^\omega(0) + h) && \text{(by (OUTv))} \\
&= v \cdot (\omega \cdot w^\omega(0) + h) && \text{((by IDv))} \\
&= v(h) + w^\omega(0) && \text{(by (OUTh))} \\
&= w^\omega(0) + v(h) && \text{(by (ZGh))} \\
&= \mu(c_{i+1}(0) + f_{i+1})
\end{aligned}$$

If c_{i+1} contains a letter in C, we stop the construction. Otherwise, there exists a letter b in C such that f_{i+1} and f have the same number of occurences of b. This implies that f_{i+1} has size greater than N, and we can repeat the process. The size of f_i decreases at each step, so the constuction must terminate.

Let $f_0, \dots, f_m$ and $c_1, \dots, c_m$ be the obtained sequences, and a be the letter in C that appears in c_m. We have that, as represented in Fig. 7.1c:

$$\mu(f) = \mu(c_m(0)) + \cdots + \mu(c_1(0)) + \mu(f_m).$$

Consider the forest $g = a^n(0) + c_m(0)$. The contexts $\nu(a^n(\Box))$ and $\nu(c_m)$ are vertical idempotents, and therefore, by (IDv), both $\mu(a^n(0))$ and $\mu(c_m(0))$ are horizontal idempotents. So by (DISTh), $\mu(g)$ is also an idempotent. Let C_g and $C_{c_m(0)}$ be defined as in the statement of Lemma 7.20.

However, g and $c_m(0)$ have the same letters, and therefore $C_g = C_{c_m(0)}$. So by Lemma 7.20:

$$\mu(g) = \mu(C_g) = \mu(C_{c_m(0)}) = \mu(c_m(0)).$$

This gives, as represented in Fig. 7.1d:

$$\begin{aligned}\mu(f) &= \mu(a^n(0)) + \mu(c_m(0)) + \cdots + \mu(c_1(0)) + \mu(f_m)\\ &= \mu(a^n(0)) + \mu(f).\end{aligned}$$

Secondly, we build a sequence of forests $f_0, \ldots, f_m$ and a sequence of integers $n_0, \ldots, n_m$ such that

- $f_0 = f$ and $n_0 = 0$,
- for every $0 \leq i \leq m$, $\mu(a^{n+n_i} + f_i) = \mu(f)$,
- the number of as in f is the number of as in f_i plus n_i,
- the number of as in f_i is stricly decreasing, and f_m has no a.

We show how to construct f_{i+1} and n_{i+1} from f_i and n_i. Assume that there still is an a in f_i. In this case, we can write $f_i = c_i(a(g_i))$. Let $f_{i+1} = c_i(g_i)$. Let $v = \nu(a(\Box))$, $w_i = \nu(c_i)$ and $h_i = \mu(g_i)$. The value $v^{\omega+n_i}(0)$ can be written, thanks to Lemma 7.18, as $(\omega+1)\cdot v^{\omega+n_i}(0)$. So we can apply (OUTh) with $v^{\omega+n_i}(0)$. We have that:

$$\begin{aligned}\mu(f) &= \mu(a^{n+n_i} + f_i)\\ &= v^{\omega+n_i}(0) + w_i v h_i\\ &= w_i(v^{\omega+n_i}(0) + v h_i) && \text{(by (OUTh))}\\ &= w_i(v^{\omega+n_i}(v h_i)) && \text{(by (OUTv))}\\ &= w_i(v^{\omega+n_i+1}(h_i))\\ &= w_i(v^{\omega+n_i+1}(0) + h_i) && \text{(by (OUTv))}\\ &= v^{\omega+n_i+1}(0) + w_i(h_i) && \text{(by (OUTh))}\\ &= \mu(a^{n+n_i+1} + f_{i+1})\end{aligned}$$

We set $n_{i+1} = n_i + 1$. We have indeed that f_{i+1} has strictly less as than f_i. In the end, thanks to the conservation of the number of as, we have that

$$\begin{aligned}\mu(f) &= \mu(a^{n+m_a} + f_m)\\ &= \mu(a^{n+r_a} + f_m)\end{aligned}$$

with f_m being the projection of f on $A/\{a\}$ and m_a is the number of as in f. The situation is represented in Fig. 7.1e. The last equality comes from the fact that $\mu(a^n) = \mu(a^{2n})$, thus we can substract n to m_a until we reach $r_a = m_a \bmod n$. To conclude, we use the induction hypothesis on f_m, that has one less letter in C than f. We obtain the figure in Fig. 7.1f. □

We have everything we need to conclude.

Theorem 7.23.
A language $\mathcal{L}$ is almost-commutative if and only if its syntactic forest algebra is in **ZG**.

Proof. By Lemma 7.16, we have the left-to-right implication. We prove the other one: let $\mathcal{L}$ be a language recognised by a morphism $(\mu, \nu) : A^\Delta \to (H, V)$, with (H, V) in **ZG**. Let $N = |V|^{5|V|^{6|V|}}$ and n be the idempotent power of V. Remember the definition of K_f from the statement Lemma 7.22. We call the set B the set of rare letters in f and C the set of frequent letters in f, and write them B_f and C_f We define an equivalence relation on the set of forests as:

$$f \sim g \text{ iff } \begin{cases} B_f = B_g \text{ and } C_f = C_g \\ \pi_{B_f}(f) = \pi_{B_g}(g) \\ \forall a \in C_f, |f|_a \equiv |g|_a \bmod n \end{cases}.$$

By Lemma 7.22, if $f \sim g$ then $K_f = K_g$ and therefore $\mu(f) = \mu(g)$. Moreover, there are finitely many equivalence classes of $\sim$. Indeed, trees of the form $\pi_{B_f}(f)$ have at most $|A| \cdot N$ letters. From these two facts we deduce that $\mathcal{L}$ is a finite union of equivalence classes of $\sim$. All is left to do is to show that an equivalence class X of $\sim$ is an almost-commutative language. Let B and C such that $A = B \cup C$, and h be a forest over B, and $m \in \{0, \dots, n-1\}^C$ such that X is the set of forests whose rare letters are B, frequent letters are C, the projection over B is h and for every $a \in C$, there is a number of as congruent to m_a modulo n. We define $\mathcal{L}_1$ to be the virtually-singleton language of forests whose projection over B is h. Let S be the ultimately-periodic set such that S consists of vectors x such that

- $x_a \geq N$ and $x_a \equiv m_a$ modulo n for every $a \in C$,
- $x_a = |h|_a$ for every $a \in B$.

We define $\mathcal{L}_2$ to be the regular-commutative language of forest whose Parikh image is in S. We have that

$$X = \mathcal{L}_1 \cap \mathcal{L}_2,$$

proving that it is almost-commutative. □

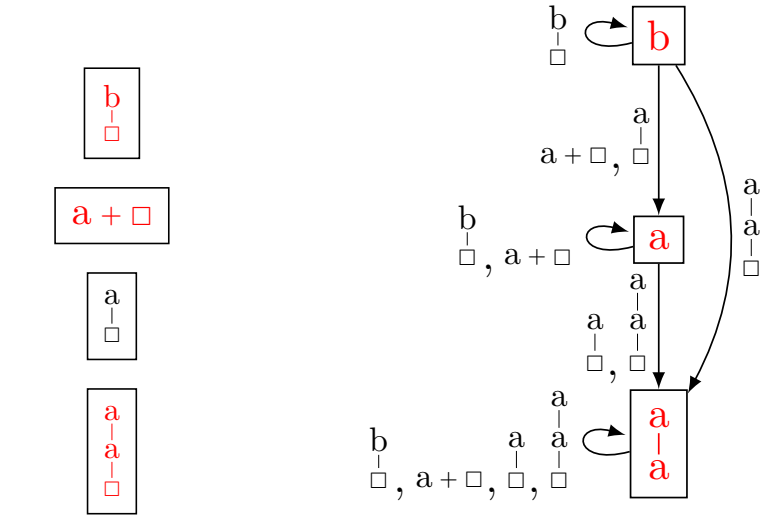
7.2.3 Regular languages maintainable in constant time

To conclude, we still need a lower bound. Fortunately for us, the equation that characterises almost-commutative languages gives us that we can use as a black-box the lower bound obtained in [7].

Theorem 7.24.
Let $\mathcal{L}$ be a regular tree language. Assuming Conjecture 7.3,

$$\mathcal{L} \text{ has a dynamic RAM program in} O(1) \text{ iff } \mathcal{L} \text{ is almost-commutative}$$
$$\text{iff } \mathcal{L} \text{ is in } \mathbf{ZG}.$$

Proof. The second equivalence comes from Theorem 7.23. The first reverse implication comes from Corollary 7.14. To show the last implication, we show that if a language $\mathcal{L}$ is not in **ZG**, then it has no dynamic RAM program in $O(1)$. Let (H, V) be the syntactic forest algebra of $\mathcal{L}$, we know that V is not in **ZG**. Assume that there is a dynamic program in $O(1)$ for (H, V) at every $h \in H$. Then there is a dynamic program in $O(1)$ for V at every $v \in V$. Let w be the maintained word. Indeed, we run in parallel a dynamic RAM program for every $h \in H$. The maintained tree is without branching of depth the size of w plus a leave of constant value $h + 1$. When a letter of the word is changed, we change the corresponding node of the trees. To



(a) Vertical monoid.

(b) Horizontal monoid.

Figure 7.2: Syntactic forest algebra for the antichain language.

answer a query, we check that, for each $h \in H$, the tree maintained for h evaluates to $v(h)$. By faithfulness, we can conclude whether w evaluates to v or not.

So we have a dynamic RAM program in $O(1)$ for V. By Theorem 7.7, this is a contradiction. So there is no dynamic RAM program in $O(1)$ for (H, V). By Corollary 7.9, neither does $\mathcal{L}$. □

We illustrate this theorem on an example.

Example 7.25.
Let $A = \{a, b\}$ and let $\mathcal{L}$ be the regular language of trees whose as form an antichain. In other words, there is no node labelled by a with a descendant also labelled by an a. We compute its syntactic forest algebra (H, V). The corresponding eggbox pictures are depicted in Fig. 7.2, where the arrows between the elements of the horizontal monoid represent the action. We can check that $a + \square$ is an idempotent in V, and that $(a + \square)(a(\square)) \neq (a(\square))(a + \square)$. This can be seen with their respective actions on 0. So V is not in **ZG**, and neither is (H, V). This implies that this antichain language has no dynamic RAM program in $O(1)$.

7.2.4 Going further

The next natural step is to try to obtain a trichotomy similar to Theorem 7.7. For that, we have to find the counterpart of **SG** for forest algebras. The task is very involved and a characterisation of the class of languages with a dynamic RAM program in $O(\log(\log(n)))$ is currently not known. To give a flavour of the difficulty of the problem, we consider the *marked ancestor problem*. It is the language of tree over $\{a, b, c\}$ that have a single b and such that there is an a in the path from the b to the root. It has been studied by Alstrup, Husfeldt and Rauhe [5] who have shown a lower bound in $\Omega(\log(n)/\log(\log(n)))$ for this problem in the cell-probe model, and therefore in the RAM model as well. However, there are slight modifications of this problem that have

more efficient algorithms and can be maintained in $O(\log(\log(n)))$. For instance, the language over $\{a,b\}$ with an a that is an ancestor of another a can be maintained with the latter complexity. To see that, we maintain the markup encoding of the maintained tree, and we use the techniques of [7] with van Emde Boas tree to maintain auxiliary data. The marked ancestor problem seems to be crucial in the understanding of an equivalent of the class **SG** for tree languages.

Another task is to change the model so that there is no fresh symbol $\bot$. Is this case, the tree is initilialised to any value with the alphabet A. This amounts to study regular tree languages that possibly have no neutral letter. To study them, we would need a theory similar to the one developed in Section 2.4 for trees, with equivalent to the notion of locality, with delay theorems and derived category theorems for trees.

7.3 Regular languages in Dyn-FO

We study the regular languages in the complexity class Dyn-FO, and we are in particular looking for precise complexity measures on the formulae that are used. Before starting, it is useful to notice that all the classes under consideration are varieties of languages.

Lemma 7.26.
The regular languages in Dyn-FO, UDyn-Prop and BDyn-Prop form varieties of languages.

Proof. We prove it for Dyn-FO only. It is direct that the constructions preserve the arities of relations and the absence of quantifiers.

- **Boolean operations.** Let $\mathcal{L}$ and $\mathcal{L}'$ be two languages in Dyn-FO maintained respectively by $\mathcal{P} = (\mathcal{S}, (\varphi^a_R)_{R,a}, \varphi_M, (\varphi^{init}_R)_R)$ and $\mathcal{P}' = (\mathcal{T}, (\psi^a_R)_{R,a}, \psi_M, (\psi^{init}_R)_R)$. Then $\mathcal{L} \cup \mathcal{L}'$ has a dynamic logic program $(\mathcal{S} \cup \mathcal{T}, (\varphi^a_R)_{R,a} \cup (\psi^a_R)_{R,a}, \varphi_M \vee \psi_M, (\varphi^{init}_R)_R \cup (\psi^{init}_R)_R)$. Then $\mathcal{L} \cup \mathcal{L}'$ has a dynamic logic program $(\mathcal{S} \cup \mathcal{T}, (\varphi^a_R)_{R,a} \cup (\psi^a_R)_{R,a}, \varphi_M \wedge \psi_M, (\varphi^{init}_R)_R \cup (\psi^{init}_R)_R)$. Then $\mathcal{L}^{\mathsf{c}}$ has a dynamic logic program $(\mathcal{S}, (\varphi^a_R)_{R,a}, \neg\varphi_M, (\varphi^{init}_R)_R)$.
- **Inverse morphisms.** Let $\mu : A^* \to B^*$ be a morphism and $\mathcal{L}$ a regular language over B in Dyn-FO. Let $\mathcal{P} = (\mathcal{S}, (\varphi^a_R)_{R,a}, \varphi_M, (\varphi^{init}_R)_R)$ be a dynamic logic program for $\mathcal{L}$. We want to show that $\mathcal{L}' = \mu^{-1}(\mathcal{L})$ is in Dyn-FO. Let w be the word under changes, of size n.
  Let S be a bound on the size of words in $\mu(A)$. For a letter a and $1 \leq s \leq S$, we use $\mu(a)_s$ to denote the s^{th} letter of $\mu(a)$. If $s \geq |\mu(a)|$, then $\mu(a)_s = \bot$.
  We define $\mathcal{S}'$ to be the schema that consists in copies of $\mathcal{S}$: for every relation R of arity k, there is a relation $R_{\bar{s}}$ in $\mathcal{S}'$ for $\bar{s}$ a tuple in $\{1,\dots,S\}^k$. In particular, there are S copies of the alphabet schema. We will use these relations to maintain the image $\mu(w)$. The formulae are, for a,b two letters and $1 \leq s \leq S$:

$$\psi^a_{W_{b,s}} = (i = j \wedge b = \mu(a)_s) \vee (i \neq j \wedge W_{a,s}(j)).$$

  We consider that the relations $W_{a,s}$ are storing a word w' of size $n \cdot S$. Let q and r be the functions that take a natural number and return its quotient and rest in the Euclidean division by S. At all time, $w'_k = a$ where a is the only letter such that $q(k)$ is in $W_{a,r(k)}$.
  We want to store in the relation $R_{\bar{s}}$ all the tuples $\bar{j}$ such that the tuple $(\bar{j_k} \cdot S + \bar{s}_k)_{1 \leq k \leq S}$ would be in the relation R after applying $\mathcal{P}$ to get w'. We will build formulae $\psi^{a,s'}_{R,\bar{s}}(i,\bar{j})$ that will update all the auxilliary relations when $w'_{i \cdot S + s'}$ is set to a. For R a relation, we

write φ_R^a in prenex normal form: there is first a block of quantifiers over variables $\bar{x}$, then a quantifier-free formula $\varphi(i, \bar{j}, \bar{x})$. We furthermore write φ in disjunctive normal form, and separate the atomic formulae between the numerical and relation ones. That is to say that:

$$\varphi = \bigvee C_{num} \wedge C_{rel},$$

where C_{num} (resp. C_{rel}) is a conjunction of numerical (resp. relation) predicates and their negations. We can assume that a clause C_{num} always contains exactly once every numerical predicate involving the variables $\bar{y} = (i, \bar{j}, \bar{x})$. Then every clause C_{num} enforces an order on the variables of $\bar{y}$ of the form:

$$Y_1 <_{t_1} Y_2 <_{t_2} \cdots <_{t_{p-1}} Y_p,$$

where the Y_i are a partition of $\bar{y}$ that symbolises the variables that have to be equals, and the $\leq_{t_i}$ with $t_i \in \mathbb{N} \cup \{\infty\}$ symbolise that the variables of Y_i have to be at distance t_i from the variables of Y_{i+1}. If $t_i = \infty$, it just means that they are ordered. This is called a profile. Then we set $\psi_{R,\bar{s}}^{a,s'}(i, \bar{j})$ to be the formula φ_R^a where every clause $R(\bar{j'})$ in C_{rel} which is associated to a profile as above is replaced by:

$$\bigvee_{\bar{t} \in X} R_{\bar{t}}(\bar{j'}),$$

where X is the set of tuples $\bar{t}$ such that:

- for all positions k such that $\bar{j'}_k = i$, we have $\bar{t}_k = s'$,
- for all positions k such that $\bar{j'}_k = \bar{j}_{k'}$ for some k', we have $\bar{t}_k = \bar{s}_{k'}$,
- for all positions k_1, k_2 such that $\bar{j'}_{k_1}$ and $\bar{j'}_{k_2}$ are in the same Y_k, then $\bar{t}_{k_1} = \bar{t}_{k_2}$,
- for all positions k_1, k_2 such that $\bar{j'}_{k_1}$ and $\bar{j'}_{k_2}$ are in two consecutive Y_k separated by $\leq_d$, then $\bar{t}_{k_1} + d = \bar{t}_{k_2}$,
- for all positions k_1, k_2 such that $\bar{j'}_{k_1}$ and $\bar{j'}_{k_2}$ are in two consecutive Y_k separated by $\leq_\infty$, then either we do not enforce other conditions, or $\bar{t}_{k_1} < \bar{t}_{k_2}$ and $(\bar{j'})_{k_2}$ is set to $(\bar{j'})_{k_1}$ (or the other way around),

This new formula emulates the behaviour of φ_R^a on the new way of storing the word w'. Now, when an operation $\mathbf{SET}_a(i)$ is seen, we have to simulate the application of $\mathbf{SET}_{\mu(a)_1}(i \cdot S), \ldots, \mathbf{SET}_{\mu(a)_S}(i \cdot S + S - 1)$ in the dynamic problem for w'. This is done by using in turn the formulae $\psi_{R'}^{\mu(a)_1,0}$ for every $R' \in \mathcal{S}'$, then every $\psi_{R'}^{\mu(a)_2,1}, \ldots$ To do that we replace every atomic formula $R_s(\bar{j})$ in $\psi_{R'}^{\mu(a)_1,0}$ by $\psi_R^{\mu(a)_2,1}$, and repeat the process.
In the end, we apply the same treatment to φ_M and the initialisation formulae to obtain a dynamic logic program that maintains $\mathcal{L}' = \mu^{-1}(\mathcal{L})$.

- **Quotients.** Let $\mathcal{L}$ be a language in Dyn-FO maintained by $\mathcal{P} = (\mathcal{S}, (\varphi_R^a)_{R,a}, \varphi_M, (\varphi_R^{init})_R)$. Let a be a letter. We maintain $a^{-1}\mathcal{L}$ with the same trick as before: we maintain in $\mathcal{L}$ a word w' that is encoded with extra relations V_a that are never updated and such that V_a only contains 1 and $V_\perp$ contains all the other integers. We initialise the relations with the initialisation formulae of $\mathcal{L}$, to take into account the presence of a from the start.

□

This implies that we can work with dynamic problems on monoids directly.

Corollary 7.27.
Let $\mathcal{L}$ be a regular language and M be its syntactic monoid. Then:

$$\mathcal{L} \in \text{Dyn-FO} \Leftrightarrow M \in \text{Dyn-FO}.$$

The statement is also true for UDyn-Prop and BDyn-Prop.

Proof. Assume that M is in Dyn-FO, and that $\mathcal{L}$ is recognised by the subset P. There is a dynamic logic program $\mathcal{P}$ for M at x for every $x \in P$. Therefore there is a dynamic logic program for M at P, because Dyn-FO is stable by union. Then $\mathcal{L}$ is maintained by $\mathcal{P}$ on the subalphabet $\mu(A)$.

Conversely, assume that $\mathcal{L}$ is in Dyn-FO. Let $x \in M$ and $\mu : M^* \to M$ be the morphism that evaluates a word in M^*. Then $\mu^{-1}(x)$ is in Dyn-FO because it is a variety and it is recognised by the syntactic morphism of $\mathcal{L}$. This is exactly solving the dynamic problem for M at x. □

First of all, if we allow binary relations in the auxiliary schema, we can maintain every regular language very efficiently in Dyn-FO.

Lemma 7.28.
Every regular language is in BDyn-Prop.

Proof. Let M be a monoid and $t \in M$. We want to exhibit a dynamic logic program for the dynamic problem for M at t. To do that, we maintain the evaluation of every infix thanks to binary auxiliary relations.

The schema is $\mathcal{S} = \{R_x \mid x \in M\}$, where every relation is binary. The query formula is simply $\varphi_M = R_t(1, \max)$. The initialisation formulae put every couple in R_1 and let the other relations empty. The update formulae are:

$$\varphi^a_{R_x}(i,j,k) = \quad (j < i < k) \wedge \left(\bigvee_{\substack{y,z \in M \\ yaz = x}} R_y(j, i-1) \wedge R_z(i+1, k) \right)$$
$$\vee \quad (i < j \vee k < i) \wedge R_x(j,k).$$

To be completely precise, we would also have to treat the cases $j = i < k$, $j < i = k$ and $j = i = k$. Note that we do not care about what is stocked in $R_x(j,k)$ for $j > k$. With these, after every sequence of **SET** operations, we have that R_x contains the set of couples (j,k) such that $w_j \cdots w_k$ evaluates to x. By Corollary 7.27, every regular language is thus in Dyn-FO. □

This justifies the study of Dyn-FO with only unary auxiliary relations. For instance, we will see that there are regular languages that are not in UDyn-Prop. The dynamic complexity of regular languages have also been studied by Tschirbs, Vortmeier and Zeume [132]. They give efficient algorithms to maintain regular languages in Dyn-FO when several, up to polylogarithmically many, changes are made at the same time. They also study the *work* of such programs, that is to say the number of steps needed to update the auxiliary relations in the RAM model.

7.3.1 Regular languages in UDyn-Prop

We start the study of regular languages in UDyn-FO by looking at the subfragment that cannot use any quantifier. First of all, we remark that when a monoid is a group, maintaining all the prefixes is enough to have the information about the infixes as well. This motivates the following lemma.

Lemma 7.29.
Every group is in UDyn-Prop.

Proof. Let G be a group and $t \in G$. We will maintain all the prefixes with unary relations. The schema is $\mathcal{S} = \{R_g \mid g \in G\}$, where every relation is unary. The query formula is simply $\varphi_M = R_t(\max)$. The initialisation formulae put every integer in R_1 and let the other relations empty. The update formulae are:

$$\varphi^a_{R_g}(i,j) = \quad (i<j) \wedge \left(\bigvee_{\substack{x,y,z\in G \\ xay^{-1}z=g}} R_x(i-1) \wedge R_y(i) \wedge R_z(j) \right)$$
$$\vee \quad (j<i) \wedge R_g(j).$$

To be completely precise, we woul also have to treat the case $i = j$. With these, after every sequence of **SET** operations, we have that R_g contains the set of integers j such that $w_1 \cdots w_k$ evaluates to g. □

We will prove a converse to this lemma, that is to say that we will show that a monoid that is not a group cannot be maintained in UDyn-Prop. We need a small result from algebra first. Recall that U_1 is the syntactic monoid $(a+b)^*a(a+b)*$ and has only two elements: one is neutral and the other a zero. We will prove that it is, in some sense, the smallest aperiodic monoid.

Fact 7.30.
Let M be a monoid that is not a group. Then U_1 divides M.

Proof. If M is not a group, it means that there is at least one $\mathcal{H}$-class which does not contain the neutral element. Let x ba an element of this class and $e = x^\omega$ be its idempotent. We know that e is not is the same $\mathcal{J}$-class as the identity. Then $\{1, e\}$ is a submonoid of M. It is isomorphic to U_1, with e playing the role of the zero. □

The only ingredient left is a lower bound against UDyn-Prop. Luckily for us, they already exist in [139]. It is based on the substructure lemma [43]: two isomorphic substructures remain isomorphic when similar changes are applied to them.

Lemma 7.31 (based on Schwentick and Zeume [139])**.**
The language $(a+b)^*a(a+b)*$ is not in UDyn-Prop.

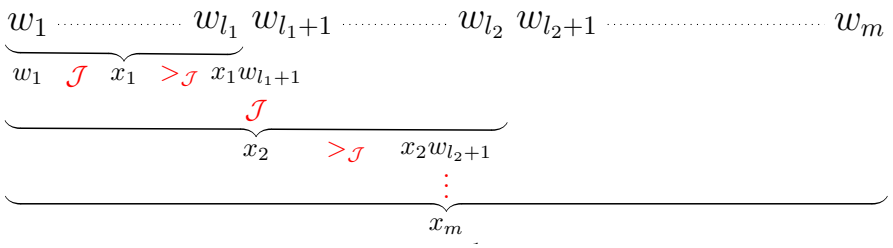


Figure 7.3: Representation of Lemma 7.33

Proof. We derive the result from Proposition 4.8 in [139]. In their framework, they can maintain not just words but also graphs and are studying maintenance of the reachability relation in graphs. A 1-layered s-t graph is a (directed) graph of the form:

- vertices are $s, t \cup \{1, \dots, n\}$ for some integer n,
- there are all the edges from $\{1, \dots, n\}$ to t, and some edges from s to $\{1, \dots, n\}$.

The precise result in [139] is that the existence of a path from s to t cannot be maintained in UDyn-Prop. It is straighforward to see that it is in fact only checking whether there is an edge from s to $\{1, \dots, n\}$, and therefore the latter problem reduces to the regular language $(a+b)^* a (a+b)*$. □

We are ready to put everything together.

Theorem 7.32.
The regular languages of UDyn-Prop are precisely the group languages. In symbols,

$$\text{UDyn-Prop} \cap \text{Reg} = \mathbf{G}.$$

Proof. With Lemma 7.29, we only have to take a regular language $\mathcal{L}$ which is not a group language, and show that it is not in UDyn-Prop. We proceed by contradiction. Let M be the syntactic monoid of $\mathcal{L}$. By Corollary 7.27, $M \in$ UDyn-Prop. It is not a group, so by Fact 7.30 U_1 divides M. By Lemma 7.26, U_1 is in UDyn-Prop. With Corollary 7.27 again, it implies that $(a+b)^* a (a+b)*$ is in UDyn-Prop, contradicting Lemma 7.31. □

7.3.2 Regular languages in UDyn-Σ_2

We tackle the question of the least quantifier alternation needed to maintain all regular languages. It turns out that we only need one alternation. Our proof finds its inspiration in Section 2.3.2 of William Hesse's PhD thesis [61]. However, his proof works by studying the combinatorics of the graph of transitions during the execution of an automaton, and is rather involved. We use the framework of algebra here to give a simpler and more intuitive proof. It also reduces the quantifier alternations of the obtained formulae.

The proof is based on the fact that there are only finitely many $\mathcal{J}$-classes in a monoid. Therefore, when we evaluate a word from left to right, there is only a finite number of changes of $\mathcal{J}$-classes. We give some notations: for a word $w \in M^*$, $w[i,j]$ denotes the subword of w between indices i and j both included. For x an element, we denote by $\mathcal{J}(x)$ its $\mathcal{J}$-class.

Lemma 7.33.
Let w be a word of M^* of size n. Then there exist positions $1 = l_0 < l_1 < \ldots < l_m = n$ with $m \leq |M|$ and monoid elements $x_1, \ldots, x_m$ such that:

i) for $1 \leq i \leq m$, $w[1, l_i]$ evaluates to x_i,
ii) for $0 \leq i \leq m$ and $l_i < j \leq l_{i+1}$, $w[1, l_i + 1] \mathcal{J} w[1, j]$,
iii) for $1 \leq i \leq m$, $w[1, l_i] \not\mathcal{J} w[1, l_i + 1]$.

The situation is graphically pictured in Fig. 7.3.

Proof. We set $l_0 = 0$, and we will instanciate each element l_i by induction. We set l_{i+1} to be the greatest index greater than l_i such that $w[1, l_i + 1] \mathcal{J} w[1, l_{i+1}]$. By maximality, we know that iii) is satisfied for l_{i+1}. Then let $x_{i+1} = w[1, l_{i+1}]$. Moreover, x_i is a prefix of x_{i+1} and thus $x_i \geq_{\mathcal{J}} x_{i+1}$. Then $x_i \not\mathcal{J} x_i w_{l_i+1}$ and it is impossible that x_{i+1} and x_i are in the same $\mathcal{J}$-class. It implies that we have a chain of strict relations:

$$x_1 >_{\mathcal{J}} x_2 >_{\mathcal{J}} \cdots >_{\mathcal{J}} x_i.$$

Such a chain has size bounded by the size of M, concluding the proof. □

We now need a way to detect that a portion of a word evaluates in the same $\mathcal{J}$-class. A small purely algebraic fact on Green's relations is proven first.

Fact 7.34.
Let $x, y, z \in M$ such that $x \mathcal{L} y$ and they both belong to the $\mathcal{J}$-class J. Then:

$$xz \in J \Leftrightarrow yz \in J.$$

Proof. The statement is completely symmetrical, we only need to prove one direction. Suppose that $xz \in J$, which is equivalent to $x \mathcal{J} xz$. By assumption, $My \subseteq Mx$. In particular, $y \in Mx$ and therefore there exists $\alpha \in M$ such that $y = \alpha x$. It is a basic fact of monoid theory that if $x \mathcal{J} xz$ then $x \mathcal{R} xz$ (see for instance [95, Theorem 1.9]) As before, there exists $\beta \in M$ such that $xz\beta = x$. Multiplying on the left by α, we get $yz\beta = y$. It implies that $yz \mathcal{R} y$, and in particular that $yz \in J$. □

We extend $\geq_{\mathcal{J}}$ to make sense between an element and a $\mathcal{J}$-class as well. Let J be a $\mathcal{J}$-class of M, $w \in M^*$ and i a position. Let j be the least position such that $w[j, i] \geq_{\mathcal{J}} J$. Then $L_{\geq J}(w, i)$ is the evaluation of $w[j, i]$. Similarly, let j be the greatest position such that $w[i, j] \geq_{\mathcal{J}} J$. Then $R_{\geq J}(w, i)$ is the evaluation of $w[i, j]$. In case the indices j do not exists, we set $L_{\geq J}$ and $R_{\geq J}$ to the value $\perp$.

Lemma 7.35.
Let $w = uxyzv$ with $u, x, y, z, v \in M^*$, with xy_1 is some $\mathcal{J}$-class J. Then $xy \in J$ if and only if for all integers $1 \leq i \leq |y|$:

$$L_{\geq J}(w, i - 1 + |x| + |u|) \cdot y_i \geq_{\mathcal{J}} J.$$

Proof. First, assume that for every i we have $L_{\geq J}(w, i-1+|x|+|u|)\cdot y_i \geq_{\mathcal{J}} J$. We prove by induction that for every $1 \leq i \leq |y|$, the word $xy_1 \cdots y_i$ is in J. The case $i = 1$ is assumed to hold thanks to the statement. Now if $xy_1 \cdots y_i$ is in J, then $L_{\geq J}(w, i+|x|+|u|)$ necessarily contains the evaluation of $xy_1 \cdots y_i$ and the assumption gives that $xy_1 \cdots y_i y_{i+1} \geq_{\mathcal{J}} J$. However, it has also a prefix in J and therefore belongs to J.

Second, assume that there exists an i such that $L_{\geq J}(w, i-1+|x|+|u|)\cdot y_i \not\geq_{\mathcal{J}} J$. We take the smallest such i, implying as before that $xy_1 \cdots y_{i-1}$ is in J. Let k be the index such that $L_{\geq J}(w, i-1+|x|+|u|)$ is the evaluation of $w[k, i-1+|x|+|u|]$. The word $w[k, i-1+|x|+|u|]$ has $xy_1 \cdots y_{i-1}$ has a suffix and is $\mathcal{J}$-greater than J, and thus is in J. This gives that $w[k, i-1+|x|+|u|]$ and $xy_1 \cdots y_{i-1}$ are $\mathcal{L}$-equivalent, as they are $\mathcal{J}$-equivalent and $\mathcal{L}$-ordered. We know that $w[k, i-1+|x|+|u|]\cdot y_i$ is not in J, and so by Fact 7.34 $xy_1 \cdots y_i$ is not in J. This implies that xy cannot belong to J. □

As in the case of groups, the goal is to find a unary information that is enough to retreive all evaluations of infixes in a word. Here the information is the one given by the functions $L_{\geq J}$ and $R_{\geq J}$.

Lemma 7.36.
Let w be the maintained word in a dynamic problem. Assume we have, for every $\mathcal{J}$-class J and $x \in M$, unary relations $L_{\geq J,x}(i)$ and $R_{\geq J,x}(i)$ that respectively store whether $L_{\geq J}(w, i) = x$ and $R_{\geq J}(w, i) = x$.

Then for every $x \in M$, there exists a formula $\psi_x(j, k)$ in Σ_2 that holds if and only if $w_j \cdots w_k$ evaluates to x.

Proof. We first give the desired formula, and then prove that it correctly computes the value of the infixes. It is defined, where $J_1, \ldots, J_m$ denote the respective $\mathcal{J}$-classes of $x_1, \ldots, x_m$, by:

$$\psi_x(j,k) = \bigvee_{\substack{m \leq |M| \\ x_1,\ldots,x_m = x \in M}} \exists j = l_0 < \ldots < l_m = k,$$

$$\bigwedge_{0 \leq s < m} R_{\geq J_s, x_s}(j) \tag{a}$$

$$\wedge \quad \bigwedge_{0 \leq s \leq m} \left[\forall l_s < i < l_{s+1}, \bigvee_{y \cdot z \geq_{\mathcal{J}} J_s} L_{\geq J_s, y}(i) \wedge W_z(i+1) \right] \tag{b}$$

$$\wedge \quad \bigwedge_{0 < s < m} \bigvee_{x_s \cdot y \not\geq_{\mathcal{J}} J_s} W_y(l_s + 1). \tag{c}$$

Assume that $w_j \cdots w_k$ evaluates to x. Let $x_1, \cdots x_m \in M$ and $j = l_0 < \cdots < l_m = k$ given by Lemma 7.33. We will refer to i), ii) and iii) in this lemma. Let $0 \leq s < m$. By i) and iii), we know that the greatest word on the right of j and greater than J_s evaluates to x_s. By ii) and Lemma 7.35, we have that for every $l_s < i < l_{s+1}$, the evaluation of the greateast word on the left of i greater than J_s multiplied by w_{i+1} stays greater than J_s. By iii), the letter w_{l_s+1} makes x_s fall in $\mathcal{J}$-classes. Therefore the formula is satisfied for the parameters j and k.

Now assume that the formula is satisfied. We use the numbering in the formula. By induction, we prove that for every s, $w_j \cdots w_{l_s}$ evaluates to x_s. Because, $l_m = k$ and $x_s = x$, this

concludes the proof. By Lemma 7.35 and (b), $w_j \cdots w_{l_{s+1}}$ belongs to J. By (c), $w_j \cdots w_{l_{s+1}} w_{l_{s+1}+1}$ does not belong to J. So by (a), the greatest word greater than J_s on the right of j, which is necessarily $w_j \cdots w_{l_{s+1}}$, evaluates to x_{s+1}. □

The ability to simulate the maintenance of infixes is all we need to have an efficient dynamic logic program.

Theorem 7.37.
Every regular language is in UDyn-Σ_2.

Proof. Let $\mathcal{L}$ be a regular language and M be its syntactic monoid. We describe a dynamic logic program for the dynamic problem for M at $x \in M$. Let w be the maintained word. The schema $\mathcal{S}$ consists in unary relations, for J a $\mathcal{J}$-class and $y \in M$, $L_{\geq J,y}(i)$ and $R_{\geq J,y}(i)$ that respectively store whether $L_{\geq J}(w,i) = x$ and $R_{\geq J}(w,i) = x$. For $y \in M$, let ψ_y be the formula over $\mathcal{S}$ given by Lemma 7.36 that holds if and only if the infix of w evaluates to y. The membership formula simply uses ψ_x and is therefore in Σ_2:

$$\varphi_M = \psi_x(1, \max).$$

Now assume we see an operation $\mathbf{SET}_a(i)$. We can compute the new values of the infixes with the same quantifier-free formulae as in Lemma 7.28.

$$\psi_x^a(i,j,k) = \quad (j < i < k) \wedge \left(\bigvee_{\substack{y,z \in M \\ yaz = x}} \psi_y(j, i-1) \wedge \psi_z(i+1, k) \right)$$
$$\vee \quad (i < j \vee k < i) \wedge \psi_x(j,k).$$

There are no negations, hence these still are formulae in Σ_2. All is left to do is to give the updates formulae for $R_{\geq J,y}$, the case for $L_{\geq J,y}$ being symmetric.

$$\varphi^a_{R_{\geq J,y}}(i,j) = \exists k \geq j, \bigvee_{\substack{z \geq_{\mathcal{J}} J \\ zt \not\geq_{\mathcal{J}} M}} \psi_z^a(i,j,k) \wedge W_t(k+1).$$

Once again, the absence of negation of the presence of only an existential quantifier makes these formulae in Σ_2. Hence $M \in$ UDyn-Σ_2. Because the right-to-left implication of Corollary 7.27 is always true, without the variety assumption, we have that $\mathcal{L} \in$ UDyn-Σ_2. □

7.3.3 Regular languages in UDyn-FO^2

We end, for now, the study of the regular languages of Dyn-FO by considering UDyn-FO^2. We defined FO^2 by the set of first-order formulae that can use only two variable names. However, in dynamic logic programs, the update formulae already have several free variables. We therefore need another definition of FO^2 for formulae with free variables. We split the variables between those that can be quantified and the others. Formally, we write $\mathbb{V}_1 = \mathbb{V}_q \cup \mathbb{V}_f$. In the syntax, quantifications $\exists x \varphi$ only are for $x \in \mathbb{V}_q$. In dynamic logic programs, we ask that the variable i applied in $\varphi^a_R(i, \bar{j})$ is from $\mathbb{V}_f$, meaning that it cannot be quantified. All the other variables, like $\bar{j}$ in the definition, are from $\mathbb{V}_q$.

Definition 7.38.
We redefine FO^2 to be the set of first-order formulae that can use only two variables of $\mathbb{V}_q$. It is equivalent to ask that every subformula of the form $\exists x\varphi$ has at most one free variable in $\mathbb{V}_q$.

For sentences, this definition is equivalent to the one of Chapter 2. It only makes sense to consider Dyn-FO^2 when the schema is unary. Indeed, otherwise there are already more than two variables from $\mathbb{V}_q$ that come from the arity of the updates formulae, and therefore the computed information could be represented in unary. One of the good property of FO^2 over a unary schema with this definition is *compositionality*, in a sense we precise in the following statement.

Fact 7.39.
Let φ be a formula of FO^2 over a unary schema with an atomic subformula $R(x)$. Let also $\psi(i,x)$ be a FO^2 formula with two free variables $i \in \mathbb{V}_f$ and $x \in \mathbb{V}_q$. Then the formula defined as φ with $R(x)$ replaced by $\psi(i,x)$ with $i \in \mathbb{V}_f$ is also in FO^2.

Proof. The proof is direct. In the new formula, every existential subformula of $\psi(i,x)$ has at most one free variable in $\mathbb{V}_q$ by assumption. Moreover, $R(x)$ and $\psi(i,x)$ have the same free variables in $\mathbb{V}_q$, namely only x, and thus any other existential subformula has at most one free variable in $\mathbb{V}_q$ by assumption on φ. □

This property gives a hint that the class UDyn-FO^2 is stable under wreath products, making it possible to maintain complicated languages from simple ones.

Recall the definition of wreath product of M by N from Definition 2.18. If we unfold it, for $f_1,\dots,f_n \in M^N$ and $x_1,\dots,x_n \in N$, the product in $M \circ N$ is:

$$(f_1,x_1)\cdots(f_n,x_n) = (z \mapsto f_1(z)f_2(x_1 z)\cdots f_n(x_1\cdots x_{n-1}z), x_1\cdots x_n)$$

This motivates to store the evaluation of every prefix of a word w, instead of only being able to know its overall evaluation. Let M be a monoid. We say that prefix-M is in Dyn-FO (or any other class) if there is a dynamic logic program with distinguished unary relations P_x for $x \in M$ such that after any sequence of operations, P_x contains exactly all the integers i such that $w_1\cdots w_i = x$. In these programs, we can drop φ_M. It is clear that prefix-$M \in$ Dyn-FO implies $M \in$ Dyn-FO.

Lemma 7.40.
Let M and N be two monoids such that prefix-M and prefix-N are both in UDyn-FO^2. Then:

$$\text{prefix-}(M \circ N) \in \text{UDyn-FO}^2.$$

Proof. Let $\mathcal{P} = (\mathcal{S}, (\psi_R^a)_{R,a}, (\psi_R^{init})_R)$ and $\mathcal{P}' = (\mathcal{S}', (\psi_R'^a)_{R,a}, (\psi_R'^{init})_R)$ be the dynamic logic programs for N and M respectively, with unary schemas. We maintain the word w.

By Fact 7.39, we can compose formulae. Formally, assume we can maintain some relations Rs, and a relation T that uses the new values stored in R with a formula $\varphi_T^a(i,j)$. Then replacing every occurence of $R(x)$ in $\varphi_T^a(i,j)$ by $\varphi_R^a(i,x)$ gives a formula in FO^2 that can be used to maintain

the relation T.

The alphabet relations are of the form W_x for $x \in M \circ N$. For $f \in M^N$ and $y \in N$, we also maintain the relations W_f and W_y that carry the projection of the letters into M^N and N. Let π_1 be the projection into M^N and π_2 be the projection into N. They are constructed with, for instance:

$$\varphi^a_{W_f}(i,j) = \bigvee_{x \mid \pi_1(x)=f} W_x(j).$$

By using the relations in $\mathcal{S}$ and the formulae ψ^a_R, in which the occurences of alphabet relations are replaced by W_y for $y \in N$, we maintain relations P_y, for $y \in N$, that contain the evaluation in N of the prefixes of the projection of w on N.

Then, we add relations Q_f, for $f \in M^N$, that contain all integers i such that $f = z \mapsto f_i(x_1 \cdots x_{i-1} z)$. The update formulae are, for $f \in M^N$:

$$\varphi^a_{Q_f}(i,j) = \bigvee_{(g,y)\in X} W_g(j) \wedge P_y(j-1)$$

where X is the set of $(g,y) \in M^N \times N$ such that for all $z \in N$, $f(z) = g(yz)$.

We then use $\mathcal{P}'$ to have relations S_f for every $f \in M^N$, that contain the integers i such that $f = z \mapsto f_1(z) \cdots f_i(x_1 \cdots x_{i-1} z)$. We just replace the alphabet relations in ψ'^a_R by Q_f. To be precise, we consider M^N as the tuples of M indexed by N. Then we project into every component and maintain $|N|$ more relations. We use $\mathcal{P}'$ to maintain the prefixes, then we zip the information back into the relations S_f. All that can be done easily.

To conclude, we construct one last set of relations T_x for $x \in M \circ N$ that will carry the desired prefixes in the wreath product. The updates formulae are:

$$\varphi^a_{T_x}(i,j) = T_{\pi_1(x)}(j) \wedge P_{\pi_2(x)}(j).$$

□

The proof of Lemma 7.29 actually shows that we can maintain every prefix of groups in UDyn-Prop, and therefore in UDyn-FO^2 as well. Recall that U_2 is the syntactic monoid of the language $(a+b)^*a$. It has three elements $\{1, a, b\}$ with 1 neutral, a and b idempotent and $a \cdot b = b$ and $b \cdot a = a$.

Lemma 7.41.
We have:

$$\text{prefix-}U_2 \in \text{UDyn-FO}^2.$$

Proof. We can directly maintain prefix-U_2 without any other auxiliary relations. Let P_1, P_a and P_b be the relations with the prefixes. A word in U_2^* evaluates to 1 if and only if all letters are 1. Therefore for any letter c,

$$\varphi^c_{P_1}(i,j) = \forall x, x \leq j \Rightarrow W_1(x).$$

We now consider the maintenance of the relation P_a, the case for P_b being symmetric. It is equivalent with storing if the first non-neutral letter to the left of a position is an a or not. We describe all the update formulae.

- If a position i is set to a, then $P_a(j)$ stands if and only if there was already an a to the left

of j, or if there are only neutral letters between i and j. In symbols:

$$\varphi^a_{P_a}(i,j) = P_a(j) \vee (\forall x,\ i \leq x \leq j \Rightarrow W_1(x)).$$

- If a position i is set to b, then $P_a(j)$ stands if and only if there was already an a to the left of j and there is a non-neutral letter between i and j. In symbols:

$$\varphi^b_{P_a}(i,j) = P_a(j) \wedge \neg(j < i \wedge \forall x,\ i \leq x \leq j \Rightarrow W_1(j)).$$

- If a position i is set to 1, then $P_a(j)$ depends on whether there are any non-neutral letter between i and j. If so, then we keep the previous value. If not, then we look at the first non-neutral letter at the left of i. In symbols:

$$\begin{aligned}\varphi^1_{P_a}(i,j) = \quad & (\forall x,\ i \leq x \leq j \Rightarrow W_1(x)) \wedge P_a(i) \\ \vee \quad & (j < i \vee \exists x,\ i \leq x \leq j \wedge W_1(x)) \wedge P_a(j).\end{aligned}$$

□

We can conclude thanks to Krohn-Rhodes theorem.

Theorem 7.42.
Every regular language is in UDyn-FO^2.

Proof. Let $\mathcal{L}$ be a regular language and M be its syntactic monoid. By Krohn-Rodes theorem (Theorem 2.20), M divides a wreath product of the form $M_1 \circ \cdots \circ M_n$ where the M_i are either groups or U_2. By Lemma 7.29 and Lemma 7.41, we know that for every i, prefix-$M_i \in$ UDyn-FO^2. By Lemma 7.40, we have that $M_1 \circ \cdots \circ M_n$ is in UDyn-FO^2 as well. It is not hard to see that UDyn-FO^2 is stable by division, and therefore $M \in$ UDyn-FO^2. Because the right-to-left implication of Corollary 7.27 is always true, without the variety assumption, we have that $\mathcal{L} \in$ UDyn-FO^2.

□

Note that no inclusion is known between UDyn-Σ_2 and UDyn-FO^2, making both results uncomparable.

7.3.4 Going further

We have shown that the classes UDyn-Σ_2 and UDyn-FO^2 are powerful enough to express every regular language. We have also identified the regular languages of UDyn-Prop. The next obvious step is to identify the regular languages of UDyn-Σ_1, to have a complete picture. First of all, it is necessary to prove that UDyn-$\Sigma_1 \cap$ Reg forms a positive variety of regular languages. In the proof of closure under the inverse of a morphism μ in Lemma 7.26, we maintain the image $\mu(w)$ of the word and we have to modify every letter in $\mu(a)$ when we change a letter to a. This is a problem since we cannot compose formulae in Σ_1 when there are possibly negations in the quantifier-free formula. One way to tackle this issue is to change the setting and allow to change a constant number of letters at the same time. In this setting, the class UDyn-$\Sigma_1 \cap$ Reg is a variety of languages. Remark that every expressibility result of the chapter stands even if there are constantly many changes at the same time. For instance in the formulae given in Lemma 7.28, when updating the infix between j and k, we check which positions are updates between them, and recompute the value of the infix with finitely many previous infixes.

Moreover, the expressive power of UDyn-Σ_1 is not clear, and is hard to precisely describe. For instance, we know that all languages whose syntactic monoid is in **EJ** are maintainable. We recall that this class is the set of monoids whose idempotents generate a $\mathcal{J}$-trivial monoid.

Fact 7.43.
Let $\mathcal{L}$ be a regular language whose syntactic monoid is in **EJ**. It stands that $\mathcal{L} \in$ UDyn-Σ_1.

Proof. This class has been extensively studied by Pin in [93]. Therein (Theorem 6.1 and the main result), such a language $\mathcal{L}$ is shown to have a particular form. There exist group languages $\mathcal{G}_0, \dots, \mathcal{G}_n$ and letters $a_1, \dots, a_n$ such that

$$\mathcal{L} = \mathcal{G}_0 a_1 \mathcal{G}_1 \cdots a_n \mathcal{G}_n.$$

With Lemma 7.29, we have a unary dynamic logic program with quantifier-free formulae that maintain whether every prefix of the word is in each G_i. Thanks to the group property, it allows to compute all infixes with quantifier-free formulae. Now the membership formula quantifies over $x_1, \dots, x_n$ and checks if for every $1 \leq i \leq n$, there is the letter a_i at position x_i, and the infix between x_i and x_{i+1} is in $\mathcal{G}_i$. □

We also supect that all of **DA** can be maintained. However, we lack lower bounds techniques to show that some language is not in UDyn-Σ_1, which makes it difficult to build an intuition on candidates that could not be in this dynamic class. The lower bound from [139] used in Lemma 7.31 is tailored for quantifier-free formula, and is hard to extend to this setting, even without alternation of quantifiers.

Finally, it is left to future work to broaden the scope of this line of research. One direction would be to study the fine-grained complexity of the maintainance of regular languages under polylogarithmically many updates at the same time, and to have some control on the work in the RAM model as in [132]. Another would be to change the model to accept any initialisation of the word, instead of setting it to some fresh letter $\perp$. Algebraically, it is expected to behave like the study of regular languages without a neutral letter. For example, it is very likely that the regular languages of UDyn-Prop in this setting are **QLG** = **LG**.

Bibliography of the current chapter

[5] S. Alstrup, T. Husfeldt, and T. Rauhe. "Marked ancestor problems". In: *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*. 1998. DOI: 10.1109/SFCS.1998.743504.

[7] Antoine Amarilli, Louis Jachiet, and Charles Paperman. "Dynamic Membership for Regular Languages". In: *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Ed. by Nikhil Bansal, Emanuela Merelli, and James Worrell. Vol. 198. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. DOI: 10.4230/LIPIcs.ICALP.2021.116.

[8] Antoine Amarilli and Charles Paperman. "Locality and Centrality: The Variety ZG". In: *Logical Methods in Computer Science* Volume 19, Issue 4 (Oct. 2023). DOI: 10.46298/lmcs-19(4:4)2023.

[24] Jin-yi Cai. "Lower bounds for constant-depth circuits in the presence of help bits". en. In: *Information Processing Letters* 36.2 (Oct. 1990). DOI: 10.1016/0020-0190(90)90101-3.

[28] R. F. Cohen and R. Tamassia. "Dynamic expression trees". In: *Algorithmica* 13 (1995). DOI: 10.1007/BF01190506.

[33] Guozhu Dong and Jianwen Su. "Arity Bounds in First-Order Incremental Evaluation and Definition of Polynomial Time Database Queries". In: *Journal of Computer and System Sciences* 57.3 (1998). DOI: 10.1006/jcss.1998.1565.

[34] Guozhu Dong and Jianwen Su. "First-Order Incremental Evaluation of Datalog Queries". In: *Database Programming Languages (DBPL-4)*. Ed. by Catriel Beeri, Atsushi Ohori, and Dennis E. Shasha. London: Springer London, 1994. DOI: 10.1007/978-1-4471-3564-7_17.

[35] Guozhu Dong and Rodney W. Topor. "Incremental Evaluation of Datalog Queries". In: *Proceedings of the 4th International Conference on Database Theory*. ICDT '92. Berlin, Heidelberg: Springer-Verlag, 1992. DOI: 10.5555/645500.655916.

[40] M. Fredman and M. Saks. "The cell probe complexity of dynamic data structures". In: *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*. STOC '89. Seattle, Washington, USA: Association for Computing Machinery, 1989. DOI: 10.1145/73007.73040.

[43] Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. "The dynamic complexity of formal languages". In: *ACM Trans. Comput. Logic* 13.3 (2012). DOI: 10.1145/2287718.2287719.

[48] Étienne Grandjean and Louis Jachiet. *Which arithmetic operations can be performed in constant time in the RAM model with addition?* 2023. DOI: 10.48550/arXiv.2206.13851.

[61] William Hesse. "Dynamic Computational Complexity". PhD thesis. University of Massachusetts Amherst, 2003.

[64] Ismaël Jecker. "A Ramsey Theorem for Finite Monoids". In: *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*. Ed. by Markus Bläser and Benjamin Monmege. Vol. 187. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. DOI: 10.4230/LIPIcs.STACS.2021.44.

[81] Peter Bro Miltersen, Sairam Subramanian, Jeffrey Scott Vitter, and Roberto Tamassia. "Complexity models for incremental computation". In: *Theoretical Computer Science* 130.1 (1994). DOI: 10.1016/0304-3975(94)90159-7.

[90] Sushant Patnaik and Neil Immerman. "Dyn-FO: A Parallel, Dynamic Complexity Class". In: *Journal of Computer and System Sciences* 55.2 (1997). DOI: 10.1006/jcss.1997.1520.

[93] Jean-Eric Pin. "BG=PG: a success story". In: *NATO ASI Series C Mathematical and Physical Sciences-Advanced Study Institute* 466 (1995).

[95] Jean-Eric Pin. *Mathematical foundations of automata theory*. 2014. URL: http://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf.

[114] Thomas Schwentick and Thomas Zeume. "Dynamic complexity: recent updates". In: *ACM SIGLOG News* 3.2 (2016). DOI: 10.1145/2948896.2948899.

[122] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. "Dynamic word problems". In: *J. ACM* 44.2 (1997). DOI: 10.1145/256303.256309.

[132] Felix Tschirbs, Nils Vortmeier, and Thomas Zeume. "Dynamic Complexity of Regular Languages: Big Changes, Small Work". In: *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*. Ed. by Bartek Klin and Elaine Pimentel. Vol. 252. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. DOI: 10.4230/LIPIcs.CSL.2023.35.

[139] Thomas Zeume and Thomas Schwentick. "On the quantifier-free dynamic complexity of Reachability". In: *Information and Computation* 240 (2015). DOI: 10.1016/j.ic.2014.09.011.

Conclusion

Motivated by the success of the systematic study of regular languages inside circuits classes, we proposed to continue this study for other complexity classes, hoping to improve our understanding of them in the process. We obtained several new results in this thesis, for different models of computation:

- **Circuit complexity.** We proved that Σ_2 has the Straubing property, showing that the regular languages expressible by a depth-3 Boolean circuit with bounded top fan-in are precisely those expressible by a formula in $\Sigma_2[\mathrm{reg}]$.
- **Streaming complexity.** We characterised the regular path languages of trees that can be weakly validated both with a finite automaton and with a stackless automaton, a new model that can store the current depth of the streamed tree.
- **Incremental complexity in RAM.** We showed that the regular tree languages that can be maintained in RAM in constant time are those whose syntactic forest algebra has a vertical monoid in **ZG**, extending the result for words.
- **Incremental first-order complexity .** We studied the fine-grained complexity of regular languages in Dyn-FO. In particular, we showed that the regular languages in UDyn-Prop are precisely the group languages. We also gave efficient algorithms to maintain every regular language in UDyn-Σ_2 and UDyn-FO^2.

We already mentioned in their respective chapters the leads for future work in each of these areas. We hope to keep studying the regular languages in other complexity classes. For instance, Theorem 4.17 gives that the set of regular languages of AC^0 with logarithmically many maj-gates is **QA**, the same as for AC^0. It would be interesting to investigate the fine-grained complexity of AC^0 with fixed small depth and few additional maj-gates. For instance, we give a lower bound in Theorem 3.27 against circuits with a single maj-gate, depth 2, and linear size. We want to extend this result, with the help of the discriminator lemma (Lemma 3.25) to characterise the regular languages of very small classes of circuits with maj-gates.

Bibliography

[1] M. Ajtai. "Σ^1_1-Formulae on finite structures". en. In: *Annals of Pure and Applied Logic* 24.1 (July 1983). DOI: 10.1016/0168-0072(83)90038-6.

[2] Jorge Almeida. *Finite Semigroups and Universal Algebra*. World Scientific, 1995. DOI: 10.1142/2481.

[3] Jorge Almeida and Ana P. Escada. "The globals of pseudovarieties of ordered semigroups containing B_2 and an application to a problem proposed by Pin". eng. In: *RAIRO - Theoretical Informatics and Applications* 39.1 (2010). DOI: 10.1051/ita:2005001.

[4] Noga Alon and Pavel Pudlak. "Superconcentrators of depths 2 and 3; odd levels help (rarely)". In: *Journal of Computer and System Sciences* 48.1 (Feb. 1994). DOI: 10.1016/S0022-0000(05)80027-3.

[5] S. Alstrup, T. Husfeldt, and T. Rauhe. "Marked ancestor problems". In: *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*. 1998. DOI: 10.1109/SFCS.1998.743504.

[6] Ryan Alweiss, Shachar Lovett, Kewen Wu, and Jiapeng Zhang. "Improved bounds for the sunflower lemma". In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 2020. DOI: 10.4007/annals.2021.194.3.5.

[7] Antoine Amarilli, Louis Jachiet, and Charles Paperman. "Dynamic Membership for Regular Languages". In: *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Ed. by Nikhil Bansal, Emanuela Merelli, and James Worrell. Vol. 198. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. DOI: 10.4230/LIPIcs.ICALP.2021.116.

[8] Antoine Amarilli and Charles Paperman. "Locality and Centrality: The Variety ZG". In: *Logical Methods in Computer Science* Volume 19, Issue 4 (Oct. 2023). DOI: 10.46298/lmcs-19(4:4)2023.

[9] Vince Bárány, Christof Löding, and Olivier Serre. "Regularity Problems for Visibly Pushdown Languages". In: *Proc. STACS*. Springer, 2006. DOI: 10.1007/11672142_34.

[10] Corentin Barloy, Michael Cadilhac, Charles Paperman, and Thomas Zeume. "The Regular Languages of First-Order Logic with One Alternation". In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '22. Association for Computing Machinery, Aug. 2022. DOI: 10.1145/3531130.3533371.

[11] Corentin Barloy, Filip Murlak, and Charles Paperman. "Stackless Processing of Streamed Trees". In: *PODS*. June 2021. DOI: 10.4230/LIPIcs.

[12] David Barrington, Neil Immerman, and Howard Straubing. "On uniformity within NC_1". In: July 1988. DOI: 10.1109/SCT.1988.5262.

[13] David A. Barrington. "Bounded-width polynomial-size branching programs recognize exactly those languages in NC1". In: *Journal of Computer and System Sciences* 38 (1989). DOI: 10.1016/0022-0000(89)90037-8.

[14] David A. Barrington, Kevin Compton, Howard Straubing, and Denis Thérien. "Regular languages in NC_1". In: *Journal of Computer and System Sciences* 44.3 (1992). DOI: 10.1016/0022-0000(92)90014-A.

[15] David A. Mix Barrington and James C. Corbett. "On the Relative Complexity of Some Languages in NC_1". In: *Inf. Process. Lett.* 32.5 (1989). DOI: 10.1016/0020-0190(89)90052-5.

[16] David A. Mix Barrington and Denis Thérien. "Finite monoids and the fine structure of NC_1". In: *J. ACM* 35.4 (1988). DOI: 10.1145/48014.63138.

[17] Mikolaj Bojanczyk and Igor Walukiewicz. "Forest Algebras". en. In: *Logic and Automata* (Oct. 2006). DOI: https://hal.science/hal-00346087/.

[18] Mikołaj Bojańczyk. "Algebra for trees". en. In: *Handbook of Automata Theory*. Ed. by Jean-Éric Pin. Zuerich, Switzerland: European Mathematical Society Publishing House, Sept. 2021. DOI: 10.4171/Automata-1/22.

[19] Mikołaj Bojańczyk. "Recognisable Languages over Monads". In: *Developments in Language Theory*. Ed. by Igor Potapov. Cham: Springer International Publishing, 2015. DOI: 10.1007/978-3-319-21500-6_1.

[20] Burchard von Braunmühl and Rutger Verbeek. "Input-Driven Languages are Recognized in log n Space". In: *Proc. FCT 1983*. Springer, 1983. DOI: 10.1007/3-540-12689-9_92.

[21] J. Büchi. "Weak Second-Order Arithmetic and Finite Automata". In: *Mathematical Logic Quarterly* 6 (1960). DOI: 10.1007/978-1-4613-8928-6_22.

[22] S. R. Buss. "The Boolean formula value problem is in ALOGTIME". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987. DOI: 10.1145/28395.28409.

[23] Michaël Cadilhac and Charles Paperman. *The Regular Languages of Wire Linear AC 0*. en. Dec. 2021. DOI: 10.1007/s00236-022-00432-2.

[24] Jin-yi Cai. "Lower bounds for constant-depth circuits in the presence of help bits". en. In: *Information Processing Letters* 36.2 (Oct. 1990). DOI: 10.1016/0020-0190(90)90101-3.

[25] Robert D. Cameron, Ehsan Amiri, Kenneth S. Herdy, Dan Lin, Thomas C. Shermer, and Fred Popowich. "Parallel Scanning with Bitstream Addition: An XML Case Study". In: *Proc. Euro-Par 2011*. Springer, 2011. DOI: 10.1007/978-3-642-23397-5_2.

[26] Laura Chaubard, Jean-Éric Pin, and Howard Straubing. "Actions, wreath products of $\mathcal{C}$-varieties and concatenation product". In: *Theoretical Computer Science*. In honour of Professor Christian Choffrut on the occasion of his 60th birthday 356.1 (May 2006). DOI: 10.1016/j.tcs.2006.01.039.

[27] Cristiana Chitic and Daniela Rosu. "On Validation of XML Streams Using Finite State Machines". In: *Proc. WebDB 2004*. ACM, 2004. DOI: 10.1145/1017074.1017096.

[28] R. F. Cohen and R. Tamassia. "Dynamic expression trees". In: *Algorithmica* 13 (1995). DOI: 10.1007/BF01190506.

[29] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. 2008. DOI: https://inria.hal.science/hal-03367725.

[30] Luc Dartois and Charles Paperman. "Alternation Hierarchies of First Order Logic with Regular Predicates". en. In: *Fundamentals of Computation Theory*. Ed. by Adrian Kosowski and Igor Walukiewicz. Cham: Springer International Publishing, 2015. DOI: 10.1007/978-3-319-22177-9_13.

[31] Denis Debarbieux, Olivier Gauwin, Joachim Niehren, Tom Sebastian, and Mohamed Zergaoui. "Early nested word automata for XPath query answering on XML streams". In: *Theor. Comput. Sci.* 578 (2015). DOI: 10.1016/j.tcs.2015.01.017.

[32] Danny Dolev, Cynthia Dwork, Nicholas Pippenger, and Avi Wigderson. "Superconcentrators, Generalizers and Generalized Connectors with Limited Depth (Preliminary Version)". In: Jan. 1983. DOI: 10.1145/800061.808731.

[33] Guozhu Dong and Jianwen Su. "Arity Bounds in First-Order Incremental Evaluation and Definition of Polynomial Time Database Queries". In: *Journal of Computer and System Sciences* 57.3 (1998). DOI: 10.1006/jcss.1998.1565.

[34] Guozhu Dong and Jianwen Su. "First-Order Incremental Evaluation of Datalog Queries". In: *Database Programming Languages (DBPL-4)*. Ed. by Catriel Beeri, Atsushi Ohori, and Dennis E. Shasha. London: Springer London, 1994. DOI: 10.1007/978-1-4471-3564-7_17.

[35] Guozhu Dong and Rodney W. Topor. "Incremental Evaluation of Datalog Queries". In: *Proceedings of the 4th International Conference on Database Theory*. ICDT '92. Berlin, Heidelberg: Springer-Verlag, 1992. DOI: 10.5555/645500.655916.

[36] Patrick Dymond. "Input-driven Languages Are in Log N Depth". In: *Inf. Process. Lett.* 26.5 (Jan. 1988). DOI: 10.1016/0020-0190(88)90148-2.

[37] Samuel Eilenberg. "Automata, Languages and Machines, Vol. B". In: Verlag: Academic Press Inc, 1976.

[38] R. Erdos P.and Raso. "Intersection theorems for systems of finite sets". In: *journal of the London Mathematical Society* 35.1 (1960).

[39] Ronald Fagin. "Generalized first-order spectra, and polynomial time recognizable sets". In: *SIAM-AMS Proc.* 7 (Jan. 1974).

[40] M. Fredman and M. Saks. "The cell probe complexity of dynamic data structures". In: *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*. STOC '89. Seattle, Washington, USA: Association for Computing Machinery, 1989. DOI: 10.1145/73007.73040.

[41] Merrick Furst, James B Saxe, and Michael Sipser. "Parity, circuits, and the polynomial-time hierarchy". en. In: (1984). DOI: 10.1007/BF01744431.

[42] Mai Gehrke, Serge Grigorieff, and Jean-Éric Pin. "Duality and Equational Theory of Regular Languages". In: *Automata, Languages and Programming*. Ed. by Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. DOI: 10.1007/978-3-540-70583-3_21.

[43] Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. "The dynamic complexity of formal languages". In: *ACM Trans. Comput. Logic* 13.3 (2012). DOI: 10.1145/2287718.2287719.

[44] Evangelos Georganas, Sasikanth Avancha, Kunal Banerjee, Dhiraj D. Kalamkar, Greg Henry, Hans Pabst, and Alexander Heinecke. "Anatomy of high-performance deep learning convolutions on SIMD architectures". In: *Proc. SC 2018*. IEEE / ACM, 2018. DOI: 10.5555/3291656.3291744.

[45] Mateusz Gienieczko, Filip Murlak, and Charles Paperman. "Supporting Descendants in SIMD-Accelerated JSONPath". In: *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*. ASPLOS '23. , Vancouver, BC, Canada, Association for Computing Machinery, 2024. DOI: 10.1145/3623278.3624754.

[46] Mika Göös, Artur Riazanov, Anastasia Sofronova, and Dmitry Sokolov. "Top-Down Lower Bounds for Depth-Four Circuits". In: *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. 2023. DOI: 10.1109/FOCS57990.2023.00063.

[47] Parikshit Gopalan and Rocco Servedio. *Learning and Lower Bounds for AC^0 with Threshold Gates*. en. Tech. rep. TR10-074. Electronic Colloquium on Computational Complexity (ECCC), Apr. 2010. DOI: 10.1007/978-3-642-15369-3_44.

[48] Étienne Grandjean and Louis Jachiet. *Which arithmetic operations can be performed in constant time in the RAM model with addition?* 2023. DOI: 10.48550/arXiv.2206.13851.

[49] J. A. Green. "On the Structure of Semigroups". In: *Annals of Mathematics* 54.1 (1951).

[50] Alejandro Grez, Cristian Riveros, and Martín Ugarte. "A Formal Framework for Complex Event Processing". In: *Proc. ICDT 2019*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. DOI: 10.4230/LIPIcs.ICDT.2019.5.

[51] Nathan Grosshans. "A Note on the Join of Varieties of Monoids with LI". In: *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*. 2021. DOI: 10.4230/LIPIcs.MFCS.2021.51.

[52] Nathan Grosshans, Pierre Mckenzie, and Luc Segoufin. "The Power of Programs over Monoids in DA". In: *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*. 2017. DOI: 10.4230/LIPIcs.MFCS.2017.2.

[53] Sascha Grunert and Daniel Schmidt. *A comparison of regex engines*. 2017. URL: https://rust-leipzig.github.io/regex/2017/03/28/comparison-of-regex-engines/.

[54] Ashish Kumar Gupta and Dan Suciu. "Stream Processing of XPath Queries with Predicates". In: *Proc. SIGMOD 2003*. ACM, 2003. DOI: 10.1145/872757.872809.

[55] Yuri Gurevich and Harry R. Lewis. "A logic for constant-depth circuits". In: *Information and Control* 61.1 (1984). DOI: 10.1016/S0019-9958(84)80062-5.

[56] András Hajnal, Wolfgang Maass, Pavel Pudlák, Mário Szegedy, and György Turán. "Threshold circuits of bounded depth". en. In: *Journal of Computer and System Sciences* 46.2 (Apr. 1993). DOI: 10.1016/0022-0000(93)90001-D.

[57] Kristoffer Arnsfelt Hansen and Michal Koucký. "A New Characterization of ACC0 and Probabilistic CC0". en. In: *computational complexity* 19.2 (May 2010). DOI: 10.1007/s00037-010-0287-z.

[58] J. Håstad, S. Jukna, and P. Pudlàk. "Top-down lower bounds for depth-three circuits". en. In: *Computational Complexity* 5.2 (June 1995). DOI: 10.1007/BF01268140.

[59] Johan Håstad. "Computational limitations for small depth circuits". en. Thesis. Massachusetts Institute of Technology, 1986.

[60] Yeye He, Siddharth Barman, and Jeffrey F. Naughton. "On Load Shedding in Complex Event Processing". In: *Proc. ICDT 2014*. OpenProceedings.org, 2014. DOI: 10.5441/002/icdt.2014.23.

[61] William Hesse. "Dynamic Computational Complexity". PhD thesis. University of Massachusetts Amherst, 2003.

[62] John Hopcroft and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[63] Neil Immerman. "Languages that Capture Complexity Classes". In: *SIAM Journal on Computing* 16.4 (1987). DOI: 10.1137/0216051.

[64] Ismaël Jecker. "A Ramsey Theorem for Finite Monoids". In: *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*. Ed. by Markus Bläser and Benjamin Monmege. Vol. 187. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. DOI: 10.4230/LIPIcs.STACS.2021.44.

[65] Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-24508-4.

[66] Stasys Jukna. *Extremal Combinatorics: With Applications in Computer Science*. 1st. Springer Publishing Company, Incorporated, 2010.

[67] Mark Kambites. "On the Krohn–Rhodes complexity of semigroups of upper triangular matrices". In: *International Journal of Algebra and Computation* 17.01 (2007). DOI: 10.1142/S0218196707003548.

[68] Hans Kamp. "Tense Logic and the Theory of Linear Order". PhD thesis. Ucla, 1968.

[69] SC Kleene. "Representation of events in nerve nets and finite automata". In: *Automata Studies: Annals of Mathematics Studies. Number 34* 34 (1956).

[70] Eryk Kopczynski. "Invisible Pushdown Languages". In: *Proc. LICS 2016*. ACM, 2016. DOI: 10.1145/2933575.2933579.

[71] M. Koucky, S. Poloczek, C. Lautemann, and Denis Therien. "Circuit lower bounds via Ehrenfeucht-Fraisse games". In: vol. 2006. Jan. 2006. DOI: 10.1109/CCC.2006.12.

[72] Michal Koucký, Pavel Pudlak, and Denis Therien. "Bounded-depth circuits: Separating wires from gates". In: May 2005. DOI: 10.1145/1060590.1060629.

[73] Andreas Krebs and Howard Straubing. *Regular languages defined by first-order formulas without quantifier alternation*. Aug. 2022. DOI: 10.48550/arXiv.2208.10480.

[74] Kenneth Krohn and John Rhodes. "Algebraic Theory of Machines. I. Prime Decomposition Theorem for Finite Semigroups and Machines". In: *Transactions of the American Mathematical Society* 116 (1965).

[75] Geoff Langdale and Daniel Lemire. "Parsing gigabytes of JSON per second". In: *VLDB J.* 28.6 (2019). DOI: 10.1007/s00778-019-00578-5.

[76] Per Lindström. "First Order Predicate Logic with Generalized Quantifiers". In: *Theoria* 32.3 (1966). DOI: 10.1111/j.1755-2567.1966.tb00600.x.

[77] Nancy A. Lynch. "Log Space Recognition and Translation of Parenthesis Languages". In: *J. ACM* 24 (1977). DOI: 10.1145/322033.322037.

[78] Alexis Maciel, Pierre Péladeau, and Denis Thérien. "Programs over semigroups of dot-depth one". In: *Theoretical Computer Science* 245.1 (2000). DOI: 10.1016/S0304-3975(99)00278-9.

[79] Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. research monograph no. 65)*. The MIT Press, 1971.

[80] Or Meir and Avi Wigderson. "Prediction from Partial Information and Hindsight, with Application to Circuit Lower Bounds". en. In: *computational complexity* 28.2 (June 2019). DOI: 10.1007/s00037-019-00177-4.

[81] Peter Bro Miltersen, Sairam Subramanian, Jeffrey Scott Vitter, and Roberto Tamassia. "Complexity models for incremental computation". In: *Theoretical Computer Science* 130.1 (1994). DOI: 10.1016/0304-3975(94)90159-7.

[82] Filip Murlak, Charles Paperman, and Michal Pilipczuk. "Schema Validation via Streaming Circuits". In: *Proc. PODS 2016*. ACM, 2016. DOI: 10.1145/2902251.2902299.

[83] Anil Nerode. "Linear automaton transformations". In: *Proceedings of the American Mathematical Society* 9.4 (1958).

[84] Dan Olteanu. "SPEX: Streamed and Progressive Evaluation of XPath". In: *IEEE Trans. Knowl. Data Eng.* 19.7 (2007). DOI: 10.1109/TKDE.2007.1063.

[85] Shoumik Palkar, Firas Abuzaid, Peter Bailis, and Matei Zaharia. "Filter Before You Parse: Faster Analytics on Raw Data with Sparser". In: *Proc. VLDB Endow.* 11.11 (2018). DOI: 10.14778/3236187.3236207.

[86] Yannis Papakonstantinou and Victor Vianu. "DTD inference for views of XML data". In: *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. PODS '00. New York, NY, USA: Association for Computing Machinery, May 2000. DOI: 10.1145/335168.335173.

[87] Charles Paperman. "Circuits booléens, prédicats modulaires et langages réguliers". PhD thesis. Université Paris Diderot, 2014.

[88] Charles Paperman. *Semigroup Online*. 2015. URL: https://paperman.name/semigroup/.

[89] Charles Paperman, Sylvain Salvati, and Claire Soyez-Martin. "An Algebraic Approach to Vectorial Programs". In: *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*. Ed. by Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté. Vol. 254. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. DOI: 10.4230/LIPIcs.STACS.2023.51.

[90] Sushant Patnaik and Neil Immerman. "Dyn-FO: A Parallel, Dynamic Complexity Class". In: *Journal of Computer and System Sciences* 55.2 (1997). DOI: 10.1006/jcss.1997.1520.

[91] Pierre Péladeau. "Logically defined subsets of $\mathbb{N}^k$". In: *Theoretical Computer Science* 93.2 (1992). DOI: 10.1016/0304-3975(92)90328-D.

[92] Jean-Eric Pin. "A variety theorem without complementation". In: *Russian Mathematics (Izvestija vuzov.Matematika)* 39 (1995).

[93] Jean-Eric Pin. "BG=PG: a success story". In: *NATO ASI Series C Mathematical and Physical Sciences-Advanced Study Institute* 466 (1995).

[94] Jean-Eric Pin. *Handbook of Automata Theory*. EMS Press, 2021. DOI: 10.4171/automata.

[95] Jean-Eric Pin. *Mathematical foundations of automata theory*. 2014. URL: http://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf.

[96] Jean-Eric Pin. "On reversible automata". In: *Proc. LATIN 1992*. Springer, 1992.

[97] Jean-Eric Pin. "Profinite Methods in Automata Theory". In: *26th International Symposium on Theoretical Aspects of Computer Science*. Ed. by Susanne Albers and Jean-Yves Marion. Vol. 3. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2009. DOI: 10.4230/LIPIcs.STACS.2009.1856.

[98] Jean-Eric Pin, Arnaud Pinguet, and Pascal Weil. "Ordered categories and ordered semigroups". en. In: *Communications in Algebra* 30.12 (Dec. 2002). DOI: 10.1081/AGB-120016004.

[99] Jean-Eric Pin and Howard Straubing. "Some results on C-varieties". eng. In: *RAIRO - Theoretical Informatics and Applications* 39.1 (Mar. 2010). DOI: 10.1051/ita:2005014.

[100] Jean-Eric Pin and Pascal Weil. "Polynomial Closure and Unambiguous Product". In: *Proceedings of the 22nd International Colloquium on Automata, Languages and Programming*. ICALP '95. Berlin, Heidelberg: Springer-Verlag, 1995. DOI: 10.5555/646249.685349.

[101] Jean-Eric Pin and Pascal Weil. "The wreath product principle for ordered semigroups". In: *Communications in Algebra* 30 (2002).

[102] Nicholas Pippenger. "Superconcentrators". In: *All HMC Faculty Publications and Research* (Jan. 1977). DOI: 10.1137/0206022.

[103] Thomas Place and Marc Zeitoun. "Going Higher in First-Order Quantifier Alternation Hierarchies on Words". In: *Journal of the ACM* 66.2 (Mar. 2019). DOI: 10.1145/3303991.

[104] Vladimir V. Podolskii. "Exponential lower bound for bounded depth circuits with few threshold gates". en. In: *Information Processing Letters* 112.7 (Mar. 2012). DOI: 10.1016/j.ipl.2011.12.011.

[105] Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. "Rethinking SIMD Vectorization for In-Memory Databases". In: *Proc. SIGMOD 2015*. ACM, 2015. DOI: 10.1145/2723372.2747645.

[106] P. Pudlák. "Communication in bounded depth circuits". en. In: *Combinatorica* 14.2 (June 1994). DOI: 10.1007/BF01215351.

[107] Anup Rao. *Coding for Sunflowers*. Feb. 2020. DOI: 10.48550/arXiv.1909.04774.

[108] A. A. Razborov. "Lower bounds on the size of bounded depth circuits over a complete basis with logical addition". en. In: *Mathematical notes of the Academy of Sciences of the USSR* 41.4 (Apr. 1987). DOI: 10.1007/BF01137685.

[109] Jan Reiterman. "The Birkhoff theorem for finite algebras". In: *algebra universalis* 14 (1982). DOI: 10.1007/BF02483902.

[110] Gang Ren, Peng Wu, and David A. Padua. "An Empirical Study On the Vectorization of Multimedia Applications for Multimedia Extensions". In: *Proc. IPDPS 2005*. IEEE, 2005. DOI: 10.1109/IPDPS.2005.94.

[111] Benjamin Rossman. "On the constant-depth complexity of k-clique". In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. STOC '08. Victoria, British Columbia, Canada: Association for Computing Machinery, 2008. DOI: 10.1145/1374376.1374480.

[112] M. P. Schützenberger. "Une théorie algébrique du codage". fre. In: *Séminaire Dubreil. Algèbre et théorie des nombres* 9 (1955).

[113] M.P. Schützenberger. "On finite monoids having only trivial subgroups". In: *Information and Control* 8.2 (1965). DOI: 10.1016/S0019-9958(65)90108-7.

[114] Thomas Schwentick and Thomas Zeume. "Dynamic complexity: recent updates". In: *ACM SIGLOG News* 3.2 (2016). DOI: 10.1145/2948896.2948899.

[115] Luc Segoufin and Cristina Sirangelo. "Constant-Memory Validation of Streaming XML Documents Against DTDs". In: *Proc. ICDT 2007*. Springer, 2007. DOI: 10.1007/11965893_21.

[116] Luc Segoufin and Victor Vianu. "Validating Streaming XML Documents". In: *Proc. PODS 2002*. ACM, 2002. DOI: 10.1145/543613.543622.

[117] Claude E. Shannon. "A symbolic analysis of relay and switching circuits". In: *Transactions of the American Institute of Electrical Engineers* 57.12 (1938). DOI: 10.1109/T-AIEE.1938.5057767.

[118] Claude. E. Shannon. "The synthesis of two-terminal switching circuits". In: *The Bell System Technical Journal* 28.1 (1949). DOI: 10.1002/j.1538-7305.1949.tb03624.x.

[119] Imre Simon. "Piecewise testable events". In: *Automata Theory and Formal Languages*. Ed. by H. Brakhage. Berlin, Heidelberg: Springer Berlin Heidelberg, 1975. DOI: 10.1007/3-540-07407-4_23.

[120] Michael Sipser. "A topological view of some problems in complexity theory". en. In: *Mathematical Foundations of Computer Science 1984*. Ed. by M. P. Chytil and V. Koubek. Vol. 176. Berlin/Heidelberg: Springer-Verlag, 1984. DOI: 10.1007/BFb0030341.

[121] Michael Sipser. "Borel sets and circuit complexity". In: *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*. STOC '83. New York, NY, USA: Association for Computing Machinery, 1983. DOI: 10.1145/800061.808733.

[122] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. "Dynamic word problems". In: *J. ACM* 44.2 (1997). DOI: 10.1145/256303.256309.

[123] R. Smolensky. "Algebraic methods in the theory of lower bounds for Boolean circuit complexity". en. In: *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*. New York, New York, United States: ACM Press, 1987. DOI: 10.1145/28395.28404.

[124] Howard Straubing. "Constant-depth periodic circuits". In: *International Journal of Algebra and Computation* 01.01 (1991). DOI: 10.1142/S0218196791000043.

[125] Howard Straubing. "Families of recognizable sets corresponding to certain varieties of finite monoids". In: *Journal of Pure and Applied Algebra* 15.3 (1979). DOI: 10.1016/0022-4049(79)90024-0.

[126] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. en. Boston, MA: Birkhäuser, 1994. DOI: 10.1007/978-1-4612-0289-9.

[127] Howard Straubing. "Finite semigroup varieties of the form V * D". en. In: *Journal of Pure and Applied Algebra* 36 (Jan. 1985). DOI: 10.1016/0022-4049(85)90062-3.

[128] Dan Suciu. "From searching text to querying XML streams". In: *J. Discrete Algorithms* 2.1 (2004). DOI: 10.1007/3-540-45735-6_2.

[129] Pascal Tesson and Denis Thérien. "Diamonds are forever: the variety da". In: *Semigroups, Algorithms, Automata and Languages*. WORLD SCIENTIFIC, Nov. 2002. DOI: 10.1142/9789812776884_0021.

[130] Wolfgang Thomas. "Languages, Automata, and Logic". In: *Handbook of Formal Languages: Volume 3 Beyond Words*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997. DOI: 10.1007/978-3-642-59126-6_7.

[131] Bret Tilson. "Categories as algebra: An essential ingredient in the theory of monoids". en. In: *Journal of Pure and Applied Algebra* 48.1 (Sept. 1987). DOI: 10.1016/0022-4049(87)90108-3.

[132] Felix Tschirbs, Nils Vortmeier, and Thomas Zeume. "Dynamic Complexity of Regular Languages: Big Changes, Small Work". In: *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*. Ed. by Bartek Klin and Elaine Pimentel. Vol. 252. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. DOI: 10.4230/LIPIcs.CSL.2023.35.

[133] Leslie G. Valiant. "Graph-theoretic arguments in low-level complexity". en. In: *Mathematical Foundations of Computer Science 1977*. Ed. by Jozef Gruska. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1977. DOI: 10.1007/3-540-08353-7_135.

[134] Leslie G. Valiant. "On non-linear lower bounds in computational complexity". en. In: *Proceedings of seventh annual ACM symposium on Theory of computing - STOC '75*. Albuquerque, New Mexico, United States: ACM Press, 1975. DOI: 10.1145/800116.803752.

[135] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. *Improving the speed of neural networks on CPUs*. 2011.

[136] Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer Berlin Heidelberg, 1999. DOI: 10.1007/978-3-662-03927-4.

[137] Xiang Wang, Yang Hong, Harry Chang, KyoungSoo Park, Geoff Langdale, Jiayu Hu, and Heqing Zhu. "Hyperscan: A Fast Multi-pattern Regex Matcher for Modern CPUs". In: *Proc. NSDI 2019*. USENIX Association, 2019.

[138] Ryan Williams. "Non-uniform ACC Circuit Lower Bounds". In: *2011 IEEE 26th Annual Conference on Computational Complexity*. 2011. DOI: 10.1109/CCC.2011.36.

[139] Thomas Zeume and Thomas Schwentick. "On the quantifier-free dynamic complexity of Reachability". In: *Information and Computation* 240 (2015). DOI: 10.1016/j.ic.2014.09.011.

[140] Haopeng Zhang, Yanlei Diao, and Neil Immerman. "On complexity and optimization of expensive queries in complex event processing". In: *Proc. SIGMOD 2014*. ACM, 2014. DOI: 10.1145/2588555.2593671.

[141] Yichun Zhang. *Regex Engine Matching Speed Benchmark*. 2015. URL: http://openresty.org/misc/re/bench/.

[142] Jingren Zhou and Kenneth A. Ross. "Implementing database operations using SIMD instructions". In: *Proc. SIGMOD 2002*. ACM, 2002. DOI: 10.1145/564691.564709.

On the complexity of regular languages.

Abstract

Regular languages, languages computed by finite automata, are among the simplest objects in theoretical computer science. This thesis explores several computation models: parallel computing with Boolean circuits, processing of structured documents in streaming, and information maintenance on a structure subject to incremental updates. For the latter, auxiliary structures are either stored in RAM or represented by databases updated by logical formulae.
This thesis investigates the resources required to compute classes of regular languages in each of these models. The methods employed rely on the interaction between algebra, logic, and combinatorics, notably exploiting the theory of finite semigroups. This approach of complexity has proven extremely fruitful, particularly in the context of Boolean circuits, where regular languages play a central role. This research angle was crystallised by Howard Straubing in his book "Finite Automata, Formal Logic, and Circuit Complexity", where he conjectured that any regular language definable by an arbitrary formula from a logic fragment can be rewritten to use only simple, regular predicates.
The first objective of this manuscript is to prove this conjecture in the case of the Σ_2 fragment of first-order logic with a single alternation of quantification. A second result provides a description of space complexity, in the streaming model, for verifying regular properties on trees. Special attention is given to properties verifiable in constant and logarithmic space. A third objective is to describe all regular tree languages that can be incrementally maintained in constant time in RAM. Finally, a last part focuses on the development of efficient logical formulae for maintaining all regular languages in the relational model.

Keywords: regular languages, circuit complexity, finite semigroups, formal logic, tree languages, dynamic problems.

Sur la complexité des langages réguliers.

Résumé

Les langages réguliers, langages calculés par automates finis, sont parmi les objets les plus simples de l'informatique théorique. Cette thèse étudie plusieurs modèles de calculs : le calcul parallèle avec les circuits booléens, le traitement en flot de documents structurés, et la maintenance d'information sur une structure soumise à des mises à jour incrémentales. Pour ce dernier modèle, les structures auxiliaires sont soit stockées en RAM, soit représentées par des bases de données mises à jour par des formules logiques.
Cette thèse étudie les ressources nécessaires pour calculer des classes de langages réguliers dans chacun de ces modèles. Les méthodes employées exploitent l'interaction entre algèbre, logique et combinatoire, en mettant notamment à profit la théorie des semigroupes finis. Cette approche de la complexité s'est notamment montrée extrêmement fructueuse dans le cadre des circuits booléens, où les langages réguliers jouent un rôle central. Cette angle de recherche a été cristallisé par Howard Straubing dans son livre "Finite Automata, Formal Logic, and Circuit Complexity", où il émet la conjecture que tout langage régulier définissable par une formule arbitraire d'un fragment de logique peut être réécrite en utilisant uniquement des prédicats simples, c'est-à-dire réguliers.
Le premier but de ce manuscrit est de prouver cette conjecture dans le cas du fragment Σ_2 de la logique du premier-ordre avec une seule alternance de quantification. Un deuxième résultat propose une description de la complexité en espace, dans le modèle de flot, pour vérifier des propriétés régulières sur des arbres. Une attention particulière est portée aux propriétés vérifiables en espace constant et logarithmique. Un troisième objectif est de décrire tous les langages réguliers d'arbres pouvant être maintenus incrémentalement en temps constant en RAM. Enfin, une dernière partie porte sur le développement de formules logiques efficaces pour maintenir tous les langages réguliers dans le modèle relationnel.

Mots clés : languages réguliers, complexité de circuits, semigroupes finis, logique formelle, languages d'arbres, problèmes dynamiques.

CRIStAL
Université de Lille - Campus scientifique – Bâtiment ESPRIT – Avenue Henri Poincaré – 59655 Villeneuve d'Ascq