A combinatorial approach to the separation problem for regular languages

Lorijn Van Rooijen

▶ To cite this version:

Lorijn Van Rooijen. A combinatorial approach to the separation problem for regular languages. Programming Languages [cs.PL]. Université de Bordeaux, 2014. English. <NNT : 2014BORD0229>. <tel-01154542>

HAL Id: tel-01154542

https://tel.archives-ouvertes.fr/tel-01154542

Submitted on 22 May 2015

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET INFORMATIQUE

par

Lorijn van Rooijen

pour obtenir le grade de

Docteur

Spécialité : Informatique

Une approche combinatoire du problème de séparation pour les langages réguliers

---

Soutenue le 04-12-2014 au Laboratoire Bordelais de Recherche en Informatique (LaBRI) devant le jury suivant :

Marie-Pierre Béal	Université Paris-Est Marne la vallée	Examinatrice
Manfred Kufleitner	Université de Stuttgart	Examinateur
Wim Martens	Université de Bayreuth	Rapporteur
Jean-Éric Pin	Université Paris Diderot - Paris 7	Rapporteur
Thomas Place	Université de Bordeaux	Examinateur
Pascal Weil	Université de Bordeaux	Examinateur
Marc Zeitoun	Université de Bordeaux	Directeur de thèse

‘Cada idioma es un modo distinto de sentir el universo o de percibir el universo’

-

Jorge Luis Borges

Résumé

Une approche combinatoire du problème de séparation pour les langages réguliers

Le problème de séparation pour une classe de langages $\mathcal{S}$ est le suivant : étant donnés deux langages L_1 et L_2, existe-t-il un langage appartenant à $\mathcal{S}$, qui contient L_1, en étant disjoint de L_2 ?

Si les langages à séparer sont des langages réguliers, le problème de séparation pour la classe $\mathcal{S}$ est plus général que le problème de l'appartenance à cette classe, et nous fournit des informations plus détaillées sur la classe. Ce problème de séparation apparaît dans un contexte algébrique sous la forme des parties ponctuelles, et dans un contexte profini sous la forme d'un problème de séparation topologique. Pour quelques classes de langages spécifiques, ce problème a été étudié en utilisant des méthodes profondes de la théorie des semigroupes profinis.

Dans cette thèse, on s'intéresse, dans un premier temps, à la décidabilité de ce problème pour plusieurs sous-classes des langages réguliers. Dans un second temps, on s'intéresse à obtenir un langage séparateur, s'il existe, ainsi qu'à la complexité de ces problèmes.

Nous établissons une approche générique pour prouver que le problème de séparation est décidable pour une classe de langages donnée. En utilisant cette approche, nous obtenons la décidabilité du problème de séparation pour les langages testables par morceaux, les langages non-ambigus, les langages localement testables, et les langages localement testables à seuil. Ces classes correspondent à des fragments de la logique du premier ordre, et sont parmi les classes de langages réguliers les plus étudiées. De plus, cette approche donne une description d'un langage séparateur, pourvu qu'il existe.

Mots clés : Langages réguliers, Séparation, Logiques, Automates, Monoïdes, Langages testables par morceaux, Langages non-ambigus, Langages localement testables, Langages localement testables à seuil, Langages algébriques.

Abstract

A combinatorial approach to the separation problem for regular languages

The separation problem, for a class $\mathcal{S}$ of languages, is the following: given two input languages, does there exist a language in $\mathcal{S}$ that contains the first language and that is disjoint from the second language?

For regular input languages, the separation problem for a class $\mathcal{S}$ subsumes the classical membership problem for this class, and provides more detailed information about the class. This separation problem first emerged in an algebraic context in the form of pointlike sets, and in a profinite context as a topological separation problem. These problems have been studied for specific classes of languages, using involved techniques from the theory of profinite semigroups.

In this thesis, we are not only interested in showing the decidability of the separation problem for several subclasses of the regular languages, but also in constructing a separating language, if it exists, and in the complexity of these problems.

We provide a generic approach, based on combinatorial arguments, to proving the decidability of this problem for a given class. Using this approach, we prove that the separation problem is decidable for the classes of piecewise testable languages, unambiguous languages, and locally (threshold) testable languages. These classes are defined by different fragments of first-order logic, and are among the most studied classes of regular languages. Furthermore, our approach yields a description of a separating language, in case it exists.

Key words: Regular languages, Separation, Logics, Automata, Monoids, Piecewise testable languages, Unambiguous languages, Locally testable languages, Locally threshold testable languages, Context-free languages.

Acknowledgments

First of all, I would like to thank my supervisor Marc, for having introduced me to the field of research in which this thesis is situated, and for always having taken the time to patiently explain things to me. I am happy to have had the opportunity to learn a lot from both you and Thomas, and I am grateful to both of you for all of our discussions. I also want to thank Pascal for the conversations we had, which have helped me a lot. I would like to thank Jean-Eric and Wim for having accepted to review my thesis and for having read the manuscript carefully. Also, I would like to thank all the members of the jury for their presence at the defense.

During these three years, besides working at LaBRI, I also worked on a parallel project at LIAFA in Paris, and I want to thank Mai and Sam for the collaboration in this project. I also would like to thank Mai for having brought the possibility of continuing my studies in France to my mind. I very much enjoyed taking part in the FREC seminar at LIAFA, and would like to thank all of the participants. Also, I am grateful to Laure, Dora, Florent and Alexis for brightening up my visits to Paris.

A special word of thanks to all the people who have assisted me with all the administrative procedures that I encountered after moving to France. I am grateful to Marc, Lebna, Brigitte, Pierre and Eve for helping me finding my way through the system. I also want to thank Maïté and Elia for their kindness and for all their help.

Taking up this PhD position in France also meant leaving behind many people in Holland. I want to thank my family for supporting my decision to move to Bordeaux and for always having been understanding. I am happy that you have all found the time to visit my little home in the castle of Condorcet. I would also very much like to thank my friends back in Holland for all of their support during these years, and for their friendship. In particular, I am grateful to Vera, Renée, Sietske, Dineke, Marieke, Inge and Laura for bringing a lot of true Dutch gezelligheid to Talence during their visits!

Then, I would like to express my gratitude to the people with whom I have spent most time during these last three years: my colleagues at LaBRI. I would like to thank Pierre, Anaïs, Petru, Sri, Lorenzo, Anna, Dominik, Sagnik, Razanne, Allyx, Tung, Martin, Jerôme, Andjela, Pauline and all the others. I especially want to thank Eve, who has helped me a lot to master the French language, and with whom I have had the chance to spend many great moments during these three years. Merci pour ton soutien et ton amitié !

Many many thanks are in order to all the girls from the lunch group, Nesrine, Claire, Wafa, Olfa, Omessaad, Thao, Thanh, Eya, Feten, Khaoula and Samah, for all of the nice moments we shared inside, but especially outside of LaBRI. I feel lucky to have met all of you and I would not have been able to complete these three years without you girls, thanks for making the time in Bordeaux so much more enjoyable!

Finally, I would like to thank my fiancé Maarten for always having been there for me, during these three years and before. For encouraging me, until the very end, to overcome all difficulties along the way, and for all the great times we have spent together in all of the places where our work has taken us. My biggest motivation while finishing this thesis was the prospect of finally being together.

Résumé étendu

Motivations

En informatique, un grand intérêt est porté au raisonnement sur des données et des programmes. Les mots finis servent de modèles de base et sont parmi les structures les plus étudiées en informatique. La logique nous fournit une façon formelle et intuitive de raisonner sur des mots finis. Un autre formalisme pour décrire les propriétés des mots finis est le formalisme des automates finis. Ce formalisme est moins intuitif, mais mieux adapté au traitement algorithmique.

Le théorème de Büchi est un résultat fondamental qui lie le formalisme des automates finis et la logique. Ce théorème dit qu'un langage est reconnaissable par un automate fini si et seulement si le langage est définissable par une formule de la logique monadique du second ordre (MSO). La preuve de cette équivalence est directe et constructive : étant donné un automate fini reconnaissant un certain langage régulier, la preuve nous donne une construction d'une formule MSO qui définit ce langage. Réciproquement, en partant d'une formule MSO, nous pouvons construire un automate fini qui reconnaît le langage défini par la formule. Néanmoins, cette dernière transformation peut être d'une mauvaise complexité.

La logique monadique du second ordre, interpretée sur les mots finis, permet d'exprimer des propriétés des positions et des lettres présentes sur ces positions. De plus, des propriétés des ensembles des positions peuvent être exprimées. Dans la logique du premier ordre, cette quantification des ensembles n'est pas permis. Usuellement, on s'intéresse à des propriétés qui en fait n'ont pas besoin de cette quantification du second ordre. Pour des raisons de complexité, on préfère avoir des formules les plus simples possible. On s'intéresse donc à savoir si un langage régulier pourrait être défini dans un fragment plus simple de MSO. Cette question s'appelle le problème de l'appartenance (à ce fragment plus simple).

Le problème de l'appartenance

Données : Un langage régulier L et un fragment $\mathcal{F}$.

Question : Le langage L est-il définissable dans le fragment $\mathcal{F}$?

Afin de résoudre le problème de l'appartenance, il faut trouver un ensemble de propriétés qui ne sont vraies que pour tous les langages définissables dans $\mathcal{F}$, et desquelles la vérité est décidable. Autrement dit, il faut trouver une caractérisation décidable de $\mathcal{F}$. Trouver une telle caractérisation nous fournit normalement une compréhension profonde du fragment.

Un fragment naturel qui a été étudié est la logique du premier ordre ($\mathsf{FO}(<)$), dans laquelle la quantification des ensembles n'est pas autorisée. Cette logique étant encore compliquée, il est naturel aussi d'étudier des restrictions de $\mathsf{FO}(<)$. Par exemple, les fragments obtenus en limitant le nombre de variables permises, ou en limitant l'alternance des quantificateurs (c'est-à-dire les fragments qu'on trouve dans la *quantifier alternation hierarchy*), ou encore en exigeant des propriétés combinatoires sur les langages.

Il y a beaucoup de fragments naturels et intéressants de MSO. Pour chaque fragment, on aimerait bien avoir une caractérisation décidable. Bien que le théorème de Büchi fournisse une caractérisation des langages définissables dans MSO d'une façon directe, il se trouve que pour résoudre les problèmes de l'appartenance à ces fragments, on se sert toujours de l'algèbre (plus précisément du monoïde syntaxique) comme intermédiaire. Il est important de noter que, pour un langage régulier, son monoïde syntaxique est une abstraction finie du langage, qu'on sait calculer à partir du langage.

Un des premiers résultats dans cette ligne de recherche a été le résultat de Schützenberger [Sch65] qui dit qu'un langage régulier est un langage sans étoile si et seulement si le monoïde syntaxique de ce langage est apériodique (c'est-à-dire pour chaque $s \in M$, il existe $n \in \mathbb{N}$ tel que $s^{n+1} = s^n$). Cette preuve est constructive : à partir d'un langage régulier ayant un monoïde syntaxique apériodique, on obtient une expression sans étoile. De plus, McNaughton et Papert [MP71] ont montré que les langages sans étoile sont précisément les langages définissables dans $\mathsf{FO}(<)$. Cette preuve est également constructive. Ces deux résultats ensemble résolvent le problème de l'appartenance à $\mathsf{FO}(<)$.

Ensuite, le problème de l'appartenance à la classe de langages testables par morceaux a été résolu par Simon [Sim75]. Cette classe se trouve dans la *quantifier altercation hierarchy* sous la forme de la classe $\mathcal{B}\Sigma_1$. La classe de langages testables par morceaux est donc parmi les classes les plus bas de la hiérarchie, mais elle pose déjà des défis. Simon a trouvé la caractérisation algébrique suivante de cette classe : un langage est testable par morceaux si et seulement si son monoïde syntaxique est $\mathcal{J}$-trivial (c'est à dire pour tout $s, m \in M$, si $MsM = MtM$, alors $s = t$). Vu que les monoïdes avec lesquels nous travaillons sont finis, ceci est une propriété décidable. Stern [Ste85] a traduit cette caractérisation algébrique à une caractérisation des automates minimaux, ce qui donne une approche pour tester si un langage est testable par morceaux en temps polynomial.

La connexion entre les langages réguliers et les monoïdes finis a été développée plus profondément par Eilenberg [Eil76]. Le théorème d'Eilenberg donne un cadre général aux résultats précédents : ce théorème dit qu'il existe une correspondance bijective entre les variétés de langages (ensembles de langages réguliers fermés par les opérations booléennes, image homomorphe inverse et quotient gauche et quotient droit) et les variétés de monoïdes finis (ensembles de monoïdes finis fermés par sous-monoïde, image homomorphe et produit direct fini). De plus, le théorème de Reiterman [Rei82] dit que toute variété de monoïdes finis peut être définie par un ensemble d'identités (qui sont des équations formelles entre des mots profinis).

Cette connexion entre les langages réguliers et l'algèbre a été très fructueuse. Néanmoins, le problème de l'appartenance semble être trop restrictif : il y a des problèmes de l'appertenance ouvert depuis longtemps. De plus, en n'étudiant que le problème de l'appartenance, on ignore certaines propriétés pertinentes de la classe. Par exemple, un petit calcul montre que les langages $(aa)^*$ et $(bb)^+$ ne sont pas définissables dans $\mathsf{FO}(<)$. Mais, on observe tout de suite que $\mathsf{FO}(<)$ est capable de percevoir que ces langages sont des langages disjoints. La formule $\forall x.a(x)$ en est un témoin (la formule étant vraie pour tous les mots de $(aa)^*$, et fausse pour tous les mots de $(bb)^+$). Cette information n'est pas capturée par le problème de l'appartenance, ce qui nous mène à regarder un problème plus général.

Le problème de séparation

Le problème de séparation est une généralisation naturelle du problème de l'appartenance, qui est formulée comme suit :

Le problème de séparation

Données : Deux langages L_1 et L_2, ainsi qu'une classe $\mathcal{S}$.

Question : Existe-t-il un langage appartenant à $\mathcal{S}$, qui contient L_1, en étant disjoint de L_2 ?

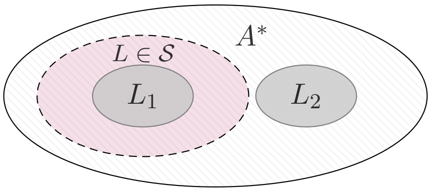


Figure 1: Le langage L, appartenant à $\mathcal{S}$, sépare L_1 et L_2.

Si les langages à séparer sont des langages réguliers, le problème de séparation pour la classe $\mathcal{S}$ est plus général que le problème de l'appartenance à cette classe (car L_1 appartient à $\mathcal{S}$ si et seulement si il existe un langage appartenant à $\mathcal{S}$, qui contient L_1, en étant disjoint de $A^* \backslash L_1$), et nous fournit des informations plus détaillées sur la classe.

Ce problème de séparation apparaît dans un contexte algébrique sous la forme des parties ponctuelles, et dans un contexte profini sous la forme d'un problème de séparation topologique. Il a été montré par Almeida [Alm99] que ces problèmes sont équivalents à notre problème de séparation. Pour quelques classes de langages spécifiques, ce problème a été étudié en utilisant des méthodes profondes de la théorie des semigroupes profinis. Néanmoins, en n'étant pas constructives, les solutions de ces problèmes algébriques et topologiques ne peuvent que donner la décidabilité du problème de séparation, et ne peuvent pas donner une description d'un langage séparateur.

Cette perspective a permis, entre autres, de résoudre les problèmes de séparation pour les classes suivantes : des langages reconnus par des groupes finis [Ash91, RZ93, Aui04, AS05], les langages sans étoile [Hen88, HRS10], les langages testables par morceaux [AZ97, ACZ08], les langages localement testables [Ste01, Nog10], et les langages localement testables à seuil [Ste98, Ste01].

Contributions

Dans cette thèse, on s'intéresse, dans un premier temps, à la décidabilité du problème de séparation pour plusieurs sous-classes des langages réguliers, en utilisant des méthodes combinatoires et constructives. Dans un second temps, on s'intéresse à obtenir un langage séparateur, s'il existe, ainsi qu'à la complexité de ces problèmes.

Nous établissons une approche générique pour prouver que le problème de séparation est décidable pour une classe de langages donnée. L'idée fondamentale est de stratifier la classe $\mathcal{S}$ de séparateurs, selon un paramètre qui est pertinent pour cette classe. Par exemple, comme paramètre, on pourrait choisir la taille des suffixes ou des sous-mots inspectés, ou le rang de quantification d'un formule, ou bien la taille d'un monoïde reconnaissant le langage. La sous-classe de $\mathcal{S}$ obtenue en fixant le paramètre inférieur ou égal à k est notée comme $\mathcal{S}[k]$. Le paramètre doit être choisi tel que la classe $\mathcal{S}[k]$ soit finie.

Cette stratification donne un semi-algorithme pour décider si deux langages sont séparables par un langage appartenant à $\mathcal{S}$: en testant subséquemment toutes les classes finies $\mathcal{S}[1], \mathcal{S}[2], \ldots$ Néanmoins, afin de décider si deux langages ne sont pas séparables, il faut a priori analyser toutes les classes $\mathcal{S}[k]$. Notre solution pour ce problème est de définir, pour chaque k, une relation $\mathrm{I}_k^{\mathcal{S}}$ sur les élements d'un monoïde reconnaissant les deux langages, ou sur les paires d'états d'un automate reconnaissant les deux langages, qui capture le fait que les langages correspondant aux élements (ou paires d'états) ne peuvent pas être séparés par un langage de $\mathcal{S}[k]$.

Plus précisément, soit M un monoïde fini et $\varphi : A^* \to M$ un morphisme surjectif. Pour $s, t \in M$,

$$(s,t) \in \mathrm{I}_k^{\mathcal{S}} \quad \Leftrightarrow \quad \varphi^{-1}(s) \text{ et } \varphi^{-1}(t) \text{ ne sont pas séparables par } \mathcal{S}[k].$$

Ces relations sont ordonnées par inclusion comme suit :

$$\mathrm{I}^{\mathcal{S}} = \bigcap_{n \in \mathbb{N}} \mathrm{I}_n^{\mathcal{S}} \subseteq \ldots \subseteq \mathrm{I}_{k+1}^{\mathcal{S}} \subseteq \mathrm{I}_k^{\mathcal{S}} \subseteq \ldots \subseteq \mathrm{I}_1^{\mathcal{S}}.$$

Comme ces relations se raffinent, et comme ce sont des relations sur des ensembles finis, elles finissent par se stabiliser. On s'intéresse à trouver une borne à partir de laquelle toutes les relations de cette séquence sont égales. Ceci est un problème difficile, et la solution est spécifique pour chaque classe étudiée dans cette thèse. C'est dans cette étape qu'on utilise des méthodes combinatoires. Notons que, si on trouve une telle borne k, le problème de séparation pour la classe $\mathcal{S}$ se réduit au problème de séparation pour la sous-classe finie $\mathcal{S}[k]$. De plus, dans nos preuves pour les bornes, nous trouvons des critères sur l'automate ou le monoïde pour la $\mathcal{S}$-séparabilité. Ces critères nous donnent des meilleurs rèsultats de complexité que les approches *brute-force* (c'est-à-dire en énumérant $\mathcal{S}[k]$) nous auraient donné.

En utilisant cette approche, nous obtenons la décidabilité du problème de séparation pour les langages testables par morceaux, les langages non-ambigus, les langages localement testables, et les langages localement testables à seuil. Ces classes correspondent à des fragments de la logique du premier ordre, et sont parmi les classes de langages réguliers les plus étudiées. De plus, cette approche donne une description d'un langage séparateur, pourvu qu'il existe. Nous donnons dans cette thèse également des résultats de complexité pour ces problèmes de séparation.

Table of contents

Introduction **1**

1 Preliminaries **11**
1.1 Words and languages 11
1.2 Automata 12
1.3 Recognition by semigroups and monoids 13
1.4 Varieties and free pro-V semigroups 14
1.4.1 Identities 14
1.5 Logic on words 15
1.5.1 Different fragments of $\mathsf{FO}(<)$ 16

2 Introduction to the separation problem **17**
2.1 The separation problem 18
2.2 Different points of view on the separation problem 21
2.2.1 Algebraic view: 2-pointlike sets 22
2.2.2 Topological view: closures in the free pro-V semigroup 24
2.2.3 Combinatorial view: indistinguishable pairs 26
2.3 Basic examples 29
2.3.1 Example I: SI 29
2.3.2 Example II: K 32

3 Group languages **37**
3.1 Characterizations of group languages 37
3.2 The separation problem for group languages 38
3.2.1 Closures in the free group 40
3.2.2 Decidability of G-separability and a construction of a separator 42
3.2.3 Closures in the free monoid 44

4 Piecewise testable languages **47**
4.1 Characterizations of piecewise testable languages 48
4.1.1 Logical characterization 48
4.1.2 Algebraic characterization 50
4.1.3 Graphical characterization 51
4.2 Separation by piecewise testable languages 51
4.2.1 PT-indistinguishable pairs of states 52
4.2.2 Common patterns 53

4.2.3 A common pattern yields PT-indistinguishability 54
4.2.4 PT-indistinguishability stems from a common pattern 55
4.2.5 Intermezzo: an alternative method 61
4.2.6 Separation theorem for piecewise testable languages 63
4.3 Complexity of PT-separability 64

5 Unambiguous languages **67**
5.1 Characterizations of unambiguous languages 68
5.1.1 Logical characterization 68
5.1.2 Algebraic characterization 70
5.2 Separation by unambiguous languages 70
5.2.1 Fixpoint algorithm to compute $\mathsf{FO}^2(<)$-indistinguishable pairs 72
5.2.2 Correctness of the fixpoint algorithm 73
5.2.3 Completeness of the fixpoint algorithm 74
5.2.4 Proof of the separation theorem for unambiguous languages 81
5.3 Complexity of separation by unambiguous languages 82

6 Locally testable and locally threshold testable languages **83**
6.1 Locally testable and locally threshold testable languages 84
6.1.1 Locally testable languages 86
6.1.2 Locally threshold testable languages 87
6.2 Separation for a fixed counting threshold 88
6.2.1 Common d-patterns 89
6.2.2 Separation theorem for a fixed counting threshold 90
6.2.3 A common d-pattern yields equivalent words for all profile sizes 92
6.2.4 From a common d-pattern in M to a common d-pattern in $\mathcal{A}$ 94
6.2.5 Bounding the profile size 95
6.2.6 Decidability of separation by locally testable languages 99
6.3 Separation for full locally threshold testable languages 100
6.3.1 Decidability of separation by locally threshold testable languages . . . 100
6.3.2 Bounding the counting threshold 102
6.3.3 Optimality of the bound on the counting threshold 105
6.4 Complexity of LT- and LTT-separability 107
6.4.1 Upper complexity bounds 107
6.4.2 Lower complexity bounds 113
6.5 Separating context-free languages by LT and LTT languages 115

Conclusion and perspectives **121**

Bibliography **125**

Index **133**

Introduction

Background and motivation

In computer science, one often wants to model data, programs or executions. One of the most basic structures to model these are finite words, which are therefore among the most used structures in computer science. Logic provides an intuitive and formal way to reason about such structures, and it is usually easy to specify a property of finite words via logic. However, one would also want to be able to use algorithms on logical structures, for example to test whether the intersection of two sets of words described by logical formulas is empty. Logics are not designed for such algorithmic treatment, which brings a need for another formalism to describe properties of finite words that is better suited for algorithmic treatment. Finite state automata form such a formalism. Compared to logic, it is, however, less intuitive to specify properties of finite words in this formalism.

A fundamental result relating the formalism of finite state automata and logic is Büchi's theorem. This theorem states that a language is recognized by a finite state automaton if and only if it can be defined by a monadic second-order (MSO) formula. For each regular language, an MSO-formula can be constructed from a finite state automaton recognizing the language. Conversely, from an MSO-formula, a finite state automaton that recognizes the language defined by the formula can be constructed. The proof of this equivalence is direct and elementary. The transformation from an MSO-formula to a finite state automaton can however have a bad complexity in the worst case.

The logic MSO has become a standard formalism to reason about structures, such as (infinite) words and trees. When interpreted on words, it can be used to express properties about positions and about the letters that these positions carry. Furthermore, in MSO, also properties about sets of positions can be expressed, which is not the case for first-order logic (FO). However, we are usually interested in properties that do not need this additional power of MSO. Transforming a formula to an automaton is non-elementary in the size of the formula, and it is therefore best to express a property by a formula that is as lean as possible. For example, a fragment of formulas with less quantifier alternations will be more amenable to efficient algorithmic treatment.

It is thus useful to know whether a regular language can also be defined in a simpler fragment of MSO. This amounts to studying the expressive power, or the membership problem, of the simpler fragment. This problem is indeed defined as follows.

Membership problem

Input: A regular language L and a fragment $\mathcal{F}$.

Question: Is L definable in the fragment $\mathcal{F}$?

To show that the membership problem is decidable, one has to find a decidable characterization for the class of languages definable in the fragment, i.e. one has to find a set of properties, which are verified by precisely those languages that are definable in the fragment, and which can be tested algorithmically for a given language. Finding a decidable characterization usually provides a deep insight in the considered fragment.

A natural fragment that, amongst others, has been studied in this respect is first-order logic, which only allows quantification over individual variables, and forbids quantification over sets of variables. Since this logic still allows complicated specifications, it is still natural to look at restrictions of first-order logic. For example, the quantifier alternation hierarchy provides natural logical fragments to study. This hierarchy contains fragments of first-order logic with a specific number of alternations between existential and universal quantifier blocks in the quantifier prefix, which is followed by a quantifier-free part. An example of a fragment that is low in this hierarchy is the fragment $\mathcal{B}\Sigma_1(<)$. This is the class of $\mathsf{FO}(<)$-formulas that are boolean combinations of $\mathsf{FO}(<)$-formulas having a quantifier prefix consisting of only existential quantifiers. A formula is thus in this logic if and only if it has no quantifier alternations. These formulas can only express a simple combinatorial property, namely the presence or absence of pieces, i.e. scattered subwords, in words.

Another way to obtain simpler and interesting fragments of $\mathsf{FO}(<)$ is to restrict the number of variable names that may be used in a formula. It follows from a result of Kamp [Kam68] that restricting to three reusable variable names yields a fragment that has the same expressive power as $\mathsf{FO}(<)$. The fragment of $\mathsf{FO}(<)$ in which only two reusable variable names are allowed is denoted by $\mathsf{FO}^2(<)$. It was proved in [PW97, TW98] that this fragment has the same expressive power as one of the fragments occurring in the quantifier alternation hierarchy.

Instead of logical fragments, one can also study classes defined by combinatorial properties. As explained above, the fragment $\mathcal{B}\Sigma_1(<)$, for example, can also be defined in this way. Another example is the class of locally testable languages (LT), which consists of languages defined by the presence or absence of prefixes, infixes and suffixes in the words of the languages.

There are thus many natural fragments of MSO, and we are interested in knowing their expressive power. Büchi's theorem gives a characterization of the MSO-definable languages, but it is not clear which classes of languages are defined by the fragments. For each fragment, one would like to have an algorithm to test whether a regular language is definable in the fragment. If it is definable in the fragment, one would like to obtain a formula, from the fragment, that witnesses this fact. This has been a successful line of research, of which we will now describe the main achievements.

The membership problem

While Büchi's theorem can be proved in a direct way, without the use of algebra, this is not the case for the membership problems for fragments of the logic. Indeed, for these membership problems, algebra is needed as an intermediate structure. This role is played by the syntactic monoid of a language. This is an abstraction of a language, in which combinatorial properties of the language are translated to simple algebraic properties. For a regular language, the syntactic monoid is finite and it is computable from the language.

Solving the membership problem in this way consists of two parts. Given a language definable in the fragment, one should prove that its syntactic monoid satisfies the characterization. On the other hand, given a language whose syntactic monoid satisfies the characterization, one should construct a formula in the fragment, that defines the language. This second part is particularly difficult and often uses induction on the structure of the monoid.

A remarkable result that formed a starting point in this line of research is Schützenberger's theorem. In [Sch65], Schützenberger proved that a regular language is star-free if and only if its syntactic monoid is aperiodic (i.e. for each $s \in M$, there is $n \in \mathbb{N}$ such that $s^{n+1} = s^n$). This proof is constructive: given a language with an aperiodic syntactic monoid, a star-free expression is calculated. In [MP71], McNaughton and Papert furthermore showed, also constructively, that the star-free languages are precisely the languages definable by first-order logic. Aperiodicity of the syntactic monoid thus is a decidable characterization for first-order logic. This algebraic characterization can be translated to properties of the minimal automaton. However, there is no direct way known to characterize the expressive power of $\mathsf{FO}(<)$ without using algebra.

Another important result that uses the connection between regular languages and algebra was obtained by Simon in [Sim75]. In this paper, a decidable characterization of the piecewise testable languages, which are the $\mathcal{B}\Sigma_1(<)$-definable languages, is given. As explained above, one motivation for studying this class is that it is among the lowest levels of the quantifier alternation hierarchy, but still is a challenging class. The characterization is again algebraic: Simon's theorem says that the syntactic monoids of these languages are exactly the $\mathcal{J}$-trivial monoids (i.e. the monoids such that for all $s, t \in M$, $MsM = MtM$ implies $s = t$). One can observe that, since we are working with finite monoids, this property is decidable. This algebraic characterization has also been translated to a characterization of the minimal automata [Ste85]. In general, having a characterization of the minimal automata gives interesting complexity bounds. For the class of piecewise testable languages, it is shown in [Ste85] that membership can be decided in Ptime.

The connection between regular languages and their finite abstractions in the form of syntactic monoids, which capture the properties of a language that are relevant with respect to the membership problem, was further developed in Eilenberg's theorem [Eil76]. This theorem provides a general framework in which Schützenberger's theorem and Simon's theorem fit. It states that there is a one-to-one correspondence between varieties of finite monoids (sets of finite monoids closed under taking submonoids, homomorphic images and finite direct products) and varieties of regular languages (sets of regular languages closed under Boolean operations, taking inverse homomorphic images, and left and right residuals).

Moreover, Reiterman showed in [Rei82] that every variety of finite monoids can be described by a set of identities, which are formal equalities between profinite words. For example, aperiodic monoids are defined by the condition that for each $s \in M$, there is $n \in \mathbb{N}$ such that $s^{n+1} = s^n$. This variety can be described by a single identity, viz. $x^{\omega+1} = x^\omega$. Most interesting classes can be given by a finite set of identities, of which it can be tested whether the syntactic monoid satisfies them. Such a characterization of a class yields that the membership problem for this class is decidable. This connection between regular languages and algebra has been a fruitful approach to obtaining decidable characterizations.

Drawbacks of this approach

A drawback of this approach to understanding classes of languages, however, is that it is not modular. That is, even if a class of languages is just slightly modified, one can usually not reuse the arguments for the original class to conclude something about the new class. As logical fragments are often defined from weaker fragments, for example by adding a predicate to the fragment, this is indeed a drawback.

One would want to be able to transfer decidability of a weaker fragment to decidability of a stronger fragment. For this, one would need two things. First of all, from an algebraic characterization of the first fragment, one should be able to obtain an algebraic characterization of the second fragment. And secondly, this characterization should be testable, i.e. decidability of membership should be preserved under this operation. For example, adding the successor relation to a fragment of $\mathsf{FO}(<)$ that is characterized by some variety V often corresponds to the operation $\mathsf{V} \mapsto \mathsf{V} * \mathsf{D}$. It is shown in [Aui10] that decidability of membership is not preserved under this operation.

This drawback, as well as the fact that this approach seemed unable to bring decidable characterizations for higher levels of the quantifier alternation hierarchy (i.e. from $\mathcal{B}\Sigma_2(<)$ onwards), indicates that studying the membership problem does not give sufficient understanding of the fragments.

The separation problem

A natural generalization of the membership problem, which provides more information about the fragment and which behaves better under operations, is the separation problem. For a given fragment, this problem asks whether, for two input languages, there exists a language definable in this fragment which contains one of the input languages and which is disjoint from the other input language. In this case we say that the two input languages can be separated by a language definable in this fragment, see also Figure 2. The separation problem is thus defined as follows.

Separation problem

Input: Two languages L_1 and L_2, and a fragment $\mathcal{F}$.

Question: Are L_1 and L_2 separable by a language definable in $\mathcal{F}$?

The membership problem reduces to the separation problem. Indeed, a language is in a given class if and only if it can be separated from its complement by a language from the class. The separation problem can thus be used to test the expressive power of a class, but it is much more informative: it tests whether two input languages are sufficiently disjoint to be perceived as such from the point of view of this class. It thus also provides information about languages that are outside of the class. The expressive power (eg. the quantifier rank) needed

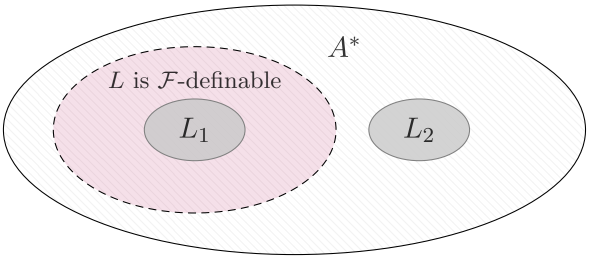


Figure 2: L_1 and L_2 are separable by a language definable in $\mathcal{F}$.

to separate the languages tells something about how different they are, from the point of view of the class of separators.

Our approach to the separation problem is to take one monoid (or automaton) that recognizes both languages, and show for each pair of monoid elements (or pairs of states) whether the languages determined by these can be separated. If the languages are separable, we give a description of a separating language, and if the languages cannot be separated, we know which parts of the languages are responsible for this. Thus, like a positive answer, also a negative answer to the separation problem provides much finer information about languages outside of the class than the membership problem would. Indeed, if a language is not definable in the fragment, an answer to the membership problem would just tell why the fragment cannot separate it from its complement.

State of the art of the separation problem

As a response to the non-modularity of the membership problem, many properties that are stronger than decidability of this problem have been introduced. One of the first such properties is based on the algebraic concept of pointlike sets, studied in [Hen88] for aperiodic semigroups. Later, Almeida proved in [Alm99] that computing the pointlike sets of size 2 for a variety V is equivalent to solving the separation problem for the class of V-recognizable languages. The relation between the V-pointlike sets of size 2 of a monoid M and the separation problem is the following: a set $\{m, n\} \subseteq M$ is V-pointlike if and only if $\varphi^{-1}(m)$ and $\varphi^{-1}(n)$ are not separable by a V-recognizable language, for any morphism $\varphi : A^* \to M$. In [Alm99], it is furthermore shown that the separation problem is equivalent to a topological problem. This topological problem is to test emptiness of the intersection of the topological closures of two regular languages inside the free pro-V semigroup, a topological space which is usually uncountable.

One reason for which pointlike sets have received attention is that they can be used to obtain decidability of the membership problem of varieties built from other varieties. For instance, in [HMPR91], it is explained that the computability of the pointlike sets for the variety of groups (denoted by G) is proved in Ash's proof [Ash91] of the type II conjecture of Rhodes [Rho87], and that this computability of G-pointlike sets yields that other varieties, built in a certain way from this one, have a decidable membership problem. This has been extended in [PW96], where it is shown that if a variety has computable pointlike sets, the membership problem stays decidable if one applies certain operations to it.

Also, from [Ste01], we know that the operation $\mathsf{V} \mapsto \mathsf{V} * \mathsf{D}$ even preserves computability of pointlike sets, thus in particular, if V has computable pointlike sets, the membership problem for the variety $\mathsf{V}*\mathsf{D}$ is still decidable. This is in contrast with the situation for the membership problem: it was shown in [Aui10] that decidability of the membership problem is not preserved under this operation.

For a small number of specific varieties, these pointlike sets have been shown to be computable, yielding that the separation problem for that variety is decidable. This has been shown, for example, for the following varieties: group languages [Ash91, RZ93, Aui04, AS05], star-free languages [Hen88, HRS10], piecewise testable languages [AZ97, ACZ08], locally testable languages [Ste01, Nog10], and locally threshold testable languages [Ste98, Ste01].

There are some important limitations with these approaches to the separation problem. First of all, the proofs for these results differ from variety to variety, and usually rely on involved (profinite) semigroup theory. On the one hand, this makes them less accessible, and on the other hand, this provides less insight in the combinatorial structure of the classes. For example, one may use topological properties, such as compactness, to obtain decidability of the problem. These arguments are not constructive, while more combinatorial approaches might give constructive proofs of decidability, even if these approaches may require more precise knowledge. Furthermore, one only obtains a yes/no answer to the separation problem. As mentioned above, if two languages can be separated by a language from a class $\mathcal{S}$, we are also interested in finding a witness of this fact, i.e. a so-called separator from the class $\mathcal{S}$. However, these approaches do not give any information about how to actually *construct* such a separator if it exists, nor do they give a witness of non-separability in the other case.

Contributions

Our goal in this project was to obtain simple proofs for the decidability of the separation problem for different subclasses of the regular languages, using combinatorial arguments rather than involved methods from profinite semigroup theory. We also wanted to obtain algorithms to decide the separation problem efficiently. Furthermore, rather than just having a yes/no answer to the separation problem, we were especially interested in finding descriptions of separating languages in case they exist, as well as witnesses of non-separability in the other case.

Contribution I: Developing a generic method

Our approach to the separation problem uses combinatorial arguments. The central idea is to provide a bound on a parameter that is pertinent to the class of separators (such as the quantifier rank for a class defined in terms of logic), and to show that the separation problem for the full class of separators reduces to the separation problem for the finite restriction of this class consisting of separator languages defined by a parameter that is smaller than this bound.

More precisely, for a class $\mathcal{S}$ of separators, we choose a parameter k, which is such that fixing this parameter gives a strictly smaller and finite subclass of languages (denoted by $\mathcal{S}[k]$), and which is such that increasing the value of k yields a more expressive class of languages. That is, for all $k \in \mathbb{N}$, $\mathcal{S}[k] \subsetneq \mathcal{S}[k+1]$ and $\mathcal{S} = \bigcup_{k\in\mathbb{N}} \mathcal{S}[k]$. Choices of parameters could be, for example, the length of the inspected prefixes, suffixes, factors or pieces, or the quantifier rank of a formula that defines the language. More in general, one could always take the size of the syntactic monoid of the language as a parameter.

The parameter k gives a means to stratify the class $\mathcal{S}$ according to the expressive power. And, since each subclass $\mathcal{S}[k]$ is finite, the separation problem for such a subclass is decidable: one can enumerate the languages and test all possible candidates.

Such a parameter defines a sequence of congruence relations on A^* in the following way. For $u, v \in A^*$, $u \sim_k v$ if and only if u and v are not separable by any language from $\mathcal{S}[k]$. Note that every language in $\mathcal{S}[k]$ is a finite union of $\sim_k$-congruence classes. Now, two input languages are $\mathcal{S}$-separable if and only if there exists $k \in \mathbb{N}$ such that no word from the first language is $\sim_k$-equivalent to a word from the second language. A priori, this means that all the infinitely many $\sim_k$-congruences should be checked to be able to conclude that two languages are *not* $\mathcal{S}$-separable.

However, by letting the $\sim_k$-congruences induce a relation on a monoid or an automaton (i.e. on a *finite* set), we work around this difficulty. We let a $\sim_k$-congruence induce a relation that will express that the monoid elements, or the pairs of states, determine languages that cannot be distinguished by $\mathcal{S}[k]$. We call this relation $\mathcal{S}[k]$-indistinguishability. We furthermore say that two monoid elements, or two pairs of states, are $\mathcal{S}$-indistinguishable if they are $\mathcal{S}[k]$-indistinguishable for all $k \in \mathbb{N}$. This relation precisely characterizes the pairs of monoid elements, or pairs of pairs of states, that determine languages that are not $\mathcal{S}$-separable. It thus corresponds to the pointlike sets of size 2 in the above terminology.

A drawback of the notion of $\mathcal{S}[k]$-indistinguishable, compared to the $\sim_k$-congruence on A^*, is that this relation is not a congruence anymore: it is not transitive. However, this drawback is compensated by its big advantage: its stabilizing behavior. Whereas the congruence relations $\sim_k$ on A^* keep getting more and more refined, this is not the case for the relation of $\mathcal{S}[k]$-indistinguishability. Since, for every k, $\mathcal{S}[k+1]$-indistinguishable pairs are also $\mathcal{S}[k]$-indistinguishable, and since these pairs are from a finite set, there will be some value κ of the parameter such that for every $k \geq \kappa$, we have that the $\mathcal{S}[\kappa]$-indistinguishable pairs are also $\mathcal{S}[k]$-indistinguishable. This gives that if two languages are $\mathcal{S}$-separable, they will already be $\mathcal{S}[\kappa]$-separable, for a κ depending on the monoid or automaton recognizing the languages.

While the existence of such a κ is immediate from the definitions, computing a bound on κ is a difficult problem. If we would establish such a bound, the $\mathcal{S}$-separation problem would reduce to the $\mathcal{S}[\kappa]$-separation problem. The class $\mathcal{S}[\kappa]$ consists of finitely many languages, thus this would already yield decidability of the $\mathcal{S}$-separation problem: one can use a brute force algorithm to test all of these languages. Furthermore, such a bound would imply that the saturation of L_1, with respect to $\sim_\kappa$, is a language from $\mathcal{S}$ that separates L_1 from L_2, in case these languages are $\mathcal{S}$-separable. We thus obtain a description of a potential separator.

Contribution II: Application to several classes

Using this approach, we were able to show that the separation problem for regular input languages is decidable for the following classes,

- piecewise testable languages,
- unambiguous (i.e. $\mathsf{FO}^2(<)$-recognizable) languages,
- locally testable languages,
- and locally threshold testable languages.

For each of these classes, we obtain a description of the complexity of a potential separator, via the bound on the parameter. This bound only depends on the input languages, and the parameters that we bound are natural parameters for the classes of separators. For the class of piecewise testable languages, we bound the size of the pieces that are inspected in a separating language. For the class of unambiguous languages, we provide a bound on the quantifier rank of an $\mathsf{FO}^2(<)$-formula defining the language. The same bound also works to bound the size of the unambiguous products that occur in a boolean combination defining a separator. For the classes of locally testable and locally threshold testable languages, we bound the size of the prefix, infixes and suffix that are inspected. For locally threshold testable languages, we furthermore bound the counting threshold.

If separation of two languages is not possible, we obtain a witness of non-separability as a pattern, specific to the class of separators, in the monoid or automaton recognizing the languages. This entails an algebraic property of the corresponding variety, which is called reducibility. This property provides information about the shape of the simplest elements that are present in the topological closures of two languages in the free pro-V semigroup. We have thus shown that the varieties corresponding to PT, $\mathsf{FO}^2(<)$, LT, and LTT have this property.

These patterns that witness non-separability are furthermore useful to bypass the brute-force algorithm to test separability that follows from the reduction to a fixed parameter (which enumerates and tests all the finitely many potential separators that are defined by a parameter that is smaller than the bound for that parameter). As was the case for decidable algebraic characterizations, translating from algebra to the level of automata yields better complexity results.

For the class of piecewise testable languages, we are able to decide separability in Ptime with respect to both the size of the automaton and the size of the alphabet. For the class of unambiguous languages, separability can be decided in Exptime. For the class of locally testable languages, deciding separability can be achieved in co-Nexptime, while we find a co-NP lower complexity bound. For locally threshold testable languages, we also have a co-NP lower bound, and we show that separability can be decided in 2-Expspace.

For all of these classes, our proofs only use elementary combinatorial techniques, in contrast to previous approaches to the separation problem that use involved algebraic and topologic techniques. We believe that our proofs provide more insight in the structure of these classes. Indeed, as explained above, our approach yields more than just the decidability of the separation problem: we also obtain descriptions of potential separators, witnesses of non-separability, and algorithms to decide separability.

We also have some results for the separation problem when input languages do not have to be regular. For the class of piecewise testable languages, we found an alternative combinatorial proof that identifies the indistinguishable pairs of states, without bounding the parameter. This alternative method gives a criterion for non-PT-separability that holds for *any* pair of input languages. It was very recently shown in [CM14] that this criterion is decidable for context-free languages.

For the classes of locally testable and locally threshold testable languages, we also consider the separation problem for context-free input languages, and show that this is undecidable. We show that separating context-free languages by locally threshold testable languages with a fixed size of factors, however, is decidable.

Organization of the dissertation

In **Chapter 1**, we briefly discuss some preliminaries about words, languages, semigroups and monoids, varieties, and logic on finite words. This chapter also serves to fix some notation that we will apply in the rest of the thesis.

We formally introduce the separation problem in **Chapter 2**. We also introduce the problem of computing 2-V-pointlike sets of a monoid, and the problem of testing whether the intersection of the closures of two regular languages in the free pro-V monoid is empty. It was shown in [Alm99] that these problems are equivalent to the separation problem for the class of V-recognizable languages. We provide a proof of these equivalences. In this chapter, we also give a general outline of the approach that we will take in Chapters 4, 5, and 6: we introduce the notion of $\mathcal{S}$-indistinguishable pairs of monoid elements and $\mathcal{S}$-indistinguishable pairs of pairs of states of an automaton, for a class $\mathcal{S}$ of regular languages. We show how these pairs capture the information relevant to separation, and indicate how we usually construct them. This chapter is concluded with an illustration of different approaches to the separation problem for two basic examples: the classes of alphabet-testable languages and prefix-testable languages.

Chapter 3 is a short chapter that discusses the separation problem for the class of **group languages**. For this class of languages, we can build on a rich history of research on very related problems. We therefore do not apply our approach with indistinguishable pairs in this chapter, but we show how decidability of the separation problem follows from the Ribes-Zalesskiĭ product theorem. A constructive proof of this theorem, provided in [AS05], can be used to obtain a separating group language, in case it exists.

In **Chapter 4**, we apply the approach with indistinguishable pairs of states to prove decidability of the separation problem for **piecewise testable languages**. We prove decidability of this problem with two methods. One method is based on Simon's Factorization Forest theorem, and the other method works by bounding the size of the pieces that are relevant for separability of the input languages. This bound furthermore yields a description of a separating language, if it exists. We find this bound by showing that words having the same pieces of this size must both fit into a certain template. Pumping arguments then allow us to see that words that have the same pieces of arbitrary length can be read between the pairs of states. We also obtain that two pairs of states are indistinguishable for this class if and only if between the first and second state of both pairs a pattern of a similar shape is present. This equivalence allows us to provide an algorithm that decides separability for this class in PTIME with respect to both the size of the automaton and the size of the alphabet.

Chapter 5 treats the separation problem for **unambiguous languages** (i.e. $\mathsf{FO}^2(<)$-recognizable languages). We show that this problem is decidable, which yields that the 2-pointlike sets for the variety DA are computable. To show the decidability of the separation problem, we provide a fixpoint algorithm that computes the $\mathsf{FO}^2(<)$-indistinguishable pairs of monoid elements. In the completeness proof of this algorithm, we find - using combinatorial arguments - a bound κ on the quantifier rank of $\mathsf{FO}^2(<)$-formulas that are relevant for separability of the input languages. We get a description of a potential $\mathsf{FO}^2(<)$-separator as a saturation of one of the two languages with respect to a congruence depending on $\mathsf{FO}^2(<)$ and κ.

Finally, we discuss the separation problem for the classes of **locally testable languages** and **locally threshold testable languages** in **Chapter 6**. In this chapter, we also work by bounding parameters using combinatorial arguments. For the class of locally threshold testable languages, there are two relevant parameters: the size of the factors, and the counting threshold. We first solve the separation problem for a fixed counting threshold (in particular, this solves the problem for the class of locally testable languages). In this case, we obtain a bound on the size of the factors, of which we show that it still works for the full class of locally threshold testable languages. We then provide a bound on the counting threshold. These bounds provide a description of a potential separator. We also exhibit patterns on the recognizing monoid and the recognizing automaton that are present if and only if the languages are not separable. Furthermore, upper and lower complexity bounds for the separation problems for these classes are discussed. We conclude the chapter with results on the separation problem for locally testable and locally threshold testable languages on *context-free* input languages. We show that these problems are undecidable, but, surprisingly, separating context-free languages by locally threshold testable languages with a fixed size of factors is decidable.

Chapter 1

Preliminaries

1.1 Words and languages 11
1.2 Automata 12
1.3 Recognition by semigroups and monoids 13
1.4 Varieties and free pro-V semigroups 14
1.4.1 Identities 14
1.5 Logic on words 15
1.5.1 Different fragments of FO(<) 16

In this chapter, we briefly present some preliminary notions and fix some notation that we use in the rest of the thesis. We assume that the reader is familiar with the basics of the theory of automata, regular languages and semigroups. We refer to [Alm94, Str94, Pin97, Pin11], for an introduction to this theory.

1.1 Words and languages

An *alphabet* is a finite set. For an alphabet A, the *free monoid* over A, denoted by A^*, is the set of all words endowed with the usual concatenation operation. The concatenation of two words $u, v \in A^*$ is denoted by $u \cdot v$, or simply by uv. The empty word is denoted by ε. The *free semigroup* over A is the set of all nonempty words, and is denoted by A^+. For a word $w \in A^*$, the smallest subset $B \subseteq A$ such that $w \in B^*$ is called the *alphabet* of w and is denoted by $\mathsf{alph}(w)$. The number of letters in a word w is the *size*, or *length*, of w, denoted by $|w|$.

An *infix*, or *factor*, of a word $w \in A^*$ is a word w' such that $w = uw'v$ for some $u, v \in A^*$. In this case, we say that w' is a *prefix* (resp. a *suffix*) of w if $u = \varepsilon$ (resp. if $v = \varepsilon$). A word w' is a *piece*, or *scattered subword*, of a word w, if there exist letters $a_1, \ldots, a_k \in A$ such that $w' = a_1 \cdots a_k$ and $w = w_0 a_1 w_1 \cdots a_k w_k$, for some $w_0, \ldots, w_k \in A^*$.

A language over A is a subset of the free monoid A^*. We extend the definition of the concatenation operation to languages, by defining $L \cdot L' = \{w \cdot w' \mid w \in L, w' \in L'\}$, and we use the Kleene star operation $L^* = \bigcup_{n \in \mathbb{N}} L^n$.

The collection of *regular* languages over an alphabet A is defined as the smallest collection of languages over A that satisfies the following properties.

- The empty set $\varnothing$ is regular,
- for every $a \in A$, the set $\{a\}$ is regular,
- if L and L' are regular, then $L \cup L'$, $L \cdot L'$, and L^* are also regular.

1.2 Automata

A *nondeterministic finite automaton* (NFA) over an alphabet A is denoted by a tuple $\mathcal{A} = (A, Q, \delta)$, where Q is the set of states and $\delta \subseteq Q \times A \times Q$ is the transition relation. For a *deterministic finite automaton* (DFA), it is furthermore required that δ is a partial function from $Q \times A$ to Q. That is, for every $q \in Q$ and $a \in A$, there is at most one state q' such that $(q, a, q') \in \delta$. We shall explain below why we use these definitions, which are not the standard ones. The *size* $|\mathcal{A}|$ of an automaton $\mathcal{A}$ is its number of states plus its number of transitions.

Given a word $u \in A^*$, a subset B of A and two states p, q of $\mathcal{A}$, we denote

- a path from state p to state q labeled by the word u by $p \xrightarrow{u} q$,
- a path from p to q of which all transitions are labeled over B by $p \xrightarrow{\subseteq B} q$,
- a path from p to q labeled by a word whose alphabet is exactly B by $p \xrightarrow{=B} q$.

We also write $(p, u, q) \in \delta^*$ to denote that there is a path $p \xrightarrow{u} q$ in $\mathcal{A}$.

Given a language $L \subseteq A^*$, we say that L is *accepted*, or *recognized*, by an NFA $\mathcal{A} = (A, Q, \delta)$ if there exist sets $I, F \subseteq Q$ such that

$$L = \{w \mid \exists q_I \in I \text{ and } \exists q_F \in F \text{ such that } (q_I, w, q_F) \in \delta^*\}.$$

In this case, we call I the set of *initial states* and F the set of *final states* for L, and we also say that $I \times F$ *determines* the language L, and we denote this language by $L(\mathcal{A}, I, F)$. For a language L to be accepted by a DFA $\mathcal{A}$, we ask furthermore that the set I consists of only one state. We recall Kleene's theorem that says that a language $L \subseteq A^*$ is regular if and only if it is recognized by a finite automaton.

Note that these definitions are not the standard definitions of NFAs and DFAs. Usually, the sets of initial and final states are fixed in the definition of the automaton. However, for the study of separation problems, which form the subject of this thesis, it will be convenient to have a single automaton that accepts the two input languages. Note that from two given regular languages L_1 and L_2 over A, recognized by NFAs $\mathcal{A}_1 = (A, Q_1, \delta_1)$ resp. $\mathcal{A}_2 = (A, Q_2, \delta_2)$, one can build an automaton that accepts both L_1 and L_2, viz. the automaton $\mathcal{A} = (A, Q_1 \uplus Q_2, \delta_1 \uplus \delta_2)$ of size $|\mathcal{A}_1| + |\mathcal{A}_2|$.

1.3 Recognition by semigroups and monoids

A *semigroup* is a set of elements equipped with an associative binary operation. We denote this operation multiplicatively. A semigroup S is a *monoid* if it has an identity element, that is, if there is $1 \in S$ such that for all $s \in S$, $1 \cdot s = s \cdot 1 = s$. An element s is *idempotent* if $s^2 = s$. For a semigroup $(S, \cdot)$ (that we will simply denote by S), we denote the set of its idempotent elements by $E(S)$. In a finite semigroup, every element s has an idempotent power, which is denoted by s^ω. For example, if $|S| = n$, then one can verify that for every $s \in S$, the element $s^{n!}$ is idempotent.

A mapping $\varphi : S \to T$ between semigroups is a *semigroup morphism* if for all $s, s' \in S$, it holds that $\varphi(s \cdot s') = \varphi(s) \cdot \varphi(s')$. If S and T are monoids, and it holds furthermore that $\varphi(1_S) = 1_T$, then φ is a *monoid morphism.* If it is clear from the context which kind of morphism is meant, we also simply write *morphism.*

A language L over A is *recognized* by a semigroup S, if there is a morphism $\varphi : A^+ \to S$ and a subset $P \subseteq S$, such that $L = \varphi^{-1}(P)$. In this case, we also say that L is recognized by φ. Given a morphism $\varphi : A^+ \to S$, we say that an element $s \in S$ *determines* the language $\varphi^{-1}(s)$.

Similarly, a language L over A is *recognized* by a monoid M, if there is a morphism $\varphi : A^* \to M$ and a subset $P \subseteq M$, such that $L = \varphi^{-1}(P)$. And, given a morphism $\varphi : A^* \to M$, we also say that an element $m \in M$ *determines* the language $\varphi^{-1}(m)$.

For example, the language A^*aA^* is recognized by the finite monoid $U_1 = \{\mathbf{0}, \mathbf{1}\}$, with multiplication given by $\mathbf{0} = \mathbf{0} \cdot \mathbf{0} = \mathbf{0} \cdot \mathbf{1} = \mathbf{1} \cdot \mathbf{0}$ and $\mathbf{1} = \mathbf{1} \cdot \mathbf{1}$, via the morphism φ that sends a to 0 and sends b to 1. Indeed, $A^*aA^* = \varphi^{-1}(0)$.

One way to find such a recognizing monoid for a language L (and in fact to find the minimal one), is to take the quotient of A^* with respect to the following congruence relation.

$$u \sim_L v \quad \text{if and only if} \quad \forall w, w' \in A^*.\ wuw' \in L \ \Leftrightarrow\ wvw' \in L.$$

This congruence is called the *syntactic congruence* of L, and the corresponding quotient is called the *syntactic monoid* of L. One can similarly define the syntactic semigroup of L.

It is well known and easy to show that if a language is regular, then its syntactic monoid is finite. Another way to construct a recognizing monoid out of an NFA is to construct its transition monoid. For a DFA, this is the monoid generated by the partial transformations on Q induced by the letters of A. For an NFA, it can be represented as the monoid generated by the boolean matrices of order $|Q| \times |Q|$, induced by the letters of A in the following way: the coordinate (p, q) of the matrix induced by $a \in A$ is 1 if and only if $(p, a, q) \in \delta$.

Conversely, every language that is recognized by a morphism $\varphi : A^* \to M$, is recognized by the DFA (A, M, δ), with $\delta(\_, a) : m \mapsto m \cdot \varphi(a)$. To sum up, a language is accepted by an NFA if and only if it is recognized by a finite monoid.

1.4 Varieties and free pro-V semigroups

Syntactic semigroups form an important connection between regular languages and finite semigroups. The concept of *variety* serves to classify regular languages according to algebraic properties of their syntactic semigroups. Here we introduce a few notions related to the theory of varieties, but we refer to the text books mentioned above for many important properties and results that we do not discuss here.

A *variety* of finite semigroups (also called *pseudovariety* of (finite) semigroups) is a collection of semigroups that is closed under taking subsemigroups, homomorphic images and finite direct products. If a language L is recognized by a semigroup from a variety V, we say that L is V-recognizable.

Let A be a finite alphabet and let V be a variety. A semigroup S *separates* $u, v \in A^+$ if there exists a morphism $\varphi : A^+ \to S$ such that $\varphi(u) \neq \varphi(v)$. Given $u, v \in A^+$, we let

$$r_{\mathsf{V}}(u,v) = \min\bigl\{|S| \mid S \in \mathsf{V} \text{ and } S \text{ separates } u \text{ and } v\bigr\} \in \mathbb{N} \cup \{\infty\},$$

with $\min \varnothing = \infty$, and we define

$$d_{\mathsf{V}}(u,v) = 2^{-r_{\mathsf{V}}(u,v)},$$

with $2^{-\infty} = 0$. For two given words, there is not necessarily a semigroup in V that is able to separate them. Thus, d_{V} does not necessarily define a metric on A^+. We therefore consider the congruence relation $\sim_{\mathsf{V}}$, defined by

$$u \sim_{\mathsf{V}} v \;\Leftrightarrow\; d_{\mathsf{V}}(u,v) = 0.$$

One can verify that d_{V} is a metric on $A^+/{\sim_{\mathsf{V}}}$.

We endow every finite semigroup with the discrete topology. A sequence $(u_n)_n$ is a Cauchy sequence for this metric if and only if for every morphism $\varphi : A^+ \to S$, the sequence $(\varphi(u_n))_n$ is eventually constant. The completion of the metric space $(A^*/{\sim_{\mathsf{V}}}, d_{\mathsf{V}})$ is denoted by $\widehat{F}_{\mathsf{V}}(A)$ and is called the *free pro-*V *semigroup*.

The semigroup operation on $\widehat{F}_{\mathsf{V}}(A)$ is given by pointwise multiplication of classes of Cauchy sequences. This transfers the semigroup structure of A^+ to $\widehat{F}_{\mathsf{V}}(A)$, on which the multiplication is continuous. This will be discussed in more detail in Section 2.2.2.

The semigroup $\widehat{F}_{\mathsf{V}}(A)$ is profinite, since one can show that the topology on $\widehat{F}_{\mathsf{V}}(A)$ is the coarsest topology which makes every morphism from A^+ to a finite discrete semigroup $S \in \mathsf{V}$ continuous.

One can furthermore show that $\widehat{F}_{\mathsf{V}}(A)$ satisfies the following universal property. For every mapping $\varphi : A \to S \in \mathsf{V}$, there is a unique uniformly continuous morphism $\hat{\varphi} : \widehat{F}_{\mathsf{V}}(A) \to S$ that extends φ.

1.4.1 Identities

The variety of all finite semigroups is denoted by S. The free pro-S semigroup is also called the *free profinite semigroup* and we denote it by $\widehat{F}(A)$. Its elements are called profinite words. Let $u, v \in \widehat{F}(A)$. We say that the *identity* $u = v$ is *satisfied* by a finite semigroup S if and only if, for every continuous morphism $\varphi : \widehat{F}(A) \to S$, it holds that $\varphi(u) = \varphi(v)$. Reiterman's theorem states that every variety of finite monoids can be defined by a set of such identities [Rei82].

For $u \in \widehat{F}(A)$, the element $\lim_{n\to\infty} u^{n!}$ is denoted by u^ω. This is an idempotent element of $\widehat{F}(A)$. Recall that in the theory of finite semigroups, s^ω denotes the idempotent power of an element $s \in S$. For $\varphi : A^+ \to S$ and $u \in A^+$, it holds that $\hat{\varphi}(u^\omega) = \varphi(u)^\omega$, which justifies the notation for $\lim_{n\to\infty} u^{n!}$.

This allows us to describe the varieties that we study in this thesis in a succinct way, by the identities that are satisfied by their semigroups. For example, in Section 2.3.1, we study the variety of semilattices. This variety consists of semigroups that are idempotent and commutative, which can be expressed by the identities $x^2 = x$ and $xy = yx$. In this case, no profinite words are needed to define the variety. The languages that we study in Chapter 4, on the other hand, are recognized by semigroups that satisfy the identities $x^\omega = x^{\omega+1}$ and $(xy)^\omega = (yx)^\omega$, or equivalently, that satisfy $y(xy)^\omega = (xy)^\omega = (xy)^\omega x$.

1.5 Logic on words

We consider first-order logic (denoted by $\mathsf{FO}(<)$) interpreted on words. In this formalism, variables are interpreted as positions in words, and one may use the following two types of predicates.

- The binary predicate $<$, where $x < y$ means that position x occurs before position y,
- for each letter a, a unary predicate $a(\_)$, where $a(x)$ means that position x carries the letter a.

An FO-formula *defines* the language of words for which the formula holds. For instance, the formula

$$\exists x \exists y.\ \Big(x < y \wedge a(x) \wedge b(y) \wedge \forall z.\ \big((x < z \wedge z < y) \Rightarrow c(z)\big)\Big) \tag{1.1}$$

defines the language $A^*ac^*bA^*$.

In this thesis, we consider different fragments of first-order logic. If a language can be defined by a formula from a fragment $\mathcal{F}$, we also say that the language is *$\mathcal{F}$-definable.*

A fragment $\mathcal{F}$ gives rise to an equivalence relation on words in the following way. For $u, v \in A^*$,

$$u \sim_{\mathcal{F}} v \ \Leftrightarrow\ u \text{ and } v \text{ satisfy exactly the same formulas of } \mathcal{F}.$$

The *quantifier rank*, or simply *rank*, of a formula is the maximal number of nested quantifiers of a formula. For example, the formula from (1.1) has rank 3.

In this thesis, we will often want to 'stratify' a class of languages according to its expressive power. The rank of a formula gives a means to obtain such a stratification for a class of languages defined by a logic. By $\mathcal{L}[k]$, we denote the restriction of the logic $\mathcal{L}$ that consists of only those formulas that have rank $\leq k$. Of course, increasing k yields a more expressive logic, and by definition, $\mathcal{L} = \bigcup_{k\in\mathbb{N}} \mathcal{L}[k]$. This stratification induces, for a logic $\mathcal{L}$, a sequence of equivalence relations $\sim_{\mathcal{L}[k]}$ on words. It is clear that $\sim_{\mathcal{L}} = \bigcap_{k\in\mathbb{N}} \sim_{\mathcal{L}[k]}$, and for all $k \in \mathbb{N}$, $\sim_{\mathcal{L}[k+1]}\subseteq \sim_{\mathcal{L}[k]}$.

1.5.1 Different fragments of $\mathsf{FO}(<)$

Let us briefly introduce the different fragments of $\mathsf{FO}(<)$ that we study in this thesis. The fragments $\mathsf{FO}^1(<)$ (see Section 2.3.1) and $\mathsf{FO}^2(<)$ (see Chapter 5) are obtained by restricting the number of (reusable) variable names that may be used in a formula. In the fragment $\mathsf{FO}^1(<)$ only one variable name is allowed. For example, the formula

$$\exists x.\ a(x)\ \wedge\ \exists x.\ b(x) \wedge\ \forall x. \neg c(x)$$

is in $\mathsf{FO}^1(<)$. For $A = \{a, b, c\}$, this formula defines the language of words whose alphabet is exactly $\{a, b\}$. It is not hard to see that this logic can only express such alphabetical conditions, and boolean combinations of these. In $\mathsf{FO}^2(<)$, two variable names may be used in a formula. An example of an $\mathsf{FO}^2(<)$-formula is the following formula,

$$\exists x\, \exists y.\ \Big(b(x) \wedge a(y) \wedge \forall x.\ \big(\neg(y < x)\big)\Big),$$

which defines the language A^*bA^*a. Note that we reused the variable x. Note that is not clear right away whether the language $A^*ac^*bA^*$, defined by (1.1), can be expressed by an $\mathsf{FO}^2(<)$-formula. In Chapter 5, we will see a characterization of the $\mathsf{FO}^2(<)$-definable languages, from which it will be easy to see that this is not the case.

In Chapter 4, we consider the fragment $\mathcal{B}\Sigma_1(<)$, which is the fragment of formulas that are boolean combinations of formulas that use only existential quantifiers. With these formulas, the presence or absence of pieces in words can be expressed. An example of a $\mathcal{B}\Sigma_1(<)$-definable language is the language of all words containing the piece aab, that is, the language $A^*aA^*aA^*bA^*$. This language is recognized by the following $\mathcal{B}\Sigma_1(<)$-formula,

$$\exists x\, \exists y\, \exists z.\ \big(x < y \wedge y < z \wedge a(x) \wedge a(y) \wedge b(z)\big).$$

This language is also defined by the following $\mathsf{FO}^2(<)$-formula,

$$\exists x\, \exists y.\ \Big(x < y \wedge a(x) \wedge a(y) \wedge \exists x.\ \big(y < x \wedge b(x)\big)\Big).$$

In fact, we will see in Chapter 5 that every $\mathcal{B}\Sigma_1(<)$-formula is equivalent to an $\mathsf{FO}^2(<)$-formula.

In Chapter 6, we consider yet another fragment, denoted by $\mathsf{FO}(=, +1)$. This is a variant of first-order logic where the binary predicate $<$ is not allowed, but where we have a unary predicate $S(\_)$ that expresses the successor relation. Here, $S(x) = y$ means that position y is the first position to the right of position x. These formulas can be used to detect prefixes, infixes and suffixes of words. For example, the language aA^*bbA^* is defined by the $\mathsf{FO}(=, +1)$-formula

$$\exists x\, \forall y.\ \big(\neg(S(y) = x) \wedge a(x)\big)\ \wedge\ \exists x\, \exists y.\ \big(S(x) = y \wedge b(x) \wedge b(y)\big).$$

Chapter 2

Introduction to the separation problem

2.1 The separation problem 18

2.2 Different points of view on the separation problem 21

2.2.1 Algebraic view: 2-pointlike sets 22

2.2.2 Topological view: closures in the free pro-V semigroup 24

2.2.3 Combinatorial view: indistinguishable pairs 26

2.3 Basic examples 29

2.3.1 Example I: SI 29

Complexity of separation by SI-recognizable languages 30

2.3.2 Example II: K 32

Computing the intersection of the closures in $\widehat{F}_{\mathsf{K}}(A)$ 32

Computing K-indistinguishable pairs of states 33

In this chapter, we introduce the separation problem and discuss different views on this problem. The separation problem for a class $\mathcal{S}$ of languages is to decide, given two regular input languages, whether there exists a language from the class $\mathcal{S}$ that contains the first language and is disjoint from the second language.

The separation problem for a class of languages generalizes the membership problem for this class, and provides a deeper understanding of the class. Furthermore, decidability of the separation problem is better preserved under operations than decidability of the membership problem is. Because of these properties, the separation problem for classes of languages has emerged in different contexts: in an algebraic guise in the study of so-called pointlike sets with respect to a variety V, and as a topological separation problem in the free pro-V semigroup.

In Section 2.1, we define the separation problem and discuss the relation with the membership problem. We continue the following sections with a discussion of results from the literature. For varieties of languages, solving the separation problem is equivalent to computing certain algebraic objects. Section 2.2.1 presents this algebraic view on the separation problem. In

Section 2.2.2, it is shown that deciding the separation problem for varieties of languages is also equivalent to computing an intersection of certain topological closures. Our approach to solving the problem is based on the algebraic objects and is presented in Section 2.2.3. This is the approach that we will apply in Chapters 4, 5, and 6 to different classes of languages. Finally, in Section 2.3, we present some basic examples to illustrate different approaches to the separation problem.

2.1 The separation problem

Deciding whether two regular languages, given as regular expressions, are disjoint can be done by building automata from the expressions, and computing the intersection of these automata. However, sometimes it is possible to see that two languages are disjoint by extending the first language to a simpler language, of which it is easier to see that it is disjoint from the second language. For example, the languages $(abb(ab)^*b)^*$ and $(ab^*aa)^*b$ are disjoint, because $(abb(ab)^*b)^* \subseteq \varepsilon \cup A^*bb$, while $(ab^*aa)^*b \cap (\varepsilon \cup A^*bb) = \varnothing$. One can thus already conclude that these regular languages are disjoint by only inspecting their respective sets of suffixes of size 2.

The class of regular languages for which membership of a word in the language only depends on its suffix of a certain size is a very restricted class compared to the full class of regular languages, yet it is still able to witness that $(abb(ab)^*b)^*$ and $(ab^*aa)^*b$ are disjoint. Of course, there are also disjoint regular languages which are not sufficiently different, from the point of view of this class, to be witnessed as disjoint. For example, the languages aa^* and ba^* contain, for every $n \in N$, words that have the same suffix of length n, and thus it is not possible to find a language in this class that witnesses that they are disjoint.

We will see that the problem of deciding whether two languages are perceived as disjoint by a given class of languages is a natural generalization of the membership problem for this class. Let us first introduce formally what is meant by being perceived as disjoint.

Definition 2.1. Given languages L, L_1, L_2, we say that L *separates* L_1 from L_2 if

$$L_1 \subseteq L \text{ and } L_2 \cap L = \varnothing. \tag{2.1}$$

Given a class $\mathcal{S}$ of languages, we say that the pair (L_1, L_2) is *$\mathcal{S}$-separable* if there exists a language $L \in \mathcal{S}$ that separates L_1 from L_2. The language L is then called an *$\mathcal{S}$-separator*, or simply *separator*. Note that (2.1) is equivalent to

$$L_1 \cap (A^* \setminus L) = \varnothing \text{ and } L_2 \subseteq A^* \setminus L.$$

Therefore, L separates L_1 from L_2 if and only if $A^* \setminus L$ separates L_2 from L_1. Hence, for a class $\mathcal{S}$ of separators closed under complement, one observes that (L_1, L_2) is $\mathcal{S}$-separable if and only if (L_2, L_1) is, in which case we simply say that L_1 and L_2 are $\mathcal{S}$-separable.

For a given class of languages and for two given languages, we are interested in knowing whether the class is able to separate these languages.

Definition 2.2. The *separation problem* for a class $\mathcal{S}$ of languages, also called the $\mathcal{S}$-separation problem, can be formulated as follows.

Input: Two languages L_1 and L_2.

Question: Are L_1 and L_2 separable by a language from $\mathcal{S}$?

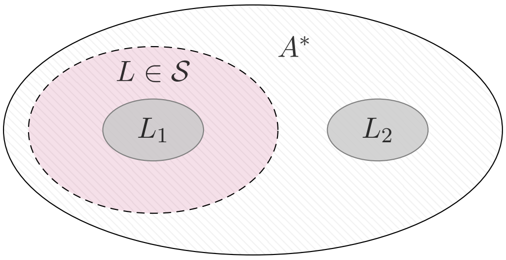


Figure 2.1: L_1 and L_2 are $\mathcal{S}$-separable.

Usually, we only consider the separation problem for input languages that are regular. For the class $\mathcal{S}$, we will often consider a class of languages that are recognized by some variety V. If two languages are $\mathcal{S}$-separable with respect to this class, we will say that they are V-*separable*. Similarly, we call two languages L-*separable* if there is a language defined by a formula from the logic L that separates them.

A positive answer to the separation problem means that L_1 and L_2 are sufficiently different, from the point of view of $\mathcal{S}$, to be perceived as disjoint. This means that $\mathcal{S}$ is able to discriminate between the two languages. The separation problem tests the *discriminative power* of $\mathcal{S}$. Note that a language L belongs to the class $\mathcal{S}$ if and only if L is $\mathcal{S}$-separable from its complement. We usually consider the separation problem for input languages that are regular. Since the class of regular languages is closed under complement, the separation problem subsumes the classical membership problem, which amounts to deciding whether a given language belongs to a given class. Indeed, the only language that can separate L from $A^*\backslash L$ is L itself. Thus, L is in $\mathcal{S}$ if and only if $(L, A^*\backslash L)$ are $\mathcal{S}$-separable. This gives the following fact.

Fact 2.3. The membership problem for the class $\mathcal{S}$ reduces to the separation problem, for regular input languages, for the class $\mathcal{S}$.

The membership problem tests the *expressive power* of a class of languages that is described by combinatorial or logical means. It is considered to be one of the main ways to understand such a class. By the above, the discriminative power of a class is more informative than the expressive power of a class. We study the separation problem because it provides a deeper understanding of a class of languages than the classical membership problem does.

In this thesis, we will see that most natural classes of languages have a decidable separation problem. Furthermore, it is already known that the separation problem has more robust properties than the membership problem: in [Ste01], an operation on classes of languages is described that preserves decidability of the separation problem, while it is shown in [Aui10] that decidability of the membership problem is not preserved under this operation.

Let us illustrate the notion of separability on small examples.

Example 2.4. The variety SI defines the class of languages that can be described in terms of their alphabet: it consists of all languages that are Boolean combinations of languages of the form B^*, for finite alphabets B. In Section 2.3.1, we will discuss this class in more detail. It is easy to see that the languages $(a(b \cup c))^+$ and $(ba)^+$ are not SI-separable, since both languages contain words whose alphabet is $\{a, b\}$. On the other hand, the languages $(ab)^+$ and $(ac)^+$ are SI-separable, since the SI-recognizable language $\{w \in A^* \mid \mathsf{alph}(w) = \{a, b\}\}$ separates them.

Example 2.5. The languages $(ab)^+$ and ba^* are $\mathsf{FO}(<)$-separable, since the language defined by $\exists x.\ \exists y.\ x < y \wedge a(x) \wedge b(y)$ contains $(ab)^+$ and is disjoint from ba^*. The languages $(aa)^*$ and $(aa)^*a$, on the other hand, are not separable by any language defined by a first-order formula. This follows from [Str94, Theorem IV.2.1], where an Ehrenfeucht-Fraïssé argument is used to show that the set of words of even length is not definable in first-order logic.

In [SW76, Hun82], the separation problem was studied for the class of context-free languages as input languages, and the class of regular languages as separators. In [SW76], the following theorem is proved by a reduction from the halting problem on Turing machines to this separation problem. In Chapter 6, we will see that this proof can be adapted to prove undecidability of separability of context-free languages for other classes of regular separators, satisfying some conditions, as well. In Theorem 6.51, we give an adaptation of this proof.

Theorem 2.6 ([SW76, Theorem 4.6])**.** *Separability of context-free languages by regular languages is undecidable.*

We conclude this section with a lemma that gives a sufficient condition to conclude that two languages are $\mathcal{S}$-separable, when the class $\mathcal{S}$ is closed under finite union and finite intersection. In this case, it suffices to decompose the two languages in a finite way such that the components of the different languages are pairwise $\mathcal{S}$-separable. As varieties of languages are closed under finite union and finite intersection, this result applies in particular to varieties of languages.

Lemma 2.7 ([Pin09, Lemma 2.1])**.** *Let $(K_i)_{i \in I}$ and $(L_j)_{j \in J}$ be two finite families of languages. Let $\mathcal{S}$ be a class of languages closed under finite union and finite intersection. If each pair K_i and L_j is $\mathcal{S}$-separable, then $\bigcup_{i \in I} K_i$ and $\bigcup_{j \in J} L_j$ are $\mathcal{S}$-separable.*

Proof. Let $T_{i,j}$ be an $\mathcal{S}$-recognizable language such that $K_i \subseteq T_{i,j}$ and $L_j \cap T_{i,j} = \varnothing$. We claim that the language $T := \bigcup_{i \in I} \bigcap_{j \in J} T_{i,j}$ separates $K := \bigcup_{i \in I} K_i$ and $L := \bigcup_{j \in J} L_j$. Note that T is $\mathcal{S}$-recognizable since $\mathcal{S}$ is closed under finite union and finite intersection.

For every $i \in I$ and $j \in J$, by definition, $K_i \subseteq T_{i,j}$. Thus, $K_i \subseteq \bigcap_{j \in J} T_{i,j}$, and $K \subseteq \bigcup_{i \in I} \bigcap_{j \in J} T_{i,j}$. On the other hand, to show that $L \cap T = \varnothing$, it suffices to show that for every $k \in J$, $L_k \cap T = \varnothing$. Fix $i \in I$. Note that $L_k \cap \bigcap_{j \in J} T_{i,j} \subseteq L_k \cap T_{i,k} = \varnothing$, by definition. Now,

$$L_k \cap T = L_k \cap \bigcup_{i \in I} \bigcap_{j \in J} T_{i,j} = \bigcup_{i \in I} \big(L_k \cap \bigcap_{j \in J} T_{i,j}\big) = \varnothing.$$

It follows that T separates K and L. □

2.2 Different points of view on the separation problem

For classes of languages that correspond to varieties, two different approaches (algebraic and topological) to the separation problem are known from the literature. In [Alm99], it is shown that these approaches are equivalent. If the separation problem is decidable for a variety V, these approaches provide a yes/no answer to the question whether two languages are V-separable. However, they do not give any information about how to actually *construct* a V-recognizable language that separates the two languages in case they are V-separable.

The algebraic approach is based on objects called pointlike sets. A reason for introducing these pointlike sets was to obtain a property that is better preserved under operations, than decidability of the membership problem is. For a small number of specific varieties, these pointlike sets have been shown to be computable, yielding the decidability of the separation problem for that variety. In particular, this has been shown for the following varieties.

1. Languages recognized by a finite group [Ash91, RZ93, Aui04, AS05],
2. Star-free (that is, $\mathsf{FO}(<)$-definable) languages [Hen88, HRS10],
3. Piecewise testable (that is, $\mathcal{B}\Sigma_1(<)$-definable) languages [AZ97, ACZ08],
4. Languages whose syntactic semigroups are $\mathcal{R}$-trivial, that is, languages such that all cycles in the graph of the minimal automaton visit just one state [ACZ08],
5. Languages for which membership of a word in the language can be tested by inspecting the prefix resp. suffix up to some length (folklore, see [Alm94, Section 3.7]),
6. Locally testable languages, that is, languages for which membership of a word in the language can be tested by inspecting the prefix, suffix and factors up to some length [Ste01, Nog10],
7. Locally threshold testable languages, that is, languages for which membership of a word in the language can be tested by inspecting the prefix, suffix and the factors, counted with a certain threshold, up to some length [Ste98, Ste01].

Our view on the separation problem is described in Section 2.2.3. For a class $\mathcal{S}$ of separators that forms a variety, this view coincides with the algebraic view, that is, we calculate the pointlike sets of size 2.

For a class $\mathcal{S}$ of separators, we choose a parameter k that defines a congruence relation $\sim_k$ on A^* in such a way that increasing k yields a more and more refined congruence relation, and such that every $\mathcal{S}$-definable language is a union of $\sim_k$-equivalence classes for some k. Then, two languages are $\mathcal{S}$-separable if and only if there exists $k \in \mathbb{N}$ for which there is no word in the first language that is $\sim_k$-equivalent to a word from the second language.

For example, if $\mathcal{S}$ is defined in terms of a logical fragment, the quantifier rank of a formula provides such a parameter k. In this case, two words are said to be $\sim_k$-equivalent if and only if they satisfy exactly the same formulas of rank k. Increasing this rank gives more and more power to discriminate between languages, and thus more and more languages become distinguishable.

Another example of a parameter that defines such a sequence of congruence relations on A^*

is the size of the semigroup. That is, if $\mathcal{S}$ is the class of V-recognizable languages, then two words are $\sim_k$-equivalent if and only if, for each morphism into a semigroup of V of size up to k, their images are the same. In other words, $u \sim_k v \Leftrightarrow d_{\mathsf{V}}(u, v) < 2^{-k}$, for $d_{\mathsf{V}}(u, v)$ as defined in Section 1.4.

Deciding whether two languages are $\mathcal{S}$-separable amounts to knowing whether there exists $k \in \mathbb{N}$ for which there is no word in the first language that is $\sim_k$-equivalent to a word from the second language. In principle, this means that we would have to know the behavior of all of the $\sim_k$-congruences to conclude that two languages are not $\mathcal{S}$-separable. However, as we will explain in Section 2.2.3, when proving decidability of the separation problem for a certain class $\mathcal{S}$, we usually obtain a bound on the parameter k, such that it suffices to consider the congruence relation up to that bound. Besides showing the decidability, we then also obtain a description of a separator if it exists.

2.2.1 Algebraic view: 2-pointlike sets

We now discuss the algebraic counterpart of separation by a variety of languages, as presented by Almeida in [Alm99]. After the introduction of the relevant notions, we state the result about the connection with algebra in Theorem 2.11.

Definition 2.8. A *relational morphism* between monoids M and N is a relation $\tau \subseteq M \times N$, also written as $\tau : M \to N$, such that

i. the projection onto the first coordinate is surjective, i.e. for every $m \in M$, $\tau(m) \neq \varnothing$,

ii. τ is a submonoid of $M \times N$, i.e. for every $m_1, m_2 \in M$, $\tau(m_1)\tau(m_2) \subseteq \tau(m_1 m_2)$ and $1_N \in \tau(1_M)$.

A relational morphism between semigroups S and T is defined similarly, by replacing condition ii by the following condition:

ii'. τ is a subsemigroup of $S \times T$, i.e. for every $s_1, s_2 \in S$, $\tau(s_1)\tau(s_2) \subseteq \tau(s_1 s_2)$.

Definition 2.9. A subset S of a monoid M is called V-*pointlike* if for every monoid $N \in \mathsf{V}$ and every relational morphism $\tau : M \to N$, it holds that there exists $n \in N$ such that for all $s \in S$, $n \in \tau(s)$. If furthermore $|S| = k$, S is said to be k-V-*pointlike*.

In other words, a subset S of a monoid M is V-pointlike if the elements of S are not perceived as disjoint sets by any member of V. It is thus not surprising that there is a link between the pointlike sets with respect to a variety V and the separation problem for V: both concepts deal with the ability of V to witness differences in objects outside of V itself. This link is made precise in Theorem 2.11.

Note that to decide, for a given variety V, whether a subset of a monoid is V-pointlike, in principle one has to check an infinite number of relational morphisms. It is thus a priori not clear whether this is a decidable property.

Furthermore, it is important to note that when each pair of elements of a set S is 2-V-pointlike, it does *not* automatically hold that S is an $|S|$-V-pointlike set. This is because of the following fact, of which an illustration can be found in Example 2.18.

Fact 2.10. The relation of being in a 2-V-pointlike set is not transitive.

The following theorem shows that the 2-pointlike sets with respect to a given variety are pertinent to the separation problem with respect to this variety. Here, we follow the proof of [Pin09, Theorem 2.2].

Theorem 2.11 ([Alm99, Lemma 3.2])**.** *Let L_1, L_2 be regular languages, recognized by a monoid M via a surjective morphism φ. Let $P, Q \subseteq M$ be such that $L_1 = \varphi^{-1}(P)$ and $L_2 = \varphi^{-1}(Q)$. Then, L_1 and L_2 are V-separable if and only if for every $p \in P$ and every $q \in Q$, the set $\{p, q\}$ is not a 2-V-pointlike set.*

Proof. Since $L_1 = \bigcup_{p\in P} \varphi^{-1}(p)$ and $L_2 = \bigcup_{q\in Q} \varphi^{-1}(q)$, it follows from Lemma 2.7 that L_1 and L_2 are V-separable if and only if for every $p \in P, q \in Q$, the languages $\varphi^{-1}(p)$ and $\varphi^{-1}(q)$ are V-separable. Let $p \in P$ and $q \in Q$. We will show that $\varphi^{-1}(p)$ and $\varphi^{-1}(q)$ are V-separable if and only if $\{p, q\}$ is not a V-pointlike set. Suppose that $\varphi^{-1}(p)$ and $\varphi^{-1}(q)$ are V-separable. Then there exists $L \subseteq A^*$ such that L is recognized by a monoid $N \in \mathsf{V}$ via a morphism $\gamma : A^* \to N$, $\varphi^{-1}(p) \subseteq L$ and $\varphi^{-1}(q) \cap L = \varnothing$. Define $\tau = \gamma \circ \varphi^{-1}$. As φ is surjective and both φ and γ are morphisms, clearly, τ is a relational morphism from M to N. Note that $\tau(p) = \gamma(\varphi^{-1}(p)) \subseteq \gamma(L)$ and $\tau(q) = \gamma(\varphi^{-1}(q)) \subseteq \gamma(L^c)$. It follows that $\tau(p) \cap \tau(q) = \varnothing$, which implies that $\{p, q\}$ is not a V-pointlike set.

On the other hand, suppose that $\{p, q\}$ is not a V-pointlike set. Then there is a monoid $N \in \mathsf{V}$ and a relational morphism $\tau : M \to N$, such that $\tau(p) \cap \tau(q) = \varnothing$. Let R denote the submonoid of $M \times N$ defined by τ, i.e. $R = \{(m, n) \mid n \in \tau(m)\}$. The projections to each of the coordinates are denoted by $\alpha : R \to M$ and $\beta : R \to N$. It follows that $\tau = \beta \circ \alpha^{-1}$. For every $m \in M$, we have that $\tau(m) \neq \varnothing$, hence α is surjective. Thus, there exists a morphism $\delta : A^* \to R$ such that $\varphi = \alpha \circ \delta$. Now let $\gamma : A^* \to N$ be the morphism $\gamma = \beta \circ \delta$. The language $L := \gamma^{-1}(\tau(p))$ is V-recognizable.

$$\begin{tikzcd}
 & A^* \arrow[ddl, bend right, "\varphi"'] \arrow[d, "\delta"] \arrow[ddr, bend left, "\gamma"] & \\
 & R \arrow[dl, "\alpha"'] \arrow[dr, "\beta"] & \\
M \arrow[rr, "\tau"] & & N
\end{tikzcd}$$

We claim that L separates $\varphi^{-1}(p)$ from $\varphi^{-1}(q)$. First note that we are now able to express τ in terms of γ and φ, since

$$\gamma \circ \varphi^{-1} = \beta \circ \delta \circ \varphi^{-1} = \beta \circ \delta \circ (\alpha \circ \delta)^{-1} = \beta \circ \delta \circ \delta^{-1} \circ \alpha^{-1} = \beta \circ \alpha^{-1} = \tau.$$

Clearly, $\varphi^{-1}(p) \subseteq \gamma^{-1}(\gamma(\varphi^{-1}(p))) = L$. Also, since $\tau(q) = \gamma(\varphi^{-1}(q))$ and $\tau(p) \cap \tau(q) = \varnothing$, we have $\varphi^{-1}(q) \cap \gamma^{-1}(\tau(p)) = \varnothing$. Thus, L witnesses that $\varphi^{-1}(p)$ and $\varphi^{-1}(q)$ are V-separable. □

2.2.2 Topological view: closures in the free pro-V semigroup

In [Alm99], a generic connection between profinite semigroup theory and the separation problem for varieties of languages is established. In this paper, it is shown that two regular languages over A are separable by a V-recognizable language if and only if the topological closures of these two languages inside the free pro-V semigroup, $\widehat{F}_{\mathsf{V}}(A)$, intersect. We state this result in Theorem 2.17 and devote this section to providing a proof of this result.

Recall from Section 1.4 that $\widehat{F}_{\mathsf{V}}(A)$ is the completion of the metric space $(A^+/{\sim_{\mathsf{V}}}, d_{\mathsf{V}})$ with the metric given on representatives u, v by $d_{\mathsf{V}}(u,v) = 2^{-r_{\mathsf{V}}(u,v)}$, with $2^{-\infty} = 0$, where $r_{\mathsf{V}}(u,v) = \min\{|S| \mid S \in \mathsf{V} \text{ and } S \text{ separates } u \text{ and } v\} \in \mathbb{N} \cup \{\infty\}$. For simplicity, we assume that every pair of distinct words can be separated by a semigroup from V, that is, the relation $\sim_{\mathsf{V}}$ is the equality relation. We first state a useful property about the metric d_{V}.

Lemma 2.12. *The metric d_{V} on the space $\widehat{F}_{\mathsf{V}}(A)$ is an ultrametric.*

Proof. We first show that d_{V} on A^+ is an ultrametric, that is, that for all $u, v, w \in A^+$, $d_{\mathsf{V}}(u,w) \leq \max(d_{\mathsf{V}}(u,v), d_{\mathsf{V}}(v,w))$. Let S be a semigroup that separates u and w. Then it also separates u and v, or it separates v and w, or both. Thus, $\min(r_{\mathsf{V}}(u,v), r_{\mathsf{V}}(v,w)) \leq r_{\mathsf{V}}(u,w)$, which gives $d_{\mathsf{V}}(u,w) \leq \max(d_{\mathsf{V}}(u,v), d_{\mathsf{V}}(v,w))$. In a similar fashion to proving that d_{V} on $\widehat{F}_{\mathsf{V}}(A)$ is indeed a metric, one can write out the definition of d_{V} on $\widehat{F}_{\mathsf{V}}(A)$, and use that d_{V} on A^+ is an ultrametric, to find that $(\widehat{F}_{\mathsf{V}}(A), d_{\mathsf{V}})$ is an ultrametric space. □

The following proposition is well known, and can be found, for example, in [Alm94] as Proposition 3.4.6. Here, we more or less follow the proof of [Pin11, Proposition 2.3].

Proposition 2.13. *The space $(\widehat{F}_{\mathsf{V}}(A), d_{\mathsf{V}})$ is compact.*

Proof. We use the fact that a complete metric space that is totally bounded is compact (see eg. [Mun00, Theorem 45.1]). Recall that a metric space is totally bounded if for all $\varepsilon > 0$, the space is covered by a finite number of open balls of radius ε. We first prove the following claim, which yields that it suffices to show that A^+ is totally bounded.

Claim. If a metric space (X, d) is totally bounded, then so is its completion $(\widehat{X}, d)$.

Let $\varepsilon > 0$. The space (X, d) is totally bounded, thus in particular there exist $x_1, \dots, x_n$ such that $X \subseteq \bigcup_{i=1}^n B(x_i, \frac{\varepsilon}{2})$. Let y be any element of $\widehat{X}$. Since X is dense in $\widehat{X}$, there is $x \in X$ such that $d(x,y) < \frac{\varepsilon}{2}$. Since X is covered by the open balls of radius $\frac{\varepsilon}{2}$ around the x_i's, there is an x_i such that $d(x_i, x) < \frac{\varepsilon}{2}$. Now, $d(x_i, y) \leq d(x_i, x) + d(x, y) < \varepsilon$. It follows that $\widehat{X} \subseteq \bigcup_{i=1}^n B(x_i, \varepsilon)$.

We now show that A^+ is totally bounded. Let $n \in \mathbb{N}$. We want to show that A^+ is covered by finitely many open balls of radius $< 2^{-n}$.

Lemma 2.12 allows us to define the following congruence relation on A^+,

$$u \sim_n v \iff d_{\mathsf{V}}(u,v) < 2^{-n}.$$

By definition, $u \sim_n v$ if and only if the words cannot be separated by any semigroup of size $\leq n$. Thus, if and only if for all morphisms $\varphi : A^+ \to S$, such that $S \in \mathsf{V}$ and $|S| \leq n$, it holds that $\varphi(u) = \varphi(v)$. Note that a morphism $\varphi : A^+ \to S$ is determined by its image on the letters of A. Since A is finite and since there are only finitely many (finite) semigroups in V of size $\leq n$, there are only finitely many such morphisms. It follows that the congruence relation $\sim_n$ has finite index. Thus, A^+ is covered by the finitely many $\sim_n$-congruence classes, which are open balls of radius $< 2^{-n}$. □

Recall that every $S \in \mathsf{V}$ is endowed with the discrete topology. The definition of d_V makes every morphism $\varphi : A^+ \to S \in \mathsf{V}$ uniformly continuous: for every $u, v \in A^+$, if $d_\mathsf{V}(u, v) < 2^{-|S|}$, then $r_\mathsf{V}(u, v) > |S|$, so S does not separate u and v, which means that $\varphi(u) = \varphi(v)$. From the universal property of the completion $\widehat{F}_\mathsf{V}(A)$ of A^+, it follows that each of the morphisms φ has a unique continuous extension $\hat{\varphi} : \widehat{F}_\mathsf{V}(A) \to S$. To be precise, the extension $\hat{\varphi}$ is defined as follows: for $x \in \widehat{F}_\mathsf{V}(A)$, let $(x_n)_n$ be a Cauchy sequence in the equivalence class of x. Then $\hat{\varphi}(x) = \lim_{n\to\infty} \varphi(x_n)_n$.

It follows from the fact that d_V is an ultrametric on A^+ that multiplication on A^+ is also uniformly continuous. For the same reason as above, we then have that multiplication on $\widehat{F}_\mathsf{V}(A)$ is uniquely defined and continuous. Let $x, y \in \widehat{F}_\mathsf{V}(A)$, and $(x_n)_n, (y_n)_n$ be Cauchy sequences in the equivalence classes of x resp. y. Then $x \cdot y = \lim_{n\to\infty}(x_n \cdot y_n)$.

Let us see that the extensions $\hat{\varphi} : \widehat{F}_\mathsf{V}(A) \to S$ are also morphisms. Consider the product $\hat{\varphi}(x) \cdot \hat{\varphi}(y)$. This is by definition $\lim_{n\to\infty} \varphi(x_n)_n \cdot \lim_{n\to\infty} \varphi(y_n)_n$, which is by continuity of the multiplication equal to $\lim_{n\to\infty}(\varphi(x_n)_n \cdot \varphi(y_n)_n)$. Using that φ is a morphism gives $\lim_{n\to\infty}(\varphi(x_n \cdot y_n)_n)$. By definition, this is $\hat{\varphi}(\lim_{n\to\infty}(x_n \cdot y_n)_n)$. And by definition of the multiplication on $\widehat{F}_\mathsf{V}A$, this is $\hat{\varphi}(x \cdot y)$.

For $L \subseteq \widehat{F}_\mathsf{V}(A)$, we denote by $\overline{L}$ its topological closure in $\widehat{F}_\mathsf{V}(A)$. The following lemma shows a way to describe closures of V-recognizable languages.

Lemma 2.14. *Let $\varphi : A^+ \to S \in \mathsf{V}$ and $K = \varphi^{-1}(P)$ for $P \subseteq S$. Then $\overline{K} = \hat{\varphi}^{-1}(P)$.*

Proof. Since finite union commutes with inverse image and with closure, we have $K = \bigcup_{p\in P} \varphi^{-1}(p)$, $\overline{K} = \overline{\bigcup_{p\in P} \varphi^{-1}(p)} = \bigcup_{p\in P} \overline{\varphi^{-1}(p)}$, and $\bigcup_{p\in P} \hat{\varphi}^{-1}(p) = \hat{\varphi}^{-1}(P)$. It thus suffices to show that $\overline{\varphi^{-1}(p)} = \hat{\varphi}^{-1}(p)$.

The semigroup S is endowed with the discrete topology, thus $\{p\}$ is a clopen set. Since $\hat{\varphi}$ is continuous, $\hat{\varphi}^{-1}(p)$ is clopen. Also, $\varphi^{-1}(p) \subseteq \hat{\varphi}^{-1}(p)$, so $\overline{\varphi^{-1}(p)} \subseteq \overline{\hat{\varphi}^{-1}(p)} = \hat{\varphi}^{-1}(p)$. Conversely, let $u \in \hat{\varphi}^{-1}(p)$. For every $n \geq |S|$, pick a word u_n such that $d_\mathsf{V}(u, u_n) < 2^{-n}$. These words exist since, by construction, A^+ is dense in $\widehat{F}_\mathsf{V}(A)$. Since $n \geq |S|$, $\varphi(u_n) = p$. Thus all words u_n are in $\varphi^{-1}(p)$, and it follows that $u \in \overline{\varphi^{-1}(p)}$. □

For $K \subseteq A^+$, we let $K^c = A^+ \setminus K$ and $(\overline{K})^c = \widehat{F}_\mathsf{V}(A) \setminus \overline{K}$.

Corollary 2.15. *If K is V-recognizable, then $\overline{K^c} = (\overline{K})^c$. If, furthermore, $L \subseteq A^+$ is such that $\overline{L} \subseteq \overline{K}$, then $L \subseteq K$.*

Proof. For the first statement, let $\varphi : A^+ \to S \in \mathsf{V}$, with $K = \varphi^{-1}(P)$. By Lemma 2.14, $\overline{K^c} = \hat{\varphi}^{-1}(S \setminus P) = \widehat{F}_\mathsf{V}(A) \setminus \hat{\varphi}^{-1}(P) = (\overline{K})^c$. To prove the second statement, one can use the first statement to see that $L \cap K^c \subseteq \overline{L} \cap \overline{K^c} \subseteq \overline{K} \cap \overline{K^c} = \overline{K} \cap (\overline{K})^c = \varnothing$. □

Proposition 2.16 (follows from [Alm94, Theorem 3.6.1])**.** *Closures of* V*-recognizable languages form a basis of the topology of* $\widehat{F}_{\mathsf{V}}(A)$*.*

Proof. By Lemma 2.14, the closure of a V-recognizable language is of the form $\hat{\varphi}^{-1}(P)$ for some continuous morphism $\hat{\varphi} : \widehat{F}_{\mathsf{V}}(A) \to S \in \mathsf{V}$. As S is endowed with the discrete topology, $\hat{\varphi}^{-1}(P)$ is open. To show that these open sets indeed form a basis of the topology of $\widehat{F}_{\mathsf{V}}(A)$, we will show that for every element x of every open ball $B(u, 2^{-n})$ of radius 2^{-n} centered at u, there is a set O_n^x that is a closure of a V-recognizable language and is such that $x \in O_n^x \subseteq B(u, 2^{-n})$. To this end, define $O_n^x = \hat{\alpha}_n^{-1}(\hat{\alpha}_n(x))$, where α_n is the product of all morphisms $\varphi : A^+ \to S \in \mathsf{V}$ for $|S| \leq n$. As V is a variety, α_n is a morphism into a semigroup of V. By Lemma 2.14, this means that O_n^x is the closure of the V-recognizable language $\alpha_n^{-1}(\hat{\alpha}_n(x))$. By construction, O_n^x is an open set containing x. To show that it is furthermore contained in $B(u, 2^{-n})$, let $y \in O_n^x$. Then, $\hat{\alpha}_n(y) = \hat{\alpha}_n(x)$, thus for all morphisms $\hat{\varphi} : \widehat{F}_{\mathsf{V}}(A) \to S \in \mathsf{V}$ such that $|S| \leq n$, $\hat{\varphi}(y) = \hat{\varphi}(x)$. This gives that $d_{\mathsf{V}}(y, x) < 2^{-n}$. We already had $d_{\mathsf{V}}(x, u) < 2^{-n}$, thus it follows from Lemma 2.12 that $d_{\mathsf{V}}(y, u) < 2^{-n}$. It follows that $O_n^x \subseteq B(u, 2^{-n})$. □

We are now ready to prove the following theorem that relates the separation problem for a variety V to the intersection of topological closures with respect to $\widehat{F}_{\mathsf{V}}(A)$.

Theorem 2.17 ([Alm99, Lemma 3.2])**.** *The languages* $L_1, L_2 \subseteq A^+$ *are* V*-separable if and only if* $\overline{L_1} \cap \overline{L_2} = \varnothing$*, where the topological closures are taken in the free pro-*V *semigroup* $\widehat{F}_{\mathsf{V}}(A)$*.*

Proof. Suppose first that L_1, L_2 are V-separable. Let K be a V-recognizable language such that $L_1 \subseteq K$ and $L_2 \cap K = \varnothing$. Then, $\overline{L_1} \cap \overline{L_2} \subseteq \overline{K} \cap \overline{K^c}$. By Corollary 2.15, this is equal to $\overline{K} \cap (\overline{K})^c = \varnothing$.

Conversely, if $\overline{L_1} \cap \overline{L_2} = \varnothing$, then every $u \in \overline{L_1}$ belongs to the open set $(\overline{L_2})^c$, so by Proposition 2.16, there exists some V-recognizable language K_u whose closure O_u contains u, and that is such that $O_u \subseteq (\overline{L_2})^c$. Therefore $\overline{L_1} \subseteq \bigcup_{u \in \overline{L_1}} O_u$. The space $\widehat{F}_{\mathsf{V}}(A)$ is compact by Proposition 2.13, and $\overline{L_1}$ is a closed set in this compact space. Thus, $\overline{L_1}$ itself is compact and has a finite cover $O_{u_1} \cup \cdots \cup O_{u_n}$. Then $K = K_{u_1} \cup \cdots \cup K_{u_n}$ is V-recognizable. We have $\overline{L_1} \subseteq \overline{K}$, so by Corollary 2.15, $L_1 \subseteq K$. Also, $K \subseteq O_{u_1} \cup \cdots \cup O_{u_n} \subseteq (\overline{L_2})^c \subseteq L_2^c$. □

To show, for a given variety V, that testing whether $\overline{L_1} \cap \overline{L_2} = \varnothing$ in the free pro-V semigroup is decidable, one usually proves two parts. First, one shows that it is sufficient to compute the closures in a countable subsemigroup of the free pro-V semigroup. Then, one develops techniques to compute the closures in this subsemigroup. This approach has been fruitful and works, for example, for the variety of finite groups, as was conjectured in [PR91]. We will discuss this result for finite groups in more detail in Chapter 3. However, this approach focusing on the intersection of the topological closures of languages in the free pro-V semigroup, in general, does not give any information on how to construct a separator, if it exists.

2.2.3 Combinatorial view: indistinguishable pairs

As we explained in the beginning of Section 2.2, our approach to the separation problem for a class of languages recognized by a variety V is to compute the 2-V-pointlike sets, using elementary combinatorial techniques. A first step to achieve this is to stratify the class of separators from simpler languages to languages that are more and more complicated. Our goal is to show that the separation problem for the full class of separators can be reduced to the separation problem for a restriction of this class.

Usually, a class $\mathcal{S}$ of separators comes with a natural parameter k, which is such that fixing this parameter gives a strictly smaller subclass of languages, which we denote by $\mathcal{S}[k]$, and which is such that increasing the value of k yields a more expressive class of languages. That is, for all $k \in \mathbb{N}$, $\mathcal{S}[k] \subseteq \mathcal{S}[k+1]$. For example, looking at the class K of languages for which membership of a word in a language only depends on the prefix up to a certain length of the word, a natural choice for the parameter along which to stratify this class would be the length of the prefix that is inspected. We will discuss the separation problem for this class in Section 2.3.2.

For other classes of separators, choices of parameters could be, for example, the length of the inspected suffixes, factors or pieces, or the quantifier rank of a formula that defines the language. More in general, one could always take the size of the syntactic monoid of the language as a parameter.

Such a parameter defines a sequence of congruence relations on A^* in the following way. For $u, v \in A^*$,

$$u \sim_k v \iff u \text{ and } v \text{ are not separable by any language from } \mathcal{S}[k].$$

For example, if $\mathcal{S} = \mathsf{K}$ and two words u and v have the same prefix of size k, but not of size $k+1$, then $u \sim_k v$ and $u \not\sim_{k+1} v$.

If the parameter considered is the size of the syntactic monoid of a language, the following equivalence holds for the congruence $\sim_k$. For $u, v \in A^*$,

$$u \sim_k v \iff d_{\mathsf{V}}(u, v) < 2^{-k},$$

for $d_{\mathsf{V}}(u, v)$ as defined in Section 1.4. As there are only finitely many monoids of size up to k, the congruence relation $\sim_k$ has finite index. This will always be the case when $\sim_k$ is defined from logical or combinatorial properties. Since a language is in $\mathcal{S}[k]$ if and only if it is a finite union of $\sim_k$-classes, $\mathcal{S}[k]$ always consists of finitely many languages. Therefore, $\mathcal{S}[k]$-separability is decidable by a brute-force approach.

Note that a language is in $\mathcal{S}$ if and only if it is a finite union of $\sim_k$-equivalence classes for some $k \in \mathbb{N}$. It follows that two input languages are $\mathcal{S}$-separable if and only if *there exists* $k \in \mathbb{N}$ for which there is no word in the first language that is $\sim_k$-equivalent to a word from the second language. A priori, this means that to be able to say that two languages are *not* $\mathcal{S}$-separable, one should check all the infinitely many $\sim_k$-congruences.

We can work around this difficulty, by letting the $\sim_k$-congruences induce a relation on a *finite* set. Recall that we are working with one recognizing device (automaton or monoid) for both input languages. We let a $\sim_k$-congruence induce a relation on a monoid, or on the set of pairs of states of an automaton, that will express that these monoid elements, or these pairs of pairs of states, determine languages that cannot be distinguished by $\mathcal{S}[k]$. Since both languages are finite unions of languages determined by a monoid element (or a pair of states) in one and the same monoid (or automaton), this relation carries the relevant information about $\mathcal{S}[k]$-separability of the languages.

More precisely, let $\mathcal{A} = (A, Q, \delta)$ be a finite automaton. For $(q_1, r_1), (q_2, r_2) \in Q^2$,

$$(q_1, r_1) \text{ and } (q_2, r_2) \text{ are } \mathcal{S}[k]\text{-indistinguishable} \quad \Leftrightarrow \quad \begin{array}{c} \exists u, v \in A^*.\ (q_1, u, r_1) \in \delta^*, \\ (q_2, v, r_2) \in \delta^*,\ u \sim_k v. \end{array}$$

Let M be a finite monoid and let $\varphi : A^* \to M$ be a surjective morphism. Then, for $s, t \in M$,

$$s \text{ and } t \text{ are } \mathcal{S}[k]\text{-indistinguishable} \ \Leftrightarrow\ \exists u \in \varphi^{-1}(s),\ v \in \varphi^{-1}(t).\ \ u \sim_k v.$$

When studying the separation problem for a specific class $\mathcal{S}$, we work either with a monoid or with an automaton, so there will be no ambiguity in the terminology. We usually write $\mathrm{I}_k^{\mathcal{S}}$ for the set of $\mathcal{S}[k]$-indistinguishable monoid elements (so then, $\mathrm{I}_k^{\mathcal{S}} \subseteq M \times M$), or $\mathcal{S}[k]$-indistinguishable pairs of states (in which case, $\mathrm{I}_k^{\mathcal{S}} \subseteq Q^2 \times Q^2$).

We say that two monoid elements, or two pairs of states, are $\mathcal{S}$-indistinguishable if they are $\mathcal{S}[k]$-indistinguishable for all $k \in \mathbb{N}$. The set of $\mathcal{S}$-indistinguishable monoid elements, or pairs of states, is denoted by $\mathrm{I}^{\mathcal{S}}$. This set precisely characterizes which pairs of monoid elements, or which pairs of pairs of states, determine languages that are not $\mathcal{S}$-separable.

Indeed, by definition, $\mathcal{S}[k]$-indistinguishability for all k implies that there is no language in $\mathcal{S}$ that can separate the languages determined by the monoid elements, or by the pairs of states. On the other hand, if there is a k for which the monoid elements, or the pairs of states are *not* $\mathcal{S}[k]$-indistinguishable, then the saturation, with respect to $\sim_k$, of one of the languages determined by the monoid elements, or by the pairs of states is an $\mathcal{S}$-separator.

If we work with a monoid, and if the variety corresponding to $\mathcal{S}$ is V, it follows from Theorem 2.11 that the set $\mathrm{I}^{\mathcal{S}}$ is equal to the set of 2-V-pointlike sets.

A drawback of the notion of $\mathcal{S}[k]$-indistinguishability, compared to the sequence of congruences $\sim_k$ on A^*, is that it is no longer a congruence relation, as it is no longer transitive. Intuitively, this is clear: if both r, s and s, t are $\mathcal{S}[k]$-indistinguishable monoid elements, this means that there are $u, v, v', w \in A^*$, with $u \sim_k v$ and $v' \sim_k w$, and such that u, v, v', w are mapped to r, s, s, t, respectively. If $\mathcal{S}[k]$-indistinguishability were transitive, there would exist words u, v, w mapped to r, s, t respectively, such that v is equivalent to *both* u and w. But the existence of such a triple is not implied by the fact that r, s and s, t are $\mathcal{S}[k]$-indistinguishable. We will see an example of this in Section 2.3.1.

However, there is an important advantage of this notion over the sequence of congruences $\sim_k$ on A^*: the stabilizing behavior. Since we have, for all $k \in \mathbb{N}$, that $\mathcal{S}[k] \subseteq \mathcal{S}[k+1]$, it follows that for all $k \in \mathbb{N}$, $\sim_{k+1} \subseteq \sim_k$. This implies that, for a given monoid or automaton, the following inclusions hold.

$$\mathrm{I}^{\mathcal{S}} = \bigcap_{n \in \mathbb{N}} \mathrm{I}_n^{\mathcal{S}} \subseteq \ldots \subseteq \mathrm{I}_{k+1}^{\mathcal{S}} \subseteq \mathrm{I}_k^{\mathcal{S}} \subseteq \ldots \subseteq \mathrm{I}_1^{\mathcal{S}}. \tag{2.2}$$

Now we use that the set $M \times M$, resp. $Q^2 \times Q^2$, in contrast to A^*, is finite. By the inclusions from (2.2), this means that there must be an index for which the sequence $(\mathrm{I}_k^{\mathcal{S}})_{k \in \mathbb{N}}$ stabilizes. That is, there exists $\kappa \in \mathbb{N}$ such that for every $k \geq \kappa$, we have $\mathrm{I}^{\mathcal{S}} = \mathrm{I}_k^{\mathcal{S}}$.

While the existence of such a κ is immediate from the definitions, computing a bound on κ is a difficult problem. Note that if we establish such a bound (which of course depends on the input languages), then the $\mathcal{S}$-separation problem reduces to the $\mathcal{S}[\kappa]$-separation problem. The class $\mathcal{S}[\kappa]$ consists of finitely many languages, thus this would already yield decidability of the $\mathcal{S}$-separation problem: one can use a brute force algorithm to test all of these languages. Furthermore, such a bound would imply that the saturation of L_1, with respect to $\sim_\kappa$, is a language from $\mathcal{S}$ that separates L_1 from L_2, in case they are $\mathcal{S}$-separable. We thus obtain a description of a potential separator in terms of this bound.

For the classes that we study in this dissertation, we obtain such a bound by looking at the presence of patterns in the automaton or monoid, which are specific to the class $\mathcal{S}$, and which give us a framework to decompose words. This decomposition should be such that pumping arguments can be applied to it. We then have to show that if there is an $\mathcal{S}[\kappa]$-indistinguishable pair, there will be $\mathcal{S}[k]$-indistinguishable pairs, for every $k \in \mathbb{N}$.

2.3 Basic examples

In this section, we illustrate the notions discussed above, by presenting the separation problem for two basic examples: the variety Sl of semilattices, and the variety K of semigroups whose idempotents are left zeros. In the following chapters, we discuss the separation problem for more involved classes of languages.

2.3.1 Example I: Sl

A semigroup S that is idempotent and commutative, i.e. that is such that for all $s, t \in S$, $s^2 = s$ and $st = ts$, is called a *semilattice*. The variety of all finite semilattices is denoted by Sl (and it is also known under the name J_1). This variety defines the class of languages that are closed under duplication of letters within a word and under permutation of the letters of a word. Thus, the class of Sl-recognizable languages consists of those languages that can be described purely in terms of their alphabets, that is, those languages that are boolean combinations of languages of the form B^*, where B is a subalphabet. For this reason, the Sl-recognizable languages are also called the *alphabet-testable languages*. The logic that corresponds to this class is $\mathsf{FO}^1(<)$, the fragment of first-order logic in which each formula can use only one variable. The Sl-recognizable languages form a strict subclass of the piecewise testable languages that we will encounter in Chapter 4.

For a language L_1, the smallest (with respect to inclusion) Sl-recognizable language that contains L_1 exists: it is the language $\bigcup_{w \in L_1} \{v \in A^* \mid \mathsf{alph}(v) = \mathsf{alph}(w)\}$. The existence of such a smallest potential separator, at first sight, seems to largely simplify the question of the separation problem: to answer the separation problem for L_1 and L_2, one only has to verify whether this language is disjoint from L_2. However, as we will see in the end of this section, the complexity of the Sl-separation problem is, surprisingly, much worse than the complexity of the separation problem for more complicated classes of languages that do not admit such a smallest separator. In the chapters to come, we will see examples of such more involved classes for which the complexity of the separation problem is lower.

The following example spells out the SI-separability for three pairs of languages. Although the separation problem for this example is immediately clear, we elaborate a bit more on the algebraic notions behind this example, as it serves to illustrate that being 2-SI-pointlike is not a transitive relation. More in general, being 2-V-pointlike is not a transitive relation.

Example 2.18. Let $L_1 = \{a\}, L_2 = \{a, b\}^{\geqslant 2}, L_3 = \{b\}$. Clearly, L_1 and L_3 are SI-separable (for example by the language a^+), while neither (L_1, L_2) nor (L_2, L_3) are SI-separable. The three languages are simultaneously recognized by the semigroup $S = \{s_a, s_b, 0\}$, in which multiplying any two elements gives 0. If $\varphi : A^+ \to S$ sends a to s_a and b to s_b, then $L_1 = \varphi^{-1}(s_a)$, $L_2 = \varphi^{-1}(0)$ and $L_3 = \varphi^{-1}(s_b)$. By Theorem 2.11, it follows that $\{s_a, 0\}$ and $\{s_b, 0\}$ must be SI-pointlike sets, and that $\{s_a, s_b\}$ is not. To see this directly, let T be any semigroup in SI and let $\tau : S \to T$ be any relational morphism. By definition, there exists $t \in \tau(s_a)$. As T is idempotent, we have $t = t^2 \in \tau(s_a)\tau(s_a) \subseteq \tau(s_a s_a) = \tau(0)$. Thus, $\{s_a, 0\} \subseteq \tau^{-1}(t)$, i.e. $\{s_a, 0\}$ is a 2-SI-pointlike set. Similarly, $\{s_b, 0\}$ is a 2-SI-pointlike set.

To see that $\{s_a, s_b\}$, nevertheless, is not a 2-SI-pointlike set, consider the semigroup $U_1 = \{\mathbf{0}, \mathbf{1}\}$, with multiplication given by $\mathbf{0} = \mathbf{0} \cdot \mathbf{0} = \mathbf{0} \cdot \mathbf{1} = \mathbf{1} \cdot \mathbf{0}$ and $\mathbf{1} = \mathbf{1} \cdot \mathbf{1}$. Clearly, U_1 is a semilattice. Define the following relational morphism.

$$\begin{array}{rrcl} \tau : & S & \to & U_1 \\ & s_a & \mapsto & \{\mathbf{0}\} \\ & s_b & \mapsto & \{\mathbf{1}\} \\ & 0 & \mapsto & \{\mathbf{0}, \mathbf{1}\} \end{array}$$

As $\tau(s_a) \cap \tau(s_b) = \varnothing$, it follows that $\{s_a, s_b\}$ is not a 2-SI-pointlike set. Note that this also implies that $\{s_a, s_b, 0\}$ is not a 3-SI-pointlike set.

Complexity of separation by SI-recognizable languages

We will now analyze the complexity of the separation problem for SI-recognizable languages. This analysis can also be found in the appendix of [RZ13]. As we noted before, one can always compute the smallest SI-recognizable language containing a given regular language. This entails an algorithm to decide whether two regular languages are separable by an SI-recognizable language. Perhaps surprisingly however, separation by an SI-recognizable language is computationally harder than, for example, by a piecewise testable language.

Recall that two languages are SI-separable if and only if they do not contain any words that share the same alphabet. A consequence of the next lemma, therefore, is that even when starting from deterministic finite automata, the separation problem is co-NP-complete for SI-recognizable languages.

Lemma 2.19. *The following problem is* NP*-complete.*

Input: *An alphabet $A = \{a_1, a_2, \ldots, a_n\}$ and two DFA's $\mathcal{A}_1, \mathcal{A}_2$ over A.*
Question: *Do there exist $u \in L(\mathcal{A}_1)$ and $v \in L(\mathcal{A}_2)$ such that $\mathsf{alph}(u) = \mathsf{alph}(v)$?*

Proof. We will first prove that the problem is NP-hard by giving a reduction from **3-SAT** to this problem. A **3-SAT** formula is a formula in conjunctive normal form, where each conjunct is the disjunction of at most three variables or negations of variables. The problem of deciding whether there exists a valuation that satisfies a **3-SAT** formula is called **3-SAT**, and it is well known that this is an NP-complete problem, see for example [GJ90].

Let φ be a **3-SAT** formula over the variables $\{x_1, \ldots, x_n\}$. Define A as the alphabet $\{x_1, \ldots x_n, \neg x_1, \ldots, \neg x_n\}$. Let $\mathcal{A}_1$ be the following automaton.

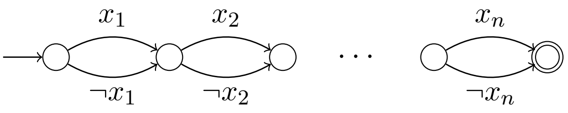


Figure 2.2: The automaton $\mathcal{A}_1$.

Let $\mathcal{A}_2$ be the serial automaton constructed as follows. For every disjunct d in the i-th clause of φ, add an arrow from state i to $i+1$, labeled by d. Then concatenate this with a copy of $\mathcal{A}_1$. For example, if $\varphi = (x_1 \vee x_3 \vee \neg x_4) \wedge \ldots \wedge (x_4 \vee \neg x_5 \vee x_2)$, the automaton $\mathcal{A}_2$ is the following.

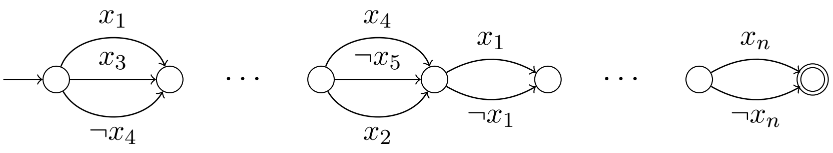


Figure 2.3: The automaton $\mathcal{A}_2$, for $\varphi = (x_1 \vee x_3 \vee \neg x_4) \wedge \ldots \wedge (x_4 \vee \neg x_5 \vee x_2)$.

We will show that φ is satisfiable if and only if the question mentioned above is answered positively for these $\mathcal{A}_1$ and $\mathcal{A}_2$.

Suppose φ is satisfiable. Then, there is a valuation $v : \{x_1, \ldots, x_n\} \to \{0, 1\}$ such that φ evaluates to 1 under v. Define $u := y_1 \cdots y_n$, with $y_i = x_i$ if $v(x_i) = 1$, and $y_i = \neg x_i$ if $v(x_i) = 0$. In each of the k clauses of φ, there is at least one disjunct d for which $v(d) = 1$. Define $v := w_1 \cdots w_k u$, where w_i is any one of the disjuncts in the i-th clause that is evaluated to 1. Now, $u \in L(\mathcal{A}_1), v \in L(\mathcal{A}_2)$, and by soundness of the valuation function, $\mathsf{alph}(u) = \mathsf{alph}(v)$.

On the other hand, suppose that for these $\mathcal{A}_1$ and $\mathcal{A}_2$, there are $u \in L(\mathcal{A}_1)$, $v \in L(\mathcal{A}_2)$ with $\mathsf{alph}(u) = \mathsf{alph}(v)$. By construction of $\mathcal{A}_1$, for every i, $\mathsf{alph}(u)$ contains either x_i or $\neg x_i$. By construction of $\mathcal{A}_2$ and since $\mathsf{alph}(u) = \mathsf{alph}(v)$, we have that $v = wu$, for some w, for which $\mathsf{alph}(w) \subseteq \mathsf{alph}(u)$. Define the valuation

$$\begin{array}{rccll} v: & \{x_1, \ldots x_n\} & \to & \{0,1\} & \\ & x_i & \mapsto & 1 & \text{if } x_i \in \mathsf{alph}(u) \\ & x_i & \mapsto & 0 & \text{else} \end{array}$$

Now v sends all variables occurring in w to 1, which gives that φ evaluates to 1 under v. Thus, we have shown that the problem is NP-hard.

Furthermore, to see that the problem is in NP, we can first guess the alphabet C that the words u and v will use, guess the order of the first occurrences of each of these letters in u, and similarly guess this order for v. Verifying whether there exist such $u \in L(\mathcal{A}_1), v \in L(\mathcal{A}_2)$ can clearly be performed in polynomial time: this now amounts to intersecting $\mathcal{A}_1$ with a DFA $\mathcal{A}_u$ that accepts all words over C that respect the order of the first occurrences of the letters as guessed for u, and similarly, intersecting $\mathcal{A}_2$ with $\mathcal{A}_v$ that is defined analogously. Now, the guess leads to a positive answer if and only if both intersections are nonempty. □

Since the question in the previous lemma is answered positively if and only if the languages are *not* SI-separable, we obtain the following corollary.

Corollary 2.20. *It is a* CO-NP*-complete problem to decide whether two regular languages, defined by two deterministic finite automata, are* SI*-separable.*

2.3.2 Example II: K

We will now look at a variety for which, contrary to the usual case, the free pro-V semigroup is easy to describe. First, we discuss the topological view on the separation problem for this variety. This discussion can also be found in [RZ13]. We then present an approach to solve this separation problem using indistinguishable pairs of states.

An element s of a semigroup S is a *left zero* if, for all $t \in S$, $st = s$. The variety of finite semigroups S in which all idempotents are left zeros is denoted by K. The class of languages that are K-recognizable consists of the finite boolean combinations of languages of the form uA^*, for a finite word u. One can verify that these coincide with the languages of the shape $XA^* \cup Y$, where X and Y are finite subsets of A^+. Because membership in this class can be tested by inspecting the set of prefixes up to a certain length, the K-recognizable languages are also called the *prefix-testable languages*.

Computing the intersection of the closures in $\widehat{F}_{\mathsf{K}}(A)$

Since K is a variety, Theorem 2.17 gives that testing whether two languages are K-separable can be done by checking that their topological closures, in the free pro-K semigroup $\widehat{F}_{\mathsf{K}}(A)$, have a nonempty intersection. It turns out that for the variety K, this profinite semigroup is easy to describe (see [Alm94, Section 3.7]). It is $A^+ \cup A^\infty$, where A^∞ denotes the set of right infinite words over A. Multiplication in this semigroup is defined as follows. Infinite words are left zeros ($v \cdot w = v$ if $v \in A^\infty$), and multiplication on the left by a finite word is the usual concatenation ($v \cdot w = vw$ if $v \in A^+$). Finally, a sequence converges

- to a finite word u if it is ultimately equal to u,
- to an infinite word v if for every finite prefix x of v, the sequence ultimately belongs to $x(A^+ \cup A^\infty)$.

Thus, by Theorem 2.17, it follows that two disjoint regular languages L_1, L_2 are not K-separable if and only if there exists an infinite word $v \in A^\infty$ such that for every prefix x of v, there exist $w_1, w_2 \in A^*$ such that $xw_1 \in L_1, xw_2 \in L_2$.

This can be tested via Büchi automata (an introduction to these automata can be found, for example, in [PP04]). From a given NFA $\mathcal{A}$ that recognizes a language L for the sets of initial states I and final states F, one can compute a Büchi automaton recognizing the language of infinite words that belong to the closure of L, as follows:

1. Trim $\mathcal{A}$, by removing all states that cannot be reached from a state in I or from which one cannot reach any state in F. This can be performed in linear time wrt. the size of $\mathcal{A}$, and does not change the language recognized by $\mathcal{A}$.
2. Build the Büchi automaton obtained from the resulting trim automaton by declaring all states accepting.

Let us see that this Büchi automaton indeed recognizes the language of infinite words from the closure of L. Let $v \in A^\infty$ be accepted by this Büchi automaton. Since we trimmed the automaton $\mathcal{A}$, there is, for every state in the automaton, a path to some state in F. This means that for every finite prefix x of v, there is a word w_x such that $x \cdot w_x \in L$. One can thus construct a sequence of elements from L that converges to v. Conversely, suppose there is a sequence in L that converges to an infinite word v. Since every finite prefix of v is part of a path in $\mathcal{A}$, we can build a run for v in the Büchi automaton. Then, the infinite word v will be accepted, since we declared all states accepting.

This construction yields a PTIME algorithm to decide separability by a prefix-testable language: first check that $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) = \varnothing$. If so, the intersection of the languages of infinite words belonging to the closures of $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ is empty if and only if the languages are K-separable. This information can be computed by the usual product construction (since all states are accepting) and a test whether this Büchi automaton accepts at least one word. We obtain the following proposition.

Proposition 2.21. *One can decide in* PTIME*, with respect to the sizes of the recognizing automata, whether two languages can be separated by a prefix-testable language.*

Computing K-indistinguishable pairs of states

For the variety K, however, one can compute the same information about separability without appealing to infinite words or Büchi automata. We will show this using the concept of K-indistinguishable pairs of states. As explained in Section 2.2.3, we will work with one automaton recognizing both input languages.

An obvious parameter for the class K is the length of the prefixes considered: we say that two pairs of states are $\sim_n$-equivalent if there are words that share the same prefix of length n and that can be read between the respective pairs of states, or if the languages defined by the pairs of states are not disjoint.

Definition 2.22. Let $\mathcal{A} = (A, Q, \delta)$ be an NFA, and let $(q_1, r_1), (q_2, r_2) \in Q^2$. Then,

$$(q_1, r_1) \sim_n (q_2, r_2) \quad \Leftrightarrow \quad \begin{cases} \exists u \in A^n.\ \exists v, w \in A^*.\ (q_1, uv, r_1), (q_2, uw, r_2) \in \delta^*, \text{ or} \\ \exists u \in A^{<n}.\ (q_1, u, r_1), (q_2, u, r_2) \in \delta^*, \end{cases}$$

in which case the pairs (q_1, r_1) and (q_2, r_2) are called $\mathsf{K}[n]$-*indistinguishable.* They are called K-*indistinguishable* if for all $n \in \mathbb{N}$, $(q_1, r_1) \sim_n (q_2, r_2)$.

The following lemma shows that this notion of K-indistinguishable pairs indeed captures the right information.

Lemma 2.23. *Let $\mathcal{A} = (A, Q, \delta)$ be an NFA. Let $(q_1, r_1), (q_2, r_2)$ be pairs of states that determine languages L_1 and L_2. Then, $(q_1, r_1), (q_2, r_2)$ are not K-indistinguishable if and only if L_1 and L_2 are K-separable.*

Proof. We use the following notation. For $i = 1, 2$,

$$\begin{aligned} X_{i,n} &:= \{u \in A^n \mid \exists v \in A^*.\ (q_i, uv, r_i) \in \delta^*\}, \\ Y_{i,n} &:= \{u \in A^{<n} \mid (q_i, u, r_i) \in \delta^*\}. \end{aligned}$$

That is, the set $X_{i,n}$ consists of the prefixes of length n occurring in words of L_i, and the set $Y_{i,n}$ consists of the words of L_i whose length is strictly smaller than n. Note that by definition, $(q_1, r_1), (q_2, r_2)$ are not K-indistinguishable if and only if there exists $n \in \mathbb{N}$, such that

- $X_{1,n} \cap X_{2,n} = \varnothing$, and
- $Y_{1,n} \cap Y_{2,n} = \varnothing$.

Suppose that $(q_1, r_1), (q_2, r_2)$ are not K-indistinguishable. Then, by definition, there exists $n \in \mathbb{N}$ such that $(q_1, r_1) \not\sim_n (q_2, r_2)$. This means that $X_{1,n} \cap X_{2,n} = \varnothing$ and $Y_{1,n} \cap Y_{2,n} = \varnothing$. Since all words in $X_{i,n}$ are of length exactly n, it also follows that $X_{1,n}A^* \cap X_{2,n}A^* = \varnothing$.

Since $X_{1,n}$ and $Y_{1,n}$ are finite, the language $X_{1,n}A^* \cup Y_{1,n}$ is K-recognizable and, clearly, it contains L_1. Note that $(X_{1,n}A^* \cup Y_{1,n}) \cap L_2 \subseteq (X_{1,n}A^* \cup Y_{1,n}) \cap (X_{2,n}A^* \cup Y_{2,n})$. Clearly, $X_{1,n}A^*$ and $Y_{2,n}$ are disjoint, as well as $Y_{1,n}$ and $X_{2,n}A^*$, as their elements have different length. Thus, $(X_{1,n}A^* \cup Y_{1,n}) \cap (X_{2,n}A^* \cup Y_{2,n}) = (X_{1,n}A^* \cap X_{2,n}A^*) \cup (Y_{1,n} \cap Y_{2,n})$, which is empty by assumption. Summing up, $X_{1,n}A^* \cup Y_{1,n}$ is disjoint from L_2, thus this language K-separates L_1 from L_2.

Now suppose that L_1 and L_2 are K-separable, and let us show that (q_1, r_1) and (q_2, r_2) are not K-indistinguishable. In particular, L_1 and L_2 are disjoint. Hence, for all $n \in \mathbb{N}$, $Y_{1,n} \cap Y_{2,n} = \varnothing$. Let X, Y be finite subsets of A^+ such that $L = XA^* \cup Y$ separates L_1 from L_2. Let n be strictly bigger than the length of the longest word in the finite sets X and Y. Then, since $L_1 \subseteq L$, it follows that $L_1 \cap A^{\geq n} \subseteq XA^*$ and that $X_{1,n}A^* \subseteq XA^* \subseteq L$. Suppose there exists $u \in X_{1,n} \cap X_{2,n}$. This means there is $v \in A^*$ such that $uv \in L_2$, and that $uv \in X_{1,n}A^* \subseteq L$. But $L_2 \cap L = \varnothing$, and thus it follows that $X_{1,n} \cap X_{2,n} = \varnothing$. Thus, $(q_1, r_1), (q_2, r_2)$ are not K-indistinguishable. □

In Proposition 2.24, we show that the length of the prefixes that one needs to take into account to decide K-indistinguishability can be bounded using information about the automaton recognizing the input languages. One applies a pumping argument to show that this bound is correct. It follows from Lemma 2.23 that finding the K-indistinguishable pairs of states in this automaton solves the separation problem for all pairs of languages that the automaton recognizes, in particular for the two input languages. These proofs provide an easy example of the method that we will use in later chapters in more complicated contexts.

Proposition 2.24. *Let $\mathcal{A} = (A, Q, \delta)$ be an NFA and let $N = |Q|^2$. Then, the pairs of states $(q_1, r_1), (q_2, r_2)$ are K-indistinguishable if and only if they are $\mathsf{K}[N]$-indistinguishable.*

Proof. As before, we denote the set of prefixes of length n, occurring between q_i and r_i, by $X_{i,n}$. That is,

$$X_{i,n} \ := \ \{u \in A^n \mid \exists v \in A^*. \ (q_i, uv, r_i) \in \delta^*\}.$$

By definition, the direction from left to right is true.

Suppose that $(q_1, r_1), (q_2, r_2)$ are $\mathsf{K}[N]$-indistinguishable. We want to show that they are K-indistinguishable, that is, that they are $\mathsf{K}[n]$-indistinguishable, for every $n \in \mathbb{N}$. The fact that $(q_1, r_1), (q_2, r_2)$ are $\mathsf{K}[N]$-indistinguishable means that there exist $u \in A^N$ and $v, w \in A^*$ such that $(q_1, uv, r_1), (q_2, uw, r_2) \in \delta^*$, or there exists $u \in A^{<N}$ such that $(q_1, u, r_1), (q_2, u, r_2) \in \delta^*$. In the second case, the pairs are clearly K-indistinguishable. In the first case, there is $u = u_1 \cdots u_N \in X_{1,N} \cap X_{2,N}$. Then the following paths exist in $\mathcal{A}$.

$$\begin{aligned} q_1 = q_1^0 \ &\xrightarrow{u_1} \ q_1^1 \ \xrightarrow{u_2} \ \cdots \ \xrightarrow{u_N} \ q_1^N \ \overset{v}{\dashrightarrow} \ r_1, \\ q_2 = q_2^0 \ &\xrightarrow{u_1} \ q_2^1 \ \xrightarrow{u_2} \ \cdots \ \xrightarrow{u_N} \ q_2^N \ \overset{w}{\dashrightarrow} \ r_2. \end{aligned}$$

Consider the pairs (q_1^i, q_2^i), for $i \in \{0, \ldots, N\}$. Among these $N + 1 = |Q|^2 + 1$ pairs, there is at least one pair occurring twice. Thus there exist i, and $0 < k \leq N - i$, such that $(q_1^i, q_2^i) = (q_1^{i+k}, q_2^{i+k})$.

This means that for every $n \in \mathbb{N}$, the prefix of length n of the word

$$u_1 u_2 \cdots u_i (u_{i+1} \cdots u_{i+k})^n u_{i+k+1} \cdots u_N$$

is an element of $X_{1,n} \cap X_{2,n}$, since there exist paths in $\mathcal{A}$,

$$\begin{aligned} q_1 = q_1^0 \ &\xrightarrow{u_1} \ \cdots \ \xrightarrow{u_i} \ q_1^i \ \overset{(u_{i+1}\cdots u_{i+k})^n}{\dashrightarrow} \ q_1^{i+k} \ \xrightarrow{u_{i+k+1}} \ q_1^{i+k+1} \ \cdots \ \xrightarrow{u_N} \ q_1^N \ \overset{v}{\dashrightarrow} \ r_1, \\ q_2 = q_2^0 \ &\xrightarrow{u_1} \ \cdots \ \xrightarrow{u_i} \ q_2^i \ \overset{(u_{i+1}\cdots u_{i+k})^n}{\dashrightarrow} \ q_2^{i+k} \ \xrightarrow{u_{i+k+1}} \ q_2^{i+k+1} \ \cdots \ \xrightarrow{u_N} \ q_2^N \ \overset{w}{\dashrightarrow} \ r_2. \end{aligned}$$

Thus, for every $n \in \mathbb{N}$, words can be read between q_1 and r_1, and between q_2 and r_2, that have the same prefix of length n. It follows that $(q_1, r_1), (q_2, r_2)$ are $\mathsf{K}[n]$-indistinguishable for every n, that is, they are K-indistinguishable. □

Thus, besides providing the decidability of the separation problem for K, Proposition 2.24 can also be used to obtain a description of a K-separator, whenever the two languages are K-separable. Namely, the saturation of one of the languages with respect to the congruence relation $\sim_{|Q|^2}$, denoted by $[L_1]_{\sim_{|Q|^2}}$, is a K-separator. It follows from the proof of Lemma 2.23 that a saturation with respect to the $\mathsf{K}[n]$-indistinguishability relation is easy to compute, and that $[L_1]_{\sim_{|Q|^2}}$ is equal to $X_{1,|Q|^2} A^* \cup Y_{1,|Q|^2}$. In general, however, providing an alternative description of such a saturation is a difficult problem.

The relation between this combinatorial approach and the topological approach previously described is the following. Note that an element that is in the intersection denoted above as $X_{1,N} \cap X_{2,N}$, by a pumping argument, gives rise to a sequence of pairs of words whose common prefix keeps growing. This leads to an infinite word in the intersection of the closures of the two languages. On the other hand, an infinite word occurs in the intersection of the closures if there is such a sequence. It is clear that in this case, for every $n \in \mathbb{N}$, $X_{1,n} \cap X_{2,n} \neq \varnothing$.

Chapter 3

Group languages

3.1 Characterizations of group languages 37

3.2 The separation problem for group languages 38

3.2.1 Closures in the free group 40

3.2.2 Decidability of G-separability and a construction of a separator . . . 42

3.2.3 Closures in the free monoid 44

In this chapter we study the separation problem for the class of group languages. This class of languages has been extensively studied and many useful results are known from the literature. We can apply some of these results directly to solve the separation problem for this class. For this reason, the approach taken in this chapter differs from the approach described in Section 2.2.3, which we will take in the following chapters.

We show how decidability of the separation problem for group languages follows from a theorem of Ribes and Zalesskiĭ [RZ93] about closed sets in the free group, which was conjectured in [PR91], and which generalizes a result from [Hal50]. It is shown in [HMPR91] that this statement is equivalent to a conjecture of Rhodes related to relational morphisms into groups [Rho87], which was solved by Ash in [Ash91].

A constructive proof of the theorem of Ribes and Zalesskiĭ was given in [AS05]. We also show how to use this constructive proof to find a separating group language, in case it exists.

3.1 Characterizations of group languages

Group languages are languages whose syntactic semigroup is a group, that is, there is an identity element ($e \in S$ such that for all $s \in S$, $se = es = s$), and for each element $s \in S$, there is an inverse element ($s' \in S$ such that $ss' = s's = e$). The variety of groups, denoted by G, is defined by the identity $x^\omega y = yx^\omega = y$, usually abbreviated as $x^\omega = 1$.

For example, the language $a(aa)^*$ is a group language. It is recognized by the finite group $\mathbb{Z}/2\mathbb{Z}$, via the morphism φ that sends a to 1 (and thus sends aa to 0). Indeed, $a(aa)^* = \varphi^{-1}(1)$.

There is also a graphical characterization of group languages: the automata recognizing group languages are the so-called permutation automata. Let us first introduce these.

Definition 3.1. An automaton $\mathcal{A} = (A, Q, \delta)$ is a *permutation automaton* if, for all $a \in A$, the mapping $\delta(\_, a) : Q \to Q$ is a permutation of Q.

In other words, a permutation automaton is complete, and the transition function is deterministic and co-deterministic.

It is not hard to see that the group languages are precisely the languages recognized by permutation automata. To this end, let $\varphi : A^* \to G \in \mathsf{G}$ and let $L = \varphi^{-1}(P)$ for some $P \subseteq G$. Define $\mathcal{A} = (A, G, \delta)$, with $\delta(g, a) = g \cdot \varphi(a)$. By construction, $\mathcal{A}$ is complete and deterministic. It follows from the cancellative property of groups that it is also co-deterministic. Taking $I = \{1\}$ and $F = P$, we see that the permutation automaton $\mathcal{A}$ recognizes L. On the other hand, if L is recognized by a permutation automaton, we can construct its transition monoid M. By definition, M consists of a set of permutations of Q, closed under multiplication, and where the empty word induces the identity permutation. It is clear that for all $m \in M$, $m^{|Q|!-1}$ is the inverse of m. It follows that M is a group that recognizes L.

3.2 The separation problem for group languages

Applying the result from [Alm99], which we proved in Theorem 2.17, to the variety of groups gives that two regular languages over A are G-separable if and only if their closures in the free profinite group $\widehat{F}_{\mathsf{G}}(A)$ have an empty intersection. In Theorem 3.6, we will show that for the variety of groups, results from the literature imply that this happens exactly when their closures *in the free group* (endowed with the topology induced from the free profinite group) have an empty intersection. Furthermore, we will show that as a consequence of a result from [AS05], a separating group language can be effectively constructed whenever two regular languages are G-separable.

The free group on A, denoted by $FG(A)$, is obtained as follows. Let $\overline{A}$ be a disjoint copy of A. Then $FG(A)$ is the quotient of $(A \cup \overline{A})^*$, in which each word is reduced by the relations $\{a\overline{a} = \varepsilon, \overline{\overline{a}} = a \mid a \in A \cup \overline{A}\}$. Throughout this section, we endow the free group with the topology induced from the free profinite group. This is the coarsest topology that makes every group morphism from $FG(A)$ onto a finite group (endowed with the discrete topology) continuous.

Every finite group G is a quotient of the free group. We denote the associated canonical morphism by $\varphi_G : FG(A) \to G$. Finding a group language that separates two given languages L_1 and L_2 is equivalent to finding a finite group G such that $\varphi_G(L_1) \cap \varphi_G(L_2) = \varnothing$. The direction from left to right is clear since, by the universal property of the free group, there is a unique extension of a morphism $\varphi : A^* \to G$ to the domain $FG(A)$. Conversely, if G is a finite group such that $\varphi_G(L_1) \cap \varphi_G(L_2) = \varnothing$, then the language $\varphi_G^{-1}{\upharpoonright_{A^*}} (\varphi_G(L_1))$ is recognized by G, contains L_1 and is disjoint from L_2. Therefore, we also say that the morphism φ_G, or the group G, separates L_1 and L_2.

The original result from the literature that we want to use does not explicitly speak about separation, but it speaks about closed sets in the free group. The following lemma shows that there is a relation between these two notions.

Lemma 3.2. *Let $H \subseteq FG(A)$. Then, H is closed in the free group if and only if for all $w \in FG(A)$ such that $w \notin H$, there is a finite group that separates w from H.*

Proof. Suppose that for all $w \notin H$, there exists a finite group that separates w from H. That is, there exists $\varphi_G : FG(A) \to G$ such that $\varphi_G(w) \notin \varphi_G(H)$. Then $\varphi_G^{-1}(\varphi_G(w)) \subseteq H^c$. Thus,

$$H^c = \bigcup_{w \notin H} \{w\} = \bigcup_{w \notin H} \varphi_G^{-1}(\varphi_G(w)).$$

Since φ_G is continuous and G is endowed with the discrete topology, it follows that H^c is open, hence H is closed.

On the other hand, suppose that H^c is open and let $w \in FG(A)$ be such that $w \notin H$. Since the topology on $FG(A)$ is the subspace topology of the free profinite group, it follows from Proposition 2.16 that the closures of group languages, taken in the free group, form a basis for the topology on $FG(A)$. Thus, there exists a set U that is the closure in the free group of a group language, and that is such that $w \in U \subseteq H^c$. In a similar way as we proved Lemma 2.14, one can prove that the closures of group languages in the free group are of the shape $\varphi^{-1}(P)$ for some $\varphi : FG(A) \to G$ and $P \subseteq G$. This means that there exist $\varphi : FG(A) \to G$ and $P \subseteq G$ such that $w \in \varphi^{-1}(P) = U \subseteq H^c$. It follows that $\varphi(w) \notin \varphi(H)$. □

In [Hal50], Hall proved that every finitely generated subgroup of the free group is closed. By Lemma 3.2, this can be seen as a first separation result for groups. Hall's result can be generalized to the following well-known theorem. We will use this theorem to see that two regular languages are G-separable if and only if their closures in the free group do not intersect. A more general version of this theorem is proved in [RZ93]. In the following form, this statement was first raised as a conjecture in [PR91]. In [HMPR91], it is proved that this statement is equivalent to the so-called type II conjecture of Rhodes [Rho87], which was solved by Ash in [Ash91].

Theorem 3.3 ([RZ93])**.** *Let $H_1, \dots, H_n$ be finitely generated subgroups of the free group. The product $H_1 \cdots H_n$ is closed in the free group.*

By Lemma 3.2, the following statement about separation is equivalent to Theorem 3.3: For $H_1, \dots, H_n$ finitely generated subgroups of the free group, and $w \in FG(A)$ such that $w \notin H_1 \cdots H_n$, there exists a finite quotient G of $FG(A)$ for which the canonical morphism separates w and $H_1 \cdots H_n$.

We want to use Theorem 3.3 to show that two regular languages over A are G-separable if and only if their closures *in the free group* have an empty intersection. Our approach is to construct, from the input languages, a word w in the free group and a product P of finitely generated subgroups. We want these to be such that the closures in the free group of the original input languages do not intersect if and only if the word does not lie in the product. The alternative formulation of Theorem 3.3 then says that this happens if and only if there exists a finite quotient G of $FG(A)$ for which the canonical morphism separates the word and the product. We will argue, using the construction of w and P, that this finite group then also separates the input languages. To sum up, we reduce the separability of two languages to the separability of a word and product constructed from these languages.

Furthermore, it is shown in [AS05] how to construct, from the input data, a finite group as described in the alternative formulation of Theorem 3.3, i.e. a finite group that separates a word outside of $H_1 \cdots H_n$ from the product $H_1 \cdots H_n$. Applying this construction to the word w and the product P described above will allow us to give a description of a group language separating the original input languages, in case it exists. We refer to the paper [AS05] for the construction of this finite group and just mention the result here.

Theorem 3.4 ([AS05, Theorem 3.3])**.** *Let $w \in FG(A)$ and let $H_1, \ldots, H_n$ be finitely generated subgroups of $FG(A)$. If $w \notin H_1 \cdots H_n$, then a finite group that separates w from $H_1 \cdots H_n$ can effectively be constructed.*

In order to be able to construct, from two regular languages L_1 and L_2, the word w and the product P with the desired properties (i.e. $w \notin P \;\Leftrightarrow\; \overline{L_1} \cap \overline{L_2} = \varnothing$), we need to take a closer look at the closure of a regular language in the free group. In the following section, we provide the standard construction for these closures.

3.2.1 Closures in the free group

As observed in [Pin91], every regular language (represented by a regular expression or an automaton) can effectively be written as a finite union of *simple* sets, which are sets of the form $L_0^* u_1 L_1^* \cdots u_n L_n^*$, where each $u_i \in A^*$, and each L_i is a regular language over A. As closure always commutes with finite union, it suffices to see how to compute the closures of simple sets.

It is proved in [PR91], that as a consequence of Theorem 3.3 (which was still a conjecture at the time), the closure in the free group of a simple set $L = L_0^* u_1 L_1^* \cdots u_n L_n^*$ is

$$\overline{L} = \langle L_0 \rangle u_1 \langle L_1 \rangle \cdots u_n \langle L_n \rangle.$$

Here, $\langle L_i \rangle$ denotes the subgroup of the free group that is generated by L_i. From a theorem of Anissimov and Seifert [AS75], it follows in particular that if L_i is a regular language in A^*, the subgroup $\langle L_i \rangle$ is finitely generated. It is well known (see for example [Gil96]) that one can effectively compute a finite set of generators for $\langle L_i \rangle$. This is done using an extension of automata, in which inverse elements can be read by traversing the arrows backwards. We briefly discuss this construction now.

Consider an automaton $\mathcal{A} = (A, Q, \delta)$, which recognizes a language L^*, for $I = F = \{1\}$. If $\mathcal{A}$ is not deterministic or not co-deterministic, we first transform it into an automaton that is, using a process called Stallings foldings. This is described in [KM02], and amounts to collapsing arrows (and their states) that have the same label and that either both leave or both enter the same state.

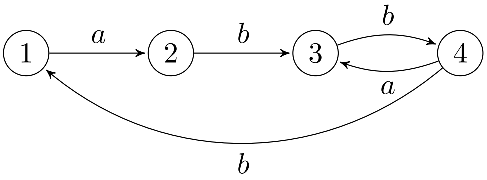
We can thus assume that $\mathcal{A}$ is deterministic and co-deterministic. Now, we allow every arrow in $\mathcal{A}$ to be traversed backwards, while reading the inverse of its label. That is, for each $(p, a, q) \in \delta$, we also get $(q, a^{-1}, p) \in \delta$. This automaton recognizes $\langle L \rangle$, for $I = F = \{1\}$. Now, identify a spanning tree in $\mathcal{A}$ with root 1 and with all edges directed away from this root. Consider all arrows (p, a, q) of $\mathcal{A}$ that are not part of the spanning tree. By definition of a spanning tree, there exist paths $(1, u, p)$ and $(1, v, q)$, for some $u, v \in A^*$ in the spanning tree. Define $g = uav^{-1}$. We claim that $\langle L \rangle$ is generated by the finite set of all such g's.

Clearly, such a g is in $\langle L \rangle$, since it is the label of a path from 1 to 1. For the other direction, let $w \in L$. Then, there is a run from 1 to 1 labeled w. This run can be written as a product of g's, by repeatedly going back to 1. More precisely, decompose w as $w = p_1 a_1 p_2 a_2 \cdots p_n$, where the p_i's label edges in the spanning tree, and the a_i's label edges that are not. By definition of a spanning tree, there is only one path from 1 to the state where the arrow labeled a_1 leaves, and it follows that $p_1 = u_1$. Similarly, for $1 < i < n$, there is only one path in the spanning tree from the state where the arrow labeled a_i enters, and the state where the arrow labeled a_{i+1} leaves. It follows that $p_{i+1} = v_i^{-1} u_{i+1}$. Note that p_n goes from the state that a_n enters to state 1, and thus is v_n^{-1}. Hence,

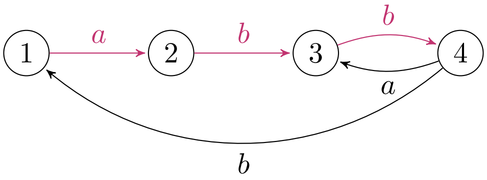
$$w = u_1 a_1 v_1^{-1} u_2 a_2 v_2^{-1} \cdots u_n a_n v_n^{-1} = g_1 \cdots g_n.$$

Let us illustrate this construction on an example.

Example 3.5. Consider the language $L = (abb(ab)^*b)^*$. As we saw above, its closure is $\langle abb(ab)^*b \rangle$. The language L itself is recognized by the following automaton $\mathcal{A}$, for $I = F = \{1\}$. Note that $\mathcal{A}$ is deterministic and co-deterministic.



In order to find a finite set of generators for $\langle abb(ab)^*b \rangle$, we first identify the following spanning tree in $\mathcal{A}$ with root 1 and with all edges directed away from this root.



We now look at the arrows that are not in the spanning tree. The one from state 4 to state 3, labeled a, gives rise to the generator $abb \cdot a \cdot b^{-1}a^{-1}$, and the arrow from state 4 to state 1, labeled b, gives rise to the generator $abb \cdot b$. By the above, $\langle abb(ab)^*b \rangle$ should be equal to $\langle abbab^{-1}a^{-1}, abbb \rangle$. Indeed, it is clear that $abbb \in \langle abb(ab)^*b \rangle$. Also, $abbab^{-1}a^{-1} = abbabb \cdot (abbb)^{-1} \in \langle abb(ab)^*b \rangle$, thus $\langle abbab^{-1}a^{-1}, abbb \rangle \subseteq \langle abb(ab)^*b \rangle$. Conversely, for every n, we have that $abb(ab)^n b = (abbab^{-1}a^{-1})^n \cdot abbb$, and it follows that $\langle abb(ab)^*b \rangle \subseteq \langle abbab^{-1}a^{-1}, abbb \rangle$.

3.2.2 Decidability of G-separability and a construction of a separator

The following theorem uses the result from [RZ93] to show that the closures in the free group (rather than the closures in the free profinite group) already suffice to test whether two regular languages are G-separable.

Theorem 3.6. *Let L_1 and L_2 be regular languages. Then, L_1 and L_2 are G-separable if and only if $\overline{L_1} \cap \overline{L_2} = \varnothing$, where the topological closures are taken in the free group.*

Proof. By Theorem 2.17, two regular languages over A are G-separable if and only if their closures in the free profinite group $\widehat{F}_{\mathsf{G}}(A)$ have an empty intersection. This implies that if L_1 and L_2 are G-separable, their closures in the free group, in particular, have an empty intersection. Let us now prove the other direction.

Since every regular language can effectively be written as a finite union of simple sets, it follows from Lemma 2.7, that we can assume that L_1 and L_2 are simple sets. As we saw above, the closures of L_1 and L_2 in the free group, in this case, are

$$\begin{aligned}\overline{L_1} &= \langle L_{1,0}\rangle u_1 \langle L_{1,1}\rangle u_2 \cdots \langle L_{1,n}\rangle, \quad \text{and}\\ \overline{L_2} &= \langle L_{2,0}\rangle v_1 \langle L_{2,1}\rangle v_2 \cdots \langle L_{2,m}\rangle.\end{aligned}$$

We denote $\langle L_{1,i}\rangle$ by I_i and $\langle L_{2,j}\rangle$ by J_j, so that

$$\begin{aligned}\overline{L_1} &= I_0 u_1 I_1 u_2 \cdots I_n, \quad \text{and}\\ \overline{L_2} &= J_0 v_1 J_1 v_2 \cdots J_m.\end{aligned}$$

From $\overline{L_1}$ and $\overline{L_2}$, we define an auxiliary product P of finitely generated subgroups and a specific word w, such that $\overline{L_1} \cap \overline{L_2} = \varnothing$ if and only if w is outside the auxiliary product. If $\overline{L_1} \cap \overline{L_2} = \varnothing$, we then have by the alternative formulation of Theorem 3.3, that there is a finite group that separates w from P. We will then proceed by arguing that any group that separates w from P, will also separate L_1 from L_2.

Define the auxiliary product P as follows.

$$\begin{aligned}P := (I_0)(u_1 I_1 u_1^{-1})(u_1 u_2 I_2 u_2^{-1} u_1^{-1}) \cdots (u_1 \cdots u_n I_n u_n^{-1} \cdots u_1^{-1})(u_1 \cdots u_n J_m u_n^{-1} \cdots u_1^{-1})\\ (u_1 \cdots u_n v_m^{-1} J_{m-1} v_m u_n^{-1} \cdots u_1^{-1}) \cdots (u_1 \cdots u_n v_m^{-1} \cdots v_1^{-1} J_0 v_1 \cdots v_m u_n^{-1} \cdots u_1^{-1}).\end{aligned}$$

This is a product of conjugated subgroups, each of which is indicated by brackets. A conjugated finitely generated subgroup is again finitely generated, for example by the conjugated generators of the original subgroup. Thus, P is a product of finitely generated subgroups.

Clearly,

$$P = I_0 u_1 I_1 u_2 \cdots I_{n-1} u_n I_n \cdot J_m v_m^{-1} J_{m-1} v_{m-1}^{-1} \cdots J_2 v_2^{-1} J_1 v_1^{-1} J_0 \cdot v_1 \cdots v_m u_n^{-1} \cdots u_1^{-1}.$$

Note that

$$1 \notin I_0 u_1 I_1 u_2 \cdots I_n \cdot J_m \cdots v_2^{-1} J_1 v_1^{-1} J_0 \;\Leftrightarrow\; v_1 \cdots v_m u_n^{-1} \cdots u_1^{-1} \notin P. \tag{3.1}$$

Recall that $\overline{L_1} = I_0u_1I_1u_2 \cdots I_n$ and $\overline{L_2} = J_0v_1J_1v_2 \cdots J_m$. We also have the following equivalence,

$$\overline{L_1} \cap \overline{L_2} = \varnothing \ \Leftrightarrow \ 1 \notin I_0u_1I_1u_2 \cdots I_n \cdot J_mv_m^{-1} \cdots v_2^{-1}J_1v_1^{-1}J_0, \tag{3.2}$$

as can be seen as follows. First, suppose there is $x \in \overline{L_1} \cap \overline{L_2}$. Then, there exist $j_j \in J_j$, for $j = 0, \ldots, m$, such that $x = j_0v_1j_1 \cdots v_mj_m$. Thus, $x^{-1} = (j_0v_1j_1 \cdots v_mj_m)^{-1}$, which is in $J_mv_m^{-1} \cdots J_1v_1^{-1}J_0$. Then, indeed,

$$1 = xx^{-1} \in I_0u_1I_1u_2 \cdots I_n \cdot J_mv_m^{-1} \cdots J_1v_1^{-1}J_0.$$

Conversely, suppose that $1 \in I_0u_1I_1u_2 \cdots I_n \cdot J_mv_m^{-1} \cdots J_1v_1^{-1}J_0$. Then, there exist $i_i \in I_i$, for $i = 0, \ldots, n$, and $j_j \in J_j$, for $j = 0, \ldots, m$, such that

$$1 = i_0u_1i_1u_2 \cdots i_n \cdot j_mv_m^{-1} \cdots j_1v_1^{-1}j_0.$$

Then, $j_0v_1j_1v_2 \cdots j_m = i_0u_1i_1u_2 \cdots i_n$, thus $\overline{L_1} \cap \overline{L_2} \neq \varnothing$.

Combining (3.1) and (3.2) yields the following equivalence,

$$\overline{L_1} \cap \overline{L_2} = \varnothing \ \Leftrightarrow \ v_1 \cdots v_mu_n^{-1} \cdots u_1^{-1} \notin P.$$

Assume that $\overline{L_1} \cap \overline{L_2} = \varnothing$. Then, it follows from the alternative formulation of Theorem 3.3 that there exists a finite group G with canonical morphism $\varphi : FG(A) \to G$ that separates the word $w := v_1 \cdots v_mu_n^{-1} \cdots u_1^{-1}$ from the product P.

Claim. The morphism $\varphi : FG(A) \to G$, which separates w from P, also separates the language L_1 from the language L_2.

To prove this claim, suppose there is $x \in \varphi(L_1) \cap \varphi(L_2)$. In particular, $x \in \varphi(\overline{L_1}) \cap \varphi(\overline{L_2})$. This means that there exist $i_i \in I_i$, for $i = 0, \ldots, n$, and $j_j \in J_j$, for $j = 0, \ldots, m$, such that $\varphi(i_0u_1i_1 \cdots u_ni_n) = \varphi(j_0v_1j_1 \cdots v_mj_m)$.

The following element is in $\varphi(P)$

$$\begin{aligned}
\varphi(i_0u_1i_1 \cdots u_ni_n \cdot j_m^{-1}v_m^{-1} \cdots v_1^{-1}j_0^{-1} \cdot v_1 \cdots v_mu_n^{-1} \cdots u_1^{-1}) &= \\
\varphi(i_0u_1i_1 \cdots u_ni_n) \cdot \varphi(j_m^{-1}v_m^{-1} \cdots v_1^{-1}j_0^{-1}) \cdot \varphi(v_1 \cdots v_mu_n^{-1} \cdots u_1^{-1}) &= \\
\varphi(i_0u_1i_1 \cdots u_ni_n) \cdot \varphi(j_0v_1j_1 \cdots v_mj_m)^{-1} \cdot \varphi(v_1 \cdots v_mu_n^{-1} \cdots u_1^{-1}).&
\end{aligned}$$

Since $\varphi(i_0u_1i_1 \cdots u_ni_n) = \varphi(j_0v_1j_1 \cdots v_mj_m)$, this gives that

$$\varphi(v_1 \cdots v_mu_n^{-1} \cdots u_1^{-1}) = \varphi(w) \in \varphi(P),$$

which is a contradiction. Hence, $\varphi : FG(A) \to G$ indeed separates the languages L_1 and L_2 (and gives $\varphi_G^{-1}\restriction_{A^*} (\varphi_G(L_1))$ as a separating group language), which proves the claim.

Thus, if $\overline{L_1} \cap \overline{L_2} = \varnothing$, where the topological closures are taken in the free group, L_1 and L_2 are G-separable, and a group language separating them can effectively be constructed. □

Since the closure of a regular language in the free group can be computed, we obtain the following corollary.

Corollary 3.7. *It is decidable whether two given regular languages are* G*-separable.*

Remark. For two G-separable languages L_1 and L_2, we obtain a description of a G-separator from [AS05]. Indeed, we saw in the claim in the proof of Theorem 3.6 that any finite group that separates the word w and the product P, as constructed from L_1 and L_2, will also separate L_1 and L_2. If L_1 and L_2 are G-separable, then applying Theorem 3.4 to this w and P therefore yields a finite group that separates L_1 and L_2.

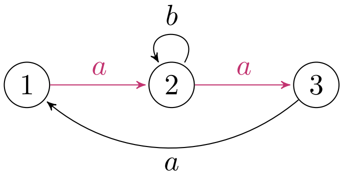
3.2.3 Closures in the free monoid

One can also endow the free monoid A^* with the topology induced from the free profinite group. This topology on the free monoid was introduced in [Reu79, Reu81]. It is the coarsest topology that makes every monoid morphism from A^* onto a finite group (endowed with the discrete topology) continuous. One can wonder whether a similar statement as Theorem 3.6 still holds when the topological closures are taken in the free monoid. In Example 3.8, we will show that this is not the case: there exist languages for which the closures in the free monoid do not intersect, while the closures in the free group do intersect.

The closure in the free monoid of a regular language L, denoted by $cl(L)$, is $cl(L) = \overline{L} \cap A^*$, where $\overline{L}$, as before, denotes the closure in the free group. For a simple set L, this means that $cl(L) = \langle L_0 \rangle u_1 \langle L_1 \rangle \cdots u_n \langle L_n \rangle \cap A^*$.

The following example shows that there exist languages for which the closures in the free monoid do not intersect, while the closures in the free group do intersect.

Example 3.8. Let $L_1 = (ab^*aa)^*b$ and $L_2 = (abb(ab)^*b)^*$. Then, $\overline{L_1} = \langle ab^*aa \rangle b$, and $\overline{L_2} = \langle abb(ab)^*b \rangle$. In Example 3.5, we saw that $\overline{L_2} = \langle abbab^{-1}a^{-1}, abbb \rangle$. In the same way, we can find a finite set of generators for the subgroup $\langle ab^*aa \rangle$. To this end, fix the following spanning tree.



The construction from [Gil96] gives that $\overline{L_1} = \langle ab^*aa \rangle b = \langle aba^{-1}, aaa \rangle b$.

For both L_1 and L_2, we have that $cl(L_i) = L_i$. This is not always the case, but is a consequence of the fact that the minimal automata of L_1 and L_2 are deterministic and co-deterministic. See for example [Reu79].

Note that $cl(L_1) \cap cl(L_2) = (ab^*aa)^*b \cap (abb(ab)^*b)^* = \varnothing$. One way to see this, is from the fact that $(ab^*aa)^*b \cap (\varepsilon \cup A^*bb) = \varnothing$, while $(abb(ab)^*b)^* \subseteq \varepsilon \cup A^*bb$. However, their closures in the free group are not disjoint. Indeed, the element $aba^{-1}b$ is in $\overline{L_1}$, but also in $\overline{L_2}$, since $aba^{-1}b = (abbab^{-1}a^{-1})^{-1} \cdot abbb$.

Hence, an empty intersection of the closures of two languages in the free monoid does not imply that the intersection of their closures in the free group is empty.

Chapter 4

Piecewise testable languages

4.1 Characterizations of piecewise testable languages 48
4.1.1 Logical characterization 48
4.1.2 Algebraic characterization 50
4.1.3 Graphical characterization 51
4.2 Separation by piecewise testable languages 51
4.2.1 PT-indistinguishable pairs of states 52
4.2.2 Common patterns 53
4.2.3 A common pattern yields PT-indistinguishability 54
4.2.4 PT-indistinguishability stems from a common pattern 55
4.2.5 Intermezzo: an alternative method 61
4.2.6 Separation theorem for piecewise testable languages 63
4.3 Complexity of PT-separability 64

In this chapter, which is based on the papers [RZ13, PvRZ13b, CMM$^+$14], we study the separation problem for the class of piecewise testable languages. This class was first introduced in [Sim72], and consists of those languages for which membership of a word in the language is determined by the *pieces*, or scattered subwords, of the word up to a certain length.

In terms of logic, this class can be defined as the fragment of first-order logic that consists of all formulas that are boolean combinations of formulas that have a quantifier prefix of existential quantifiers, and do not use any other quantifiers. We explain this in Section 4.1.1. This class is thus in one of the lower levels of the quantifier alternation hierarchy for first-order logic. However, it is still a challenging class to study, and has indeed been extensively studied in the last decades.

It has been shown in [AZ97, ACZ08] that the variety corresponding to this class has computable pointlike sets, which yields that the separation problem is decidable. In [AZ97], an algorithm to decide the separation problem was given. This algorithm runs in polynomial time with respect to the size of the automaton, and exponential time with respect to the alphabet. This approach only gives a yes/no answer.

Our motivation for studying the separation problem for this class, while it was already known to be decidable, was to find a combinatorial proof of the decidability, that does not use any profinite theory. Also, we wanted to obtain a description of a potential separator, rather than just a yes/no answer to the problem. We obtained these results by computing, from the input languages, a bound on the length of the pieces that are relevant for separability. This gives a description of a separator, if it exists. Furthermore, we exhibit forbidden patterns in the automaton recognizing the input languages. We show that certain paths of the same shape witness non-separability. This approach yields a better complexity result than the approach from [AZ97]: we obtain an algorithm that runs in polynomial time with respect to both the size of the automaton and the size of the alphabet.

We also provide an alternative combinatorial method to prove decidability of the separation problem, which uses Simon's Factorization Forest theorem. This method does not work by bounding the parameter (and does not give a description of a potential separator), but it works by showing that languages that are not separable contain sequences of words of a similar shape. For regular languages, these sequences give again the forbidden patterns on the automaton. However, as observed very recently in [CM14], this method also gives a criterion for non-separability of input languages that are not regular.

Recently, Czerwiński et. al. [CMM13] studied the separation problem for piecewise testable languages out of interest in applications in database theory. In this paper, an algorithm to decide separability by piecewise testable languages is also provided, with the same complexity, using different proof techniques. A bound on the length of the relevant pieces is not provided in this paper. Very recently, a construction to build a separating piecewise testable language from a finite so-called alternating tower was provided in [HJM14], as well as a bound on the length of the longest such tower in case the languages are separable.

In Section 4.1, we introduce the class of piecewise testable languages. We present our approach and results to the separation problem for this class in Section 4.2. Finally, in Section 4.3, we discuss the complexity of the algorithm that follows from our approach.

4.1 Characterizations of piecewise testable languages

4.1.1 Logical characterization

The class $\Sigma_1(<)$ is the class of $\mathsf{FO}(<)$-formulas of the form

$$\exists x_1 \dots \exists x_k.\ \varphi(x_1, \dots, x_k), \tag{4.1}$$

where $x_1, \dots, x_k$ are first-order variables and where φ is *quantifier-free*. In the formula φ, the only predicates allowed are the linear order $<$ and the alphabetical predicates. See also Section 1.5 for more information about first-order logic interpreted on words. This class is given the name $\Sigma_1(<)$ since only one block of the same quantifiers is allowed (there is no alternation between different types of quantifiers, hence the '1'), and the formulas start with a block of existential quantifiers (hence the 'Σ').

The class of all boolean combinations of $\Sigma_1(<)$-formulas is denoted by $\mathcal{B}\Sigma_1(<)$. The *rank* of a $\mathcal{B}\Sigma_1(<)$-formula is the size of the largest block of quantifiers present in the formula. For instance, the $\Sigma_1(<)$-formula in (4.1) has rank k.

Formulas in $\mathcal{B}\Sigma_1(<)$ can express the presence or absence of scattered subwords in words. We call these scattered subwords *pieces*, that is, we say that a word u is a *piece* of a word $v \in A^*$, denoted by

$$u \lhd v,$$

if there exist letters $a_1, \ldots, a_k \in A$ such that

$$u = a_1 \cdots a_k, \text{ and } v \in A^* a_1 A^* \cdots a_k A^*.$$

For instance, ab is a piece, of size (or length) 2, of $bb\underline{\boldsymbol{a}}ac\underline{\boldsymbol{b}}a$. The language $A^*a_1A^* \cdots a_kA^*$ is the set of all words of which $a_1 \cdots a_k$ is a piece, that is, the set $\{w \in A^* \mid a_1 \cdots a_k \lhd w\}$. We call it a *piece language*, and we call k its *width*. This language is defined by the following $\Sigma_1(<)$-formula of rank k,

$$\exists x_1 \ldots \exists x_k . \Big(\bigwedge_{i<k} (x_i < x_{i+1}) \wedge \bigwedge_{i \leq k} a_i(x_i) \Big). \tag{4.2}$$

On the other hand, not every $\Sigma_1(<)$-formula defines a piece language. Consider for example the following $\Sigma_1(<)$-formula,

$$\exists x_1 \exists x_2 \, \exists x_3 \, \exists x_4 . \Big(x_1 < x_2 \wedge x_3 < x_4 \wedge a(x_1) \wedge b(x_2) \wedge c(x_3) \wedge \neg a(x_4) \Big),$$

which, if $A = \{a, b, c\}$, defines the language $(A^*aA^*bA^* \cap A^*cA^*bA^*) \cup (A^*aA^*bA^* \cap A^*cA^*cA^*)$. However, every $\Sigma_1(<)$-formula defines a combination (using union and intersection) of piece languages. To see this, one can first eliminate negations inside the quantifier free part of the $\Sigma_1(<)$-formula, by replacing $\neg(x < y)$ by $y \leq x$ and $\neg a(x)$ by $\bigvee_{b \neq a} b(x)$. Then put the quantifier-free formula φ in disjunctive normal form.

It follows that every $\mathcal{B}\Sigma_1(<)$-formula defines a boolean combination of piece languages. One can verify that starting from a $\mathcal{B}\Sigma_1(<)$-formula of rank k, one gets a boolean combination of piece languages of width at most k.

We write $w \lesssim_k w'$ if every $\Sigma_1(<)$-formula of quantifier rank k that is satisfied by w is also satisfied by w'. Equivalently, $w \lesssim_k w'$ if every piece of size at most k of w is also a piece of w'. Clearly, for each k, the relation $\lesssim_k$ is a preorder, and it is compatible with concatenation: if $u \lesssim_k u'$ and $w \lesssim_k w'$, then $uw \lesssim_k u'w'$. The equivalence relation $\sim_k$, induced by $\lesssim_k$, is defined as

$$w \sim_k w' \quad \Leftrightarrow \quad w \lesssim_k w' \text{ and } w' \lesssim_k w.$$

Thus, $w \sim_k w'$ when w and w' have the same pieces of size up to k. Clearly, the equivalence relation $\sim_k$ is a congruence and it has finite index, since there are only finitely many pieces of size k or less.

It is easy to see that a language can be defined by a $\mathcal{B}\Sigma_1(<)$-formula of rank k if and only if it is a union of $\sim_k$-classes. These observations give the following well-known statement.

Lemma 4.1. *Let L be a language and let $k \in \mathbb{N}$. Then, the following properties are equivalent.*

(1) L is defined by a $\mathcal{B}\Sigma_1(<)$-formula of rank at most k,

(2) L is a boolean combination of piece languages of width at most k,

(3) L is a union of $\sim_k$-classes.

Definition 4.2. A language L is *k-piecewise testable* ($\mathsf{PT}[k]$) if it satisfies the three equivalent properties of Lemma 4.1, and is *piecewise testable* (PT) if it is k-piecewise testable for some $k \in \mathbb{N}$.

Note that a $\mathsf{PT}[k]$-language is a union of $\sim_k$-classes. The original definition of piecewise testable languages, provided in [Sim72], was actually in terms of unions of $\sim_k$-classes. Note that for a language L, there is always a smallest (with respect to inclusion) $\mathsf{PT}[k]$-language that contains L, namely the language

$$[L]_k := \{w \in A^* \mid \exists u \in L.\ u \sim_k w\}.$$

It is easy to see that every finite language is PT. This follows from the fact that the $\sim_k$-class of a word whose length is strictly smaller than k only contains this word. A finite language is thus a union of $\sim_k$-classes for some k strictly greater than the length of the longest word in the language.

This implies that, in contrast to $\mathsf{PT}[k]$, there is not always a smallest PT language that contains L. Indeed, as the class PT is closed under boolean operations, removing one word from a PT language yields again a PT language, and so on.

For example, let $L = (ab)^*$. Let us first see that this language itself is not PT. For every k, the word $(ab)^k \in L$ contains every possible piece of length k. The same holds, for example, for $(ba)^k$, but this word is not in L. Therefore, L cannot be a union of $\sim_k$-classes for any k, thus L is not piecewise testable. Now consider a PT language that contains L. It is a $\mathsf{PT}[k]$-language for some k, and therefore should, amongst other $\sim_k$-equivalent words, also contain $(ba)^{\geq k}$ for this value of k. Increasing k gives a smaller and smaller PT language that still contains L.

4.1.2 Algebraic characterization

The class of piecewise testable languages is characterized algebraically by Simon's theorem [Sim75]. Let us briefly recall the relation $\sim_{\mathcal{J}}$, which is one of Green's relations. Let M be a monoid and let $s, t \in M$. Then,

$$s \sim_{\mathcal{J}} t \quad \Leftrightarrow \quad \text{there exist } u, v, w, x \in M, \text{ such that } s = utv \text{ and } t = wsx.$$

A monoid M is $\mathcal{J}$-trivial when, for all $s, t \in M$, $s \sim_{\mathcal{J}} t$ implies that $s = t$. That is, when the relation $\sim_{\mathcal{J}}$ is equal to the equality relation. Clearly, this property can be decided for a given finite monoid. The following theorem thus provides an effective characterization, and yields decidability of the membership problem for the class PT. There exist many proofs of this result, see for example [Pin84, ST88, Alm91, Klí11].

Theorem 4.3 (Simon's theorem)**.** *A language is piecewise testable if and only if its syntactic monoid is $\mathcal{J}$-trivial.*

The class of $\mathcal{J}$-trivial monoids is denoted by J. A monoid is in this class if and only if it satisfies the identities $u^\omega = u^{\omega+1}$ and $(uv)^\omega = (vu)^\omega$, if and only if it satisfies $v(uv)^\omega = (uv)^\omega = (uv)^\omega u$.

4.1.3 Graphical characterization

In [Ste85], Stern provided a characterization for the class of piecewise testable languages on the recognizing minimal automaton. From this characterization, an algorithm can be found to decide membership in this class, which runs in polynomial time with respect to the size of the alphabet and the number of states. This algorithm has later been improved in [Tra01b] to obtain a lower polynomial time complexity result.

To state Stern's criterion, we need some notation. Let $\mathcal{A} = (A, Q, \delta)$ be a DFA. For a state p, the *component* of p, denoted by $C(p)$, consists of all states accessible from p. That is, $C(p) = \{p\} \cup \{q \mid \exists u \in A^+.\ (p, u, q) \in \delta^*\}$. Stern defines the following order on the states of an automaton,

$$p \leq q \quad \Leftrightarrow \quad q \in C(p).$$

An automaton is called *acyclic* if there are no loops that visit more than one state. The restriction of $\mathcal{A}$ to a subalphabet $B \subseteq A$ is defined by $\mathcal{A}\restriction_B := (B, Q, \delta \cap (Q \times B \times Q))$.

Proposition 4.4 ([Ste85, Proposition 1.2])**.** *Let L be a language, accepted by its minimal automaton $\mathcal{A}$. The language L is piecewise testable if and only if $\mathcal{A}$ is acyclic, and, for every subalphabet $B \subseteq A$, every component in $\mathcal{A}\restriction_B$ has a unique maximal state.*

This result yields an algorithm to decide if a language is piecewise testable from its minimal automaton, which runs in polynomial time with respect to the size of the alphabet and the number of states. Note that this result solves the membership problem without using algebra.

4.2 Separation by piecewise testable languages

In Section 4.2.1, we define a relation on the set of pairs of states, of which we show in Lemma 4.6 that it captures all the relevant information concerning PT-separability. This relation works on a finite set, and it is therefore a convenient tool for studying the separation problem. The definition of this relation, however, is still very close to the definition of non-PT-separability, as we will see in the proof of Lemma 4.6.

In Section 4.2.2, we will introduce another relation on pairs of states, in terms of patterns occurring in the automaton. We use Sections 4.2.3 and 4.2.4 to show that these two relations are in fact equal, and that we are thus able to use the notion of patterns in order to decide separability. Furthermore, in order to prove the equality between the two relations on pairs of states, we will compute a bound on the size of the pieces that need to be considered to see whether two languages are PT-separable. An alternative method to prove this equality is described in Section 4.2.5. We summarize these results in Section 4.2.6 in Theorem 4.27, a separation theorem for PT. In Section 4.3 we prove that these patterns can be found in PTIME, with respect to the size of the automaton and the size of the alphabet.

4.2.1 PT-indistinguishable pairs of states

We will define a notion of PT-indistinguishable pairs of states. The idea behind the notion of indistinguishable pairs is explained in Section 2.2.3. The idea is the same as we have seen in Section 2.3.2 for the class K. For the class PT, we introduce the following relation on pairs of states of an automaton.

Definition 4.5. Let $\mathcal{A} = (A, Q, \delta)$ be an NFA, and let $(q_1, r_1), (q_2, r_2) \in Q^2$. Then,

$$(q_1, r_1) \approx_k (q_2, r_2) \quad \Leftrightarrow \quad \exists u_1, u_2 \in A^*.\ u_1 \sim_k u_2, (q_1, u_1, r_1) \in \delta^*, (q_2, u_2, r_2) \in \delta^*,$$

in which case the pairs (q_1, r_1) and (q_2, r_2) are called $\mathsf{PT}[k]$-*indistinguishable.* They are called PT-*indistinguishable* if for all $k \in \mathbb{N}$, $(q_1, r_1) \approx_k (q_2, r_2)$.

The subset of $Q^2 \times Q^2$ that consists of all $\mathsf{PT}[k]$-indistinguishable pairs of $\mathcal{A}$ is denoted by $\mathrm{I}_k^{\mathsf{PT}}[\mathcal{A}]$, or simply, $\mathrm{I}_k[\mathcal{A}]$. The set of all PT-indistinguishable pairs of $\mathcal{A}$ is denoted by $\mathrm{I}^{\mathsf{PT}}[\mathcal{A}]$, or $\mathrm{I}[\mathcal{A}]$.

By definition, we have $\mathrm{I}[\mathcal{A}] = \bigcap_k \mathrm{I}_k[\mathcal{A}]$. Since for all $k \in \mathbb{N}$, it holds that $\sim_{k+1} \subseteq \sim_k$, we also have $\approx_{k+1} \subseteq \approx_k$, and thus the following inclusions hold,

$$\mathrm{I}[\mathcal{A}] = \bigcap_{n \in \mathbb{N}} \mathrm{I}_n[\mathcal{A}] \subseteq \ldots \subseteq \mathrm{I}_{k+1}[\mathcal{A}] \subseteq \mathrm{I}_k[\mathcal{A}] \subseteq \ldots \subseteq \mathrm{I}_1[\mathcal{A}].$$

Note that the set $Q^2 \times Q^2$ is finite, thus there must be an index for which the sequence $(\mathrm{I}_k[\mathcal{A}])_{k \in \mathbb{N}}$ stabilizes. That is, there exists $\kappa \in \mathbb{N}$ such that for every $k \geq \kappa$, we have $\mathrm{I}[\mathcal{A}] = \mathrm{I}_k[\mathcal{A}]$. While the existence of κ is immediate from the definitions, computing a bound on κ is a more difficult problem. We will obtain such a bound in Section 4.2.4, as a byproduct of the proof that there is a certain pattern in the automaton, whenever two pairs of states are PT-indistinguishable. This bound depends on the number of states in the automaton recognizing the input languages and the size of the alphabet A.

Because of the connection between PT-separability and PT-indistinguishable pairs, stated in Lemma 4.6, computing a stabilization index κ is of particular interest when studying the separation problem. Indeed, we will see in Theorem 4.27 that every two languages recognized by $\mathcal{A}$ that are PT-separable are already $\mathsf{PT}[\kappa]$-separable.

The following lemma shows that the PT-indistinguishable pairs defined above indeed capture the right information about PT-separability.

Lemma 4.6. *Let $\mathcal{A} = (A, Q, \delta)$ be an NFA. Let $(q_1, r_1), (q_2, r_2)$ be pairs of states that determine languages L_1 and L_2. Then, for all $k \in \mathbb{N}$, $(q_1, r_1), (q_2, r_2)$ are $\mathsf{PT}[k]$-indistinguishable if and only if L_1 and L_2 are not $\mathsf{PT}[k]$-separable. Furthermore, $(q_1, r_1), (q_2, r_2)$ are PT-indistinguishable if and only if L_1 and L_2 are not PT-separable.*

Proof. Suppose $(q_1, r_1), (q_2, r_2)$ are $\mathsf{PT}[k]$-indistinguishable. Then, there are $u, v \in A^*$ such that $u \sim_k v$, $(q_1, u, r_1) \in \delta^*$ and $(q_2, v, r_2) \in \delta^*$. Since a $\mathsf{PT}[k]$-language is a union of $\sim_k$-classes, a $\mathsf{PT}[k]$-language L that separates L_1 from L_2 must contain $[L_1]_k$. It follows that $L \cap L_2 \neq \varnothing$, thus the languages are not $\mathsf{PT}[k]$-separable. For the converse direction, suppose L_1 and L_2 are not $\mathsf{PT}[k]$-separable. Then in particular, $[L_1]_k$ is not a separator. Clearly $L_1 \subseteq [L_1]_k$, and thus $[L_1]_k \cap L_2 \neq \varnothing$. This means there are $u \in L_1$, $v \in L_2$ such that $u \sim_k v$, thus $(q_1, r_1), (q_2, r_2)$ are $\mathsf{PT}[k]$-indistinguishable. The second statement immediately follows from the first statement. □

Thus, Lemma 4.6 shows that it is enough to compute the set $\mathrm{I}[\mathcal{A}]$ of PT-indistinguishable pairs to decide PT-separability for two languages recognized by $\mathcal{A}$. In the following sections, we show how to obtain this set by analyzing the graph of $\mathcal{A}$. In [RZ13], we also studied the graph of $\mathcal{A}$ in order to decide PT-separability. An important ingredient of our proofs there was Simon's Factorization Forest theorem [Sim90]. We discuss the approach taken in [RZ13] as an intermezzo in Section 4.2.5.

In the following sections, we will more or less follow the proof technique of [PvRZ13b] and only use pumping arguments. This approach is also described in [CMM+14] under the name of NFA graph algorithm. In that paper, we also provide two other ways to compute the set $\mathrm{I}[\mathcal{A}]$: we present an algorithm based on games, and a top-down fixpoint algorithm that starts with all pairs in $Q^2 \times Q^2$ and removes pairs until a fixpoint is reached and only pairs in $\mathrm{I}[\mathcal{A}]$ remain.

4.2.2 Common patterns

We will now define another relation on pairs of states, based on common patterns occurring in the automaton. In Sections 4.2.3 and 4.2.4, we will prove that this relation is actually equal to the relation of being PT-indistinguishable. By Lemma 4.6, this means that we are able to capture non-PT-separability using these patterns in the automaton. Furthermore, in Section 4.3, we will show that the pairs having a common pattern can be found in PTIME with respect to the size of the automaton and the size of the alphabet.

Definition 4.7. Let $\mathcal{A}$ be an NFA over A. For $u_0, \dots, u_p \in A^*$ and *nonempty* subalphabets $B_1, \dots, B_p \subseteq A$, let $\vec{u} = (u_0, \dots, u_p)$ and $\vec{B} = (B_1, \dots, B_p)$. We call $(\vec{u}, \vec{B})$ a *factorization pair*. Let q and r be states of $\mathcal{A}$. A $(\vec{u}, \vec{B})$-*path* between q and r is a path of the form shown in Figure 4.1.

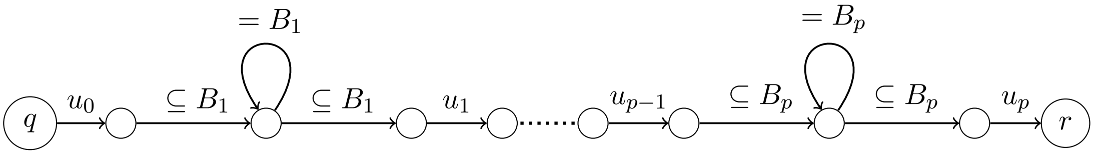


Figure 4.1: A $(\vec{u}, \vec{B})$-path between q and r.

If (q, r) has a $(\vec{u}, \vec{B})$-path, then it is clear that $\{w \mid q \xrightarrow{w} r\}$ contains a language of the form

$$u_0(x_1 y_1^* z_1)u_1 \cdots u_{p-1}(x_p y_p^* z_p)u_p,$$

with $\mathsf{alph}(x_i) \cup \mathsf{alph}(z_i) \subseteq \mathsf{alph}(y_i) = B_i$.

Definition 4.8. We say that two pairs of states $(q_1, r_1), (q_2, r_2)$ have a *common pattern* if there exists a factorization pair $(\vec{u}, \vec{B})$ such that both (q_1, r_1) and (q_2, r_2) have a $(\vec{u}, \vec{B})$-path.

Example 4.9. Consider the NFA depicted in Figure 4.2, consisting of two parts. The pairs of states $(q_1, r_1), (q_2, r_2)$ have a common pattern, since they both have a $(\vec{u}, \vec{B})$-path, for $\vec{u} = (\varepsilon, c, \varepsilon)$ and $\vec{B} = (\{a, b\}, \{a\})$.

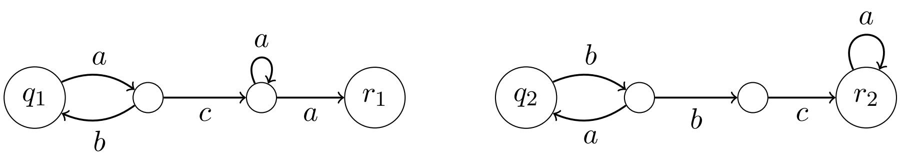


Figure 4.2: A common pattern for (q_1, r_1) and (q_2, r_2): $\vec{u} = (\varepsilon, c, \varepsilon)$, $\vec{B} = (\{a, b\}, \{a\})$.

As mentioned before, we will show in Sections 4.2.3 and 4.2.4 that two pairs of states are PT-indistinguishable if and only if they have a common pattern. Let us for now just mention that the notion of common pattern is a generalization of Stern's criterion from Section 4.1.3, in the sense that a language $L(\mathcal{A}, I, F)$ satisfies Stern's criterion if and only if there is no pair in $(I \times F) \times (I \times Q \backslash F)$ that has a common pattern.

4.2.3 A common pattern yields PT-indistinguishability

In the following proposition, we prove that if two pairs of states have a common pattern, then they are PT-indistinguishable.

Proposition 4.10. *Let $\mathcal{A}$ be an NFA, and let (q_1, r_1) and (q_2, r_2) be two pairs of states in $\mathcal{A}$ that have a common pattern. Then, (q_1, r_1) and (q_2, r_2) are PT-indistinguishable.*

Proof. Let $(\vec{u}, \vec{B})$ be a common pattern for (q_1, r_1) and (q_2, r_2). By definition, $\{w \mid q_1 \xrightarrow{w} r_1\}$ contains a language of the form

$$u_0(x_1 y_1^* z_1)u_1 \cdots u_{p-1}(x_p y_p^* z_p)u_p,$$

with $\mathsf{alph}(x_i) \cup \mathsf{alph}(z_i) \subseteq \mathsf{alph}(y_i) = B_i$, and $\{w' \mid q_2 \xrightarrow{w'} r_2\}$ contains a language of the form

$$u_0(x'_1 {y'_1}^* z'_1)u_1 \cdots u_{p-1}(x'_p {y'_p}^* z'_p)u_p,$$

with $\mathsf{alph}(x'_i) \cup \mathsf{alph}(z'_i) \subseteq \mathsf{alph}(y'_i) = B_i$. For all $k \in \mathbb{N}$, we define the words

$$\begin{aligned} w_k &= u_0(x_1 y_1^k z_1)u_1 \cdots u_{p-1}(x_p y_p^k z_p)u_p, \\ w'_k &= u_0(x'_1 {y'_1}^k z'_1)u_1 \cdots u_{p-1}(x'_p {y'_p}^k z'_p)u_p. \end{aligned}$$

From the above, it follows that $(q_1, w_k, r_1), (q_2, w'_k, r_2) \in \delta^*$. It remains to show that $w_k \sim_k w'_k$. Observe that for all i, $x_i y_i^k z_i$ and $x'_i {y'_i}^k z'_i$ both contain precisely all words from $B_i^{\leq k}$ as pieces of size up to k. It follows that $x_i y_i^k z_i \sim_k x'_i {y'_i}^k z'_i$. Using that $\sim_k$ is a congruence then yields that $w_k \sim_k w'_k$. □

4.2.4 PT-indistinguishability stems from a common pattern

While the proof of the fact that a common pattern for two pairs of states implies that these pairs are PT-indistinguishable worked in a rather straightforward way by unfolding the B-loops of the pattern, the converse direction, which we will prove in this section, is more difficult.

From the fact that two pairs of states are PT-indistinguishable, we obtain two sequences of words that are pairwise $\sim_k$-equivalent, for increasing values of k. We then need to mold these words into a similar shape, in order to be able to exhibit a common pattern for the pairs of states. If one would study this problem in a profinite setting, standard compactness arguments would give the existence of two subsequences that are still pairwise $\sim_k$-equivalent, for increasing values of k, and that converge to a profinite word. However, we want to exhibit a common pattern on the automaton. In [RZ13], we obtained this result by manually extracting subsequences of words with a similar shape, using Simon's Factorization Forest theorem [Sim90]. We will come back to this approach in Section 4.2.5, since it yields an interesting criterion for non-PT-separability of *any* two input languages.

Here, we choose to first describe the approach of [PvRZ13b], which catches the possible shapes in which a word can be molded in the notion of *template*. Roughly, we show that the templates in which a word fits depend on the pieces of the word. We provide a bound on the length of the pieces that need to be considered in this respect. Finally, from the NFA, we compute a bound such that words that have the same pieces of this length, must fit in the same way in a certain template. It then follows from pumping arguments that we can exhibit a common pattern in the automaton, along which these words are read.

The advantage of the approach that we present here, is that we also obtain a bound on the size of the pieces that are relevant for PT-separability.

In Proposition 4.21, we prove that whenever two pairs of states are PT$[k]$-indistinguishable for a sufficiently large k, this will be witnessed by a common pattern, for these pairs, in the automaton. First, we introduce some terminology. We fix an arbitrary order $a_1 < \ldots < a_m$ on A.

Definition 4.11. Let $B = \{b_1, \ldots, b_n\} \subseteq A$ be a finite alphabet, ordered as $b_1 < \ldots < b_n$. Let $p \in \mathbb{N}$. A word $w \in B^*$ is called a *(B,p)-pattern* if $w \in (B^*b_1B^* \cdots b_nB^*)^p$. Or, in other words, a word w is a (B,p)-pattern if $(b_1 \cdots b_n)^p \lhd w \in B^*$.

For example, let $B = \{a, b, c\}$ with $a < b < c$. The word $bb\underline{\boldsymbol{a}}a\underline{\boldsymbol{b}}ab\underline{\boldsymbol{c}}c\underline{\boldsymbol{a}}c\underline{\boldsymbol{b}}aba\underline{\boldsymbol{c}}a$ is a $(B,2)$-pattern but not a $(B,3)$-pattern.

The following notion, of ℓ-template, aims to provide a tool to describe, in terms of properties of words, the $(\vec{u}, \vec{B})$-paths along which a word could potentially be read. The main idea behind these ℓ-templates is that they give a framework along which words can be decomposed in a way that is suitable for pumping arguments.

Definition 4.12. An *ℓ-template* is a sequence $T = t_1, \ldots, t_\ell$ of length ℓ, such that every t_i is either a letter from the alphabet A or is a nonempty subset of A. An ℓ-template is said to be *unambiguous* if for all $i \in \{1, \ldots, \ell - 1\}$, the pair t_i, t_{i+1} either consists of two letters, or two incomparable sets, or a set and a letter that is not included in the set.

For example, $T_1 = a, \{b, c\}, d, d, \{a\}$ is unambiguous, while $T_2 = \underline{\boldsymbol{b}}, \{\underline{\boldsymbol{b}}, c\}, d, \{a\}$ and $T_3 = \{a, b\}, \{b, \underline{\boldsymbol{c}}\}, \{\underline{\boldsymbol{c}}\}, a$ are not.

We want to use the pieces of a word to detect whether, and in which way, the word fits in a certain ℓ-template. It turns out that this is only possible for unambiguous ℓ-templates. Let us first make the meaning of 'fitting in an ℓ-template' more precise in the following definition.

Definition 4.13. A word $w \in A^*$ is a *p-implementation* of an ℓ-template $T = t_1, \dots, t_\ell$ if w can be decomposed as $w = w_1 \cdots w_\ell$, such that, for all i, either $t_i = w_i \in A$ or $t_i = B \subseteq A$ and w_i is a (B, p)-pattern.

For example, $abccbbcbdaaa = a \cdot \underline{\boldsymbol{bc}}c\underline{\boldsymbol{b}}b\underline{\boldsymbol{c}}b \cdot d \cdot \underline{\boldsymbol{aa}}a$ is a 2-implementation of the 4-template $T = a, \{b, c\}, d, \{a\}$, since $\underline{\boldsymbol{bc}}c\underline{\boldsymbol{b}}b\underline{\boldsymbol{c}}b$ is a $(\{b, c\}, 2)$-pattern and $\underline{\boldsymbol{aa}}a$ is a $(\{a\}, 2)$-pattern.

One word of warning about the notation: although the notations for ℓ-template and p-implementation look quite similar, the meaning of the natural number occurring in the notation is very different. The ℓ in ℓ-template stands for the *length* of the template, and the p in p-implementation indicates to which *power* the ordered subalphabets of an ℓ-template occur in the respective factors of a word.

In order to give some intuition about the reason to restrict to unambiguous ℓ-templates, consider the words $w_n = b(ab)^n$ and $v_n = (ab)^n$. For all n, $w_n \sim_n v_n$, as both words contain all possible pieces over $\{a, b\}$ of length up to n. The word w_n is an n-implementation of the ambiguous 2-template $T = \underline{\boldsymbol{b}}, \{a, \underline{\boldsymbol{b}}\}$, but v_n does not implement this template at all. Thus, for arbitrarily large n, there are words that are $\sim_n$-equivalent while one of them can be decomposed along the template, and the other one cannot.

Remark 4.14. Every ambiguous ℓ-template T gives rise to an unambiguous ℓ'-template T', with $\ell' < \ell$, in the following way: one simply merges every pair t_i, t_{i+1} that causes an ambiguity. If a word w is a p-implementation of T, then it will also be a p-implementation of T'.

For example, the ambiguous 4-template $T = \{a, b\}, \{b, \underline{\boldsymbol{c}}\}, \{\underline{\boldsymbol{c}}\}, a$ gives rise to the unambiguous 3-template $T' = \{a, b\}, \{b, c\}, a$. The word $w = \underline{\boldsymbol{abab}}a \cdot c\underline{\boldsymbol{b}}b\underline{\boldsymbol{cbc}} \cdot \underline{\boldsymbol{cc}} \cdot a$ is a 2-template of T, and also of T', as can be seen from $w = \underline{\boldsymbol{abab}}a \cdot c\underline{\boldsymbol{b}}b\underline{\boldsymbol{cbc}}cc \cdot a$.

If two pairs of states are PT-indistinguishable, then between both pairs of states, for an arbitrary size of pieces, words containing the same pieces of this size can be read. We will show that if two words contain the same pieces of a large enough size (depending on p, $|A|$ and ℓ), then, they are both p-implementations of a common unambiguous ℓ-template. For p large enough, this will allow us to exhibit, via pumping arguments, a common $(\vec{u}, \vec{B})$-path for the pairs of states. In the next lemma, we provide a bound on ℓ that will suffice in the search for this common ℓ-template.

Lemma 4.15. *Let $p \in \mathbb{N}$. Every word over the alphabet A is the p-implementation of some unambiguous ℓ-template, with $\ell < N_{A,p} = (|A|p)^{|A|}$.*

Proof. First note that if a word is a p-implementation of some ambiguous ℓ-template, then by

Remark 4.14, it is also a p-implementation of an unambiguous ℓ'-template, where $\ell' < \ell$. It thus suffices to prove that every word is the p-implementation of some (possibly ambiguous) ℓ-template for $\ell < N_{A,p}$. Note that a word is always a p-implementation of the ℓ-template which is just the sequence of its letters. Therefore, it suffices to prove the following claim.

Claim. If a word is a p-implementation of some ℓ-template with $\ell \geq N_{A,p}$, then it is also a p-implementation of an ℓ'-template with $\ell' < \ell$.

We will show this by induction on the size of the alphabet A. For $|A| = 1$, let T be an ℓ-template with $\ell \geq N_{A,p} = p$. Then, every t_i is either the letter a or the set $\{a\}$. If a word w is a p-implementation of this ℓ-template, then it is a concatenation of at least ℓ a's. Clearly, w is then also a p-implementation of the 1-template $\{a\}$.

Now suppose the claim is true for alphabet sizes up to n. Note that if a subtemplate $t_i, t_{i+1}, \ldots, t_j$ of T can be shrunk into a template of smaller length in such a way that every word that is a p-implementation of the subtemplate still is a p-implementation of the shrunken subtemplate, then T itself can be shrunk while keeping the same property.

Let A be of size $n+1$, ordered as $a_1 < \ldots < a_{n+1}$, and let T be an ℓ-template for $\ell \geq (|A|p)^{n+1}$. Then $T = R, S$, for $R = t_1, \ldots, t_{(|A|p)^{n+1}}$ and $S = t_{(|A|p)^{n+1}+1}, \ldots, t_\ell$. We will focus on the $(|A|p)^{n+1}$-template R and show that every word that is a p-implementation of R is a p-implementation of an ℓ'-template with $\ell' < (|A|p)^{n+1}$.

For $i = 1, \ldots, |A|p$, we define $R_i = t_{1+(i-1)(|A|p)^n}, \ldots, t_{i(|A|p)^n}$. That is, every R_i is an $(|A|p)^n$-template, and $R = R_1, \ldots, R_{|A|p}$. There are two cases. Either

(1) there is an R_i that uses an alphabet different from A, or,

(2) every R_i uses the alphabet A.

In the first case, the induction hypothesis on the size of the alphabet gives that R_i can be shrunk into a template of smaller length, without reducing the set of words that are p-implementations of it. In the second case, every R_i uses the alphabet A. In this case, R can be replaced by the 1-template A: if a word is a p-implementation of R, it contains at least $|A|p$ times the alphabet A and it follows that $(a_1 \cdots a_{n+1})^p$ is a piece of the word. Thus, it is a p-implementation of the 1-template A.

In both cases, it follows that the length of T can be shrunk in such a way that all words that are p-implementations of T are still p-implementations of the shorter template. □

In Lemma 4.18 and Lemma 4.19, we will show that the unambiguous templates, of which a word is an implementation, are related to the pieces of the word. We first introduce some terminology on the special pieces that we will use in the proofs of these lemma's.

Definition 4.16. Let $T = t_1, t_2, \ldots, t_\ell$ be an unambiguous ℓ-template and let $p \in \mathbb{N}$. We denote by $v_{T,p}$ the shortest word that is a p-implementation of T. That is, $v_{T,p} = v_1 \cdots v_\ell$, such that for all i,

$$v_i = \begin{cases} t_i & \text{if } t_i \in A, \\ (b_1 \cdots b_n)^p & \text{if } t_i = \{b_1, \ldots, b_n\} \subseteq A \text{ and } b_1 < \ldots < b_n. \end{cases}$$

Definition 4.17. Let $T = t_1, t_2, \ldots, t_\ell$ be an unambiguous ℓ-template. Consider $v_{T,1} = v_1 \cdots v_\ell$, the shortest word that is a 1-implementation of T. A word v is called *incompatible with* T when v is of the form $v = v_1 \cdots v_i \cdot u \cdot v_{i+1} \cdots v_\ell$, where $u = u_1 \cdots u_n$, such that every $u_j \in A$, and one of the following holds,

- t_i and t_{i+1} are letters
- t_i is a letter, t_{i+1} a set, and there exist j such that $u_j \notin t_{i+1}$
- t_i is a set, t_{i+1} a letter, and there exist j such that $u_j \notin t_i$
- t_i and t_{i+1} are sets, and there is no j such that $\{u_1, \ldots, u_{j-1}\} \subseteq t_i$ and $\{u_j, \ldots, u_n\} \subseteq t_{i+1}$.

Note that if $v = v_1 \cdots v_i \cdot u \cdot v_{i+1} \cdots v_\ell$ is incompatible with T, then there is u' such that $|u'| \leq 2$ and $v' = v_1 \cdots v_i \cdot u' \cdot v_{i+1} \cdots v_\ell$ is already incompatible with T.

Let us start by showing a condition on the pieces of a word that is necessary for the word to be a p-implementation of some unambiguous template.

Lemma 4.18. *Let $w \in A^*$ and let T be an unambiguous ℓ-template. If w is a p-implementation of T, then $v_{T,p} \lhd w$ and, for all pieces v such that $v \lhd w$, it holds that v is not incompatible with T.*

Proof. By definition of p-implementation, we have $w = w_1 \cdots w_\ell$ such that $w_i = t_i$ if $t_i \in A$, and w_i is a (B, p)-pattern if $t_i = B \subseteq A$. In the latter case, if $B = \{b_1, \ldots, b_n\}$ is ordered as $b_1 < \ldots < b_n$, we have $(b_1 \cdots b_n)^p \lhd w_i \in B^*$. Thus, $v_{T,p} \lhd w$.

Now suppose there is $v \in A^*$ that is incompatible with T and that is such that $v \lhd w$. By definition, v is of the form $v = v_1 \cdots v_i \cdot u \cdot v_{i+1} \cdots v_\ell$, where $v_1 \cdots v_\ell = v_{T,1}$, and u satisfies one of the conditions listed in Definition 4.17. We reason by embedding the word $v = v_1 \cdots v_i \cdot u \cdot v_{i+1} \cdots v_\ell$ as soon as possible, from the left to the right, in $w = w_1 \cdots w_\ell$. Using that T is an unambiguous template, we show by induction on j that for every j, we cannot entirely embed v_j in w_{j-1}. If $t_{j-1} \subseteq A$, it follows from the unambiguity of T that v_j contains at least one letter that is not contained in t_{j-1}, thus v_j cannot be a piece of w_{j-1} which is in ${t_{j-1}}^*$. Else, $t_{j-1} \in A$ and $w_{j-1} = t_{j-1}$. By the induction hypothesis, v_{j-1} is not read as a piece of w_{j-2} and thus w_{j-1} is already used to embed v_{j-1}. In both cases, v_j cannot be entirely embedded in w_{j-1}.

Hence, $v_1 \cdots v_i$ is not a piece of the prefix $w_1 \cdots w_{i-1}$, and at least part of v_i and all of $u \cdot v_{i+1} \cdots v_\ell$ must occur to the right of w_{i-1}, that is, $u \cdot v_{i+1} \cdots v_\ell \lhd w_i \cdots w_\ell$. As the rules for unambiguity are symmetric, we also have that $v_{i+1} \cdots v_\ell$ is not a piece of the suffix $w_{i+2} \cdots w_\ell$, thus at least part of v_{i+1} and all of $v_1 \cdots v_i \cdot u$ must occur to the left of w_{i+2}, that is, $v_1 \cdots v_i \cdot u \lhd w_1 \cdots w_{i+1}$. Then, u must lie in $w_i \cdot w_{i+1}$. By definition of w_i, w_{i+1} and u, this gives a contradiction. It follows that there is no piece of w that is incompatible with T. □

The following lemma is a converse statement: it considers a stronger condition on the pieces of a word, and states that this is a sufficient condition for the word to be a p-implementation of an unambiguous ℓ-template.

Lemma 4.19. *Let $w \in A^*$ and let T be an unambiguous ℓ-template. If $v_{T,p+2} \lhd w$ and, for all pieces v such that $v \lhd w$, it holds that v is not incompatible with T, then w is a p-implementation of T.*

Proof. Besides $v_{T,p+2}$, we make use of the words $v_{T,1}$ and $v_{T,p}$, whose factors we denote here by $v_1^1, \dots, v_1^\ell$ resp. $v_p^1, \dots, v_p^\ell$ (note that the exponents do not denote powers). We will provide a decomposition $w = w_1 \cdots w_\ell$ and show that it witnesses the fact that w is a p-implementation of T. First, we define the factors $w_1, \dots, w_\ell$ inductively and then show that they indeed form a decomposition of w that has the desired properties.

Assume that the factors $w_1, \dots, w_{i-1}$ are defined, and that w_{s_i} is the remaining suffix of w, that is, $w = w_1 \cdots w_{i-1} \cdot w_{s_i}$. Now consider t_i. This is either a letter or a subset of A. If t_i is a letter, we define w_i as the first letter of w_{s_i}. If t_i is a subset of A, we define w_i as the largest prefix of w_{s_i} for which $\mathsf{alph}(w_i) \subseteq t_i$.

We will show by induction that these factors indeed form a decomposition of w that is a p-implementation of T.

As an illustration, we first treat the case that $i = 1$. First consider the case that $t_1 \in A$. Suppose that the first letter of w is $b \neq t_1$. Then, it follows from $v_{T,1} \lhd w$ that $b \cdot v_{T,1}$ is a piece of w, while it is incompatible with T. This would contradict the assumptions. Thus, the first letter of $w = w_{s_1}$ is equal to t_1. Now consider the case that $t_1 \subseteq A$. Then w_1 is the largest prefix of w such that $\mathsf{alph}(w_1) \subseteq t_1$. Denote the first letter after w_1 by c. By assumption, we have $v_{T,p+2} \lhd w$. Suppose that v_p^1 is not a piece of w_1. Then, in particular, a factor v_1^1 of $v_{T,p+2}$ must occur after c. This means that $c \cdot v_{T,1} \lhd w$. But, by definition, $c \notin t_1$, thus this piece is incompatible with T. It follows that $v_p^1 \lhd w_1$.

Now suppose that the factors $w_1, \dots, w_{i-1}$, defined as above, are such that for all these factors, $v_p^j \lhd w_j$. If t_j is a set, then by construction $w_j \in {t_j}^*$. Thus, this gives that $w_1, \dots, w_{i-1}$ is a p-implementation of $t_1, \dots t_{i-1}$. By unambiguity of T, the factor w_{i-1} cannot contain $v_p^{i-1} \cdot v_1^i$, since w_{i-1} is either a letter or is in ${t_{i-1}}^*$. It then follows that $v_{p+1}^i \cdot v_{p+2}^{i+1} \cdots v_{p+2}^\ell \lhd w_{s_i}$.

Now consider t_i. There are two cases that we treat separately.

Case 1. $t_i \in A$.
Suppose that the first letter of w_{s_i} is $b \neq t_i$. Note that if t_{i-1} is a set, then by construction of w_{s_i}, $b \notin t_{i-1}$. It follows that $v_1 \cdots v_{i-1} \cdot b \cdot v_i \cdots v_\ell \lhd w$, and that this piece is incompatible with T. Thus, the first letter of w_{s_i} is equal to t_i.

Case 2. $t_i \subseteq A$.
In this case, w_i is defined as the largest prefix of w_{s_i} such that $\mathsf{alph}(w_i) \subseteq t_i$. We have to prove that $v_p^i \lhd w_i$. Denote the first letter after w_i by c. Note that $c \notin t_i$.

Suppose that $w_i = \varepsilon$. Then c is the first letter after w_{i-1}, which means that if t_{i-1} is a set, then $c \notin t_{i-1}$. Then, the piece $v_1^1 \cdots v_{i-1}^1 \cdot c \cdot v_i^1 \cdots v_\ell^1$ is incompatible with T, and is a piece of w. Thus, $w_i \neq \varepsilon$. Denote the first letter of w_i by b. By construction, if t_{i-1} is a set, then $b \notin t_{i-1}$. Suppose that v_p^i is not a piece of w_i. Since we have $v_{p+1}^i \cdot v_{p+2}^{i+1} \cdots v_{p+2}^\ell \lhd w_{s_i}$ from the induction hypothesis, this means that one can read $v_1^i \cdot v_{p+2}^{i+1} \cdots v_{p+2}^\ell$ in w_{s_i}, after the c. We then have $v_1^1 \cdots v_{i-1}^1 \cdot b \cdot c \cdot v_i^1 \cdots v_\ell^1 \lhd w$, and this piece is incompatible with T.

We have thus shown that $w_1, \dots, w_\ell$ is a p-implementation of T. By construction, $w_1 \cdots w_\ell$ is a prefix of w. It is easy to see that they are equal. To this end, let $s \in A^*$ be such that $w = w_1 \cdots w_\ell \cdot s$. If $s \neq \varepsilon$, we have by construction of w_ℓ that the first letter b of s is not in t_ℓ, if it is a set. It follows that the piece $v_1 \cdots v_\ell \cdot b \lhd w$ is incompatible with T. □

The following proposition is a consequence of Lemma 4.18 and Lemma 4.19. It shows that for fixed ℓ and p, it is possible to describe all the unambiguous ℓ-templates, of which a given word is a p-implementation, in terms of the pieces of the word.

Proposition 4.20. *Let $\ell, p \in \mathbb{N}$. Let $K = |A|(p+2)\ell$ and let $w, w' \in A^*$ be such that $w \sim_K w'$. Let T be an unambiguous ℓ-template. If w is a $(p+2)$-implementation of T, then w' is a p-implementation of T.*

Proof. By Lemma 4.18, it follows that $v_{T,p+2} \lhd w$ and, for all pieces v such that $v \lhd w$, it holds that v is not incompatible with T. Note that $|v_{T,p+2}| \leq |A|(p+2)\ell = K$. As noted in Definition 4.17, the presence of pieces in a word that are incompatible with T is determined by the presence of such pieces that have length $\leq |v_{T,1}| + 2 \leq |A|\ell + 2 < K$. Since $w \sim_K w'$, it follows that $v_{T,p+2} \lhd w'$ and that there is no piece v that is incompatible with T such that $v \lhd w'$. Thus, by Lemma 4.19, w' is a p-implementation of T. □

We are now ready to show that whenever two pairs of states are PT-indistinguishable, there is a common pattern for these pairs of states in the automaton. As announced before, we actually prove a stronger statement: already if two pairs of states are $\mathsf{PT}[k]$-indistinguishable, for k large enough, this is witnessed by a common pattern for the pairs of states. This is made precise in the following proposition.

Proposition 4.21. *Let $(q_1, r_1), (q_2, r_2)$ be pairs of states of an NFA (A, Q, δ). Let $p = |Q|+1$ and $\kappa = (|A|(p+2))^{|A|+1}$. If $(q_1, r_1), (q_2, r_2)$ are $\mathsf{PT}[\kappa]$-indistinguishable, then (q_1, r_1) and (q_2, r_2) have a common pattern.*

Proof. Assume that $(q_1, r_1), (q_2, r_2)$ are $\mathsf{PT}[\kappa]$-indistinguishable. By definition, there exist words $w_1 \in \{w \mid q_1 \xrightarrow{w} r_1\}$ and $w_2 \in \{w \mid q_2 \xrightarrow{w} r_2\}$ such that $w_1 \sim_\kappa w_2$. We first show that there exists an ℓ-template of which both w_1 and w_2 are p-implementations. Define $N_{A,p+2} = (|A|(p+2))^{|A|}$. By Lemma 4.15, w_1 is a $(p+2)$-implementation of some ℓ-template T, with $\ell < N_{A,p+2}$. Since $\kappa = |A|(p+2)N_{A,p+2} > |A|(p+2)\ell$, Proposition 4.20 then yields that both w_1 and w_2 are p-implementations of this ℓ-template T.

We now use the fact that $p = |Q| + 1$ in order to find the appropriate patterns in the automaton. Define $\vec{B} = (B_1, \dots, B_n)$ as the subsequence of elements of T that are sets. Define $\vec{u} = (u_0, \dots, u_n)$, such that u_i is the concatenation of the letters between B_i and B_{i+1} in T. By definition, $(\vec{u}, \vec{B})$ is a factorization pair. As w_1 is a $(|Q|+1)$-implementation of T, the path from q_1 to r_1 used to read w_1 must traverse loops labeled by each of the B_i. Clearly, this is a $(\vec{u}, \vec{B})$-path. In the same way, the path from q_2 to r_2 used to read w_2 is a $(\vec{u}, \vec{B})$-path. Thus, (q_1, r_1) and (q_2, r_2) have a common pattern. □

Since PT-indistinguishable pairs are $\mathsf{PT}[\kappa]$-indistinguishable for all κ, thus in particular for $\kappa = (|A|(p+2))^{|A|+1}$, we immediately obtain the following corollary.

Corollary 4.22. *Let $\mathcal{A}$ be an NFA, and let (q_1, r_1) and (q_2, r_2) be two pairs of states in $\mathcal{A}$ that are* PT*-indistinguishable. Then, (q_1, r_1) and (q_2, r_2) have a common pattern.*

4.2.5 Intermezzo: an alternative method

Let us briefly discuss the approach that we took in [RZ13] to obtain the result of Corollary 4.22, which is different from the approach taken in the previous section. Given two PT-indistinguishable pairs of states (q_1, r_1) and (q_2, r_2), there are, by definition, for every $n \in \mathbb{N}$, words $v_n \in \{x \mid q_1 \xrightarrow{x} r_1\}$ and $w_n \in \{x \mid q_2 \xrightarrow{x} r_2\}$ such that $v_n \sim_n w_n$. This gives a sequence $(v_n, w_n)_n$. Above, we saw that we only need to consider this sequence up to $n = \kappa$.

In [RZ13], however, we do not bound this sequence, but we use combinatorial arguments to obtain a subsequence of words with a similar shape. An advantage of this approach over the approach of bounding the parameter is very recently observed and described in [CM14]: it gives a criterion that is satisfied by two input languages if and only if these languages are not PT-separable, and this criterion works for all input languages, i.e. even for *non-regular* input languages. We state this more general result in Theorem 4.26.

To make precise what we mean by a sequence of words with a similar shape, we introduce the following notion of adequateness.

Definition 4.23. Given an ℓ-template T, we say that a sequence $(w_n)_n$ is *T-adequate* if for all $n \geq 0$, w_n is an n-implementation of T. A sequence is called *adequate* if it is T-adequate for some T.

By Remark 4.14, it follows that for each adequate sequence, there is some unambiguous ℓ-template T, such that the sequence is T-adequate.

We use Simon's Factorization Forest theorem to show that every sequence of words admits an adequate subsequence. See [Sim90, Kuf08, Col10] for proofs and extensions of this theorem. Let us first recall this theorem. A *factorization tree* of a nonempty word x is a finite ordered unranked tree $F(x)$, whose nodes are labeled by nonempty words, in such a way that

- all leaves of $F(x)$ are labeled by letters,
- all internal nodes of $F(x)$ have at least 2 children,
- if a node labeled y has k children labeled $y_1, \ldots, y_k$ from left to right, then $y = y_1 \cdots y_k$.

Given a semigroup morphism $\varphi : A^+ \to S$ into a finite semigroup S, such a factorization tree is called *φ-Ramseyan* if every internal node has either 2 children, or k children labeled $y_1, \ldots, y_k$, in which case φ maps all words $y_1, \ldots, y_k$ to the same idempotent of S. Simon's Factorization Forest theorem states that every word has a φ-Ramseyan factorization tree of height at most $3|S|$.

Lemma 4.24. *Every sequence $(w_n)_n$ of words admits an adequate subsequence.*

Proof. Let $(w_n)_n$ be a sequence of words. We use Simon's Factorization Forest theorem with the morphism $\mathsf{alph} : A^+ \to \mathcal{P}(A)$. Consider a sequence $(F(w_n))_n$, where $F(w_n)$ is an alph-

Ramseyan tree of w_n, given by the Factorization Forest theorem. In particular, $F(w_n)$ has height at most $3 \cdot 2^{|A|}$. Therefore, extracting a subsequence if necessary, one may assume that the sequence of depths of the trees $F(w_n)$ is a constant H. We argue by induction on H. If $H = 0$, then every w_n is a letter. Hence, one may extract a constant subsequence from $(w_n)_n$, and this subsequence is adequate (just take the constant letter as ℓ-template).

If $H > 0$, we denote the arity of the root of $F(w_n)$ by $\mathsf{arity}(w_n)$, and we call it the arity of w_n. Two cases may arise.

Case 1. One can extract from $(w_n)_n$ a subsequence of bounded arity. Therefore, one may extract a subsequence of constant arity, say K, from w_n. This implies that each w_n has a factorization in K factors

$$w_n = w_{n,1} \cdots w_{n,K},$$

where $w_{n,i}$ is the label of the i-th child of the root in $F(w_n)$. Therefore, the alph-Ramseyan subtree of each $w_{n,i}$ is of height at most $H - 1$. By induction, one can extract from $(w_{n,i})_n$ an adequate subsequence. Proceeding iteratively for $i = 1, 2, \dots K$, one extracts from $(w_n)_n$ a subsequence $(w_{\sigma(n)})_n$ such that every $(w_{\sigma(n),i})_n$ is adequate. But a finite product of adequate sequences is obviously adequate. Therefore, the subsequence $(w_{\sigma(n)})_n$ of $(w_n)_n$ is also adequate.

Case 2. The arity of w_n grows to infinity. Therefore, extracting if necessary, one can assume that for every n, $\mathsf{arity}(w_n) \geq \max(|A| \cdot n, 3)$. Since all arities of words in the sequence are at least 3, all children of the root map to the same idempotent in $\mathcal{P}(A)$. But this says that each word from the subsequence is of the form

$$w_{\sigma(n)} = w_{n,1} \cdots w_{n,K_n},$$

with $K_n \geq |A| \cdot n$, and where, since the factorization tree is alph-Ramseyan, the alphabet of $w_{n,i}$ is the same for all i, say $B = \{b_1, \dots, b_m\}$. Since $|B| \leq |A|$, it follows that $(b_1 \cdots b_m)^n \lhd w_{\sigma(n)} \in B^*$, that is, $w_{\sigma(n)}$ is an n-implementation of the template B. Therefore, $(w_{\sigma(n)})_n$ is adequate. □

Next to the Factorization Forest theorem, we need another combinatorial result to relate the ℓ-templates of adequate sequences that are pairwise equivalent to each other. We state this result in the following lemma. The proof of this result is technical and is based on Lemma 8.2.5 and Theorem 8.2.6 from [Alm94]. We refer to [RZ13] for a proof of this lemma.

Lemma 4.25. *Let T be an unambiguous ℓ-template and let T' be an unambiguous ℓ'-template. Let $(v_n)_n$ and $(w_n)_n$ be two sequences of words such that*

- *$(v_n)_n$ is T-adequate*
- *$(w_n)_n$ is T'-adequate*
- *$v_n \sim_n w_n$ for every $n \geq 0$.*

Then, $T = T'$.

We now have all the ingredients to state a criterion for non-PT-separability of any two input languages.

Theorem 4.26 (follows from [RZ13, Lemma 6], [CM14, Lemma 2])**.** *Let L_1 and L_2 be languages. Then, L_1 and L_2 are not* PT*-separable if and only if there exists an ℓ-template T such that, for all n, there exist words $v_n \in L_1$ and $w_n \in L_2$, such that both $(v_n)_n$ and $(w_n)_n$ are T-adequate.*

Proof. The direction from right to left is clear: for every n, we have that both v_n and w_n are n-implementations of T, whence $v_n \sim_n w_n$, and it follows that L_1 and L_2 are not PT-separable.

For the other direction, suppose that L_1 and L_2 are not PT-separable. Then, for every $n \in \mathbb{N}$, there exist $v_n \in L_1$ and $w_n \in L_2$ such that

$$v_n \sim_n w_n. \tag{4.3}$$

This defines an infinite sequence of pairs $(v_n, w_n)_n$, from which we will iteratively extract infinite subsequences to obtain additional properties, while keeping (4.3). By Lemma 4.24, one can extract from $(v_n, w_n)_n$ a subsequence whose first component forms an adequate sequence. From this subsequence of pairs, using Lemma 4.24 again, we extract a subsequence whose second component is also adequate (note that the first component remains adequate). Therefore, one can assume that both $(v_n)_n$ and $(w_n)_n$ are themselves adequate.

Lemma 4.25 shows that one can choose the *same* ℓ-template T such that both $(v_n)_n$ and $(w_n)_n$ are T-adequate. □

In [CM14], it is shown that this criterion is decidable for a wide range of classes of input languages, among which the context-free languages.

Let us show that with the approach of Theorem 4.26, we have also obtained in a very different way (without having used any bound on the parameter) Corollary 4.22 again. Indeed, let $(q_1, r_1), (q_2, r_2)$ be PT-indistinguishable pairs of states of an NFA (A, Q, δ). By Lemma 4.6, these pairs determine languages that are not PT-separable. Therefore, we obtain from Theorem 4.26 that there is an ℓ-template T such that for every n, there are $v_n \in \{x \mid q_1 \xrightarrow{x} r_1\}$ and $w_n \in \{x \mid q_2 \xrightarrow{x} r_2\}$ that are n-implementations of T. In particular, this is the case for $n = |Q| + 1$. Now, we combine this with the second half of the proof of Proposition 4.21, which says that every two pairs of states, between which words can be read that are $(|Q| + 1)$-implementations of the same ℓ-template T, have a common pattern. This yields Corollary 4.22 again.

Corollary 4.22. *Let $\mathcal{A}$ be an NFA, and let (q_1, r_1) and (q_2, r_2) be two pairs of states in $\mathcal{A}$ that are* PT*-indistinguishable. Then, (q_1, r_1) and (q_2, r_2) have a common pattern.*

4.2.6 Separation theorem for piecewise testable languages

We collect the results of the previous sections in the following separation theorem for piecewise testable languages.

Theorem 4.27. *Let L_1 and L_2 be regular languages. Let $\mathcal{A} = (A, Q, \delta)$ be an NFA recognizing both L_1 and L_2, with $L_i = L(\mathcal{A}, I_i, F_i)$. Let $\kappa = (|A|(|Q| + 3))^{|A|+1}$. Then, the following conditions are equivalent.*

(1) L_1 and L_2 are PT*-separable,*

(2) L_1 and L_2 are $\mathsf{PT}[\kappa]$*-separable,*

(3) The language $[L_1]_\kappa$ separates L_1 from L_2,

(4) $(I_1 \times F_1) \times (I_2 \times F_2) \cap \mathrm{I}[\mathcal{A}] = \varnothing$,

(5) $(I_1 \times F_1) \times (I_2 \times F_2) \cap \mathrm{I}_\kappa[\mathcal{A}] = \varnothing$,

(6) There is no pair in $(I_1 \times F_1) \times (I_2 \times F_2)$ with a common pattern.

Proof. The implications (3) $\Leftrightarrow$ (2) $\Rightarrow$ (1) are trivial, as well as the implication (5) $\Rightarrow$ (4). We proved implication (4) $\Rightarrow$ (6) in Proposition 4.10, and implication (6) $\Rightarrow$ (5) in Proposition 4.21. The equivalences (1) $\Leftrightarrow$ (4) and (2) $\Leftrightarrow$ (5) follow from Lemma 4.6. □

Since there are finitely many $\mathsf{PT}[\kappa]$-languages, the equivalence (1) $\Leftrightarrow$ (2) yields a brute-force algorithm to test PT-separability. This gives Corollary 4.28. In the following section, we will show that, using Condition (6), we obtain another algorithm to test PT-separability, which has a better complexity result.

Corollary 4.28. *It is decidable whether two given regular languages are* PT*-separable.*

4.3 Complexity of PT-separability

In this section, we show that the presence of a common pattern for two pairs of states is a property that can be tested in PTIME with respect to the size of the alphabet and the size of the automaton. As a consequence, it follows from Theorem 4.27 that PT-separability can be decided in PTIME with respect to the size of the alphabet and the size of the automaton. We first introduce some notation.

Given a state p, we denote by $\mathsf{scc}(p, \mathcal{A})$ the strongly connected component of p in $\mathcal{A}$ (that is, the set of states that are reachable from p and from which p can be reached), and the set of labels of all transitions occurring in this strongly connected component is denoted by $\mathsf{alph\_scc}(p, \mathcal{A})$. Recall that the restriction of $\mathcal{A}$ to a subalphabet $B \subseteq A$ is defined by $\mathcal{A}\restriction_B := (B, Q, \delta \cap (Q \times B \times Q))$.

Let us first show that the following problem is in PTIME.

Lemma 4.29. *Consider the following problem.*

Input: *An NFA $\mathcal{A}$ over alphabet A, and states $p_1, q_1, r_1, p_2, q_2, r_2$ of $\mathcal{A}$.*
Question: *Do there exist a nonempty $B \subseteq A$ and paths $p_i \xrightarrow{\subseteq B} q_i \xrightarrow{=B} q_i \xrightarrow{\subseteq B} r_i$ in $\mathcal{A}$ for both $i = 1, 2$?*

This can be solved in PTIME *with respect to the size of $\mathcal{A}$ and the size of the alphabet.*

Proof. We will compute a decreasing sequence $(C_i)_i$ of alphabets that are overapproximating the greatest alphabet B that can be chosen for labeling the loops around q_1 and q_2. Note that if there exists such an alphabet B, it should be contained in

$$C_1 := \mathsf{alph\_scc}(q_1, \mathcal{A}) \cap \mathsf{alph\_scc}(q_2, \mathcal{A}).$$

Using Tarjan's algorithm to compute strongly connected components in linear time [Tar72], one can compute C_1 in linear time as well. Then, we restrict the automaton to alphabet C_1, and we repeat the process to obtain the sequence $(C_i)_i$. That is, we define

$$C_{i+1} := \mathsf{alph\_scc}(q_1, \mathcal{A}\restriction_{C_i}) \cap \mathsf{alph\_scc}(q_2, \mathcal{A}\restriction_{C_i}).$$

After a finite number n of iterations, we obtain $C_n = C_{n+1}$. Note that $n \leq |A|$. If $C_n = \varnothing$, then there exists no nonempty B for which there is an $(= B)$-loop around both q_1 and q_2. If $C_n \neq \varnothing$, then C_n is the maximal nonempty alphabet B such that there are $(= B)$-loops around both q_1 and q_2. It then remains to determine whether there exist paths $p_1 \xrightarrow{\subseteq B} q_1 \xrightarrow{\subseteq B} r_1$ and $p_2 \xrightarrow{\subseteq B} q_2 \xrightarrow{\subseteq B} r_2$, which can be performed in linear time.

To sum up, since the number n of iterations such that $C_n = C_{n+1}$ is bounded by $|A|$, and since each computation is linear with respect to the size of $\mathcal{A}$, one can decide in Ptime with respect to both $|A|$ and this size whether such a pair of paths occurs. □

The following proposition considers the problem of testing whether two pairs of states have a common pattern.

Proposition 4.30. *Given an NFA $\mathcal{A} = (A, Q, \delta)$, and two pairs of states, one can determine in* Ptime *with respect to the size of $\mathcal{A}$ and the size of the alphabet, whether these pairs of states have a common pattern.*

Proof. We extend the automaton $\mathcal{A}$ in the following way, to obtain a new automaton $\tilde{\mathcal{A}}$. For each 6-tuple $\tau = (p_1, q_1, r_1, p_2, q_2, r_2) \in Q^6$, we test whether there exist nonempty $B \subseteq A$ and paths $p_i \xrightarrow{\subseteq B} q_i \xrightarrow{=B} q_i \xrightarrow{\subseteq B} r_i$ in $\mathcal{A}$ for $i = 1, 2$. If this is the case, we add a new letter a_τ to the alphabet, and we add the "summary" transitions $p_1 \xrightarrow{a_\tau} r_1$ and $p_2 \xrightarrow{a_\tau} r_2$. Lemma 4.29 shows that for each 6-tuple, this test can be performed in Ptime. Since there are $|Q|^6$ of these tuples, computing $\tilde{\mathcal{A}}$ can then be done in Ptime.

Let (s_1, t_1) and (s_2, t_2) be pairs of states. By construction, we have the following. There exists some pair $(\vec{u}, \vec{B})$ such that there is a $(\vec{u}, \vec{B})$-path between both s_1 and t_1 and between s_2 and t_2, if and only if $\{w \mid s_1 \xrightarrow{w} t_1 \text{ in } \tilde{\mathcal{A}}\} \cap \{w \mid s_2 \xrightarrow{w} t_2 \text{ in } \tilde{\mathcal{A}}\} \neq \varnothing$. Since $\tilde{\mathcal{A}}$ can be built in Ptime, this can be decided in polynomial time as well. □

Combining Theorem 4.27 and Proposition 4.30, we obtain the following theorem.

Theorem 4.31. *It is decidable whether two regular languages, recognized by an NFA, are* PT*-separable. This can be achieved in* Ptime *with respect to the size of the alphabet and the size of the NFA.*

Recall from Section 2.3.1 that deciding whether two languages, recognized by a DFA, are SI-separable is co-NP-complete. What is different for SI, compared to PT, is that a common pattern for SI just consists of two words that have the same alphabet, whereas for PT the paths should have a certain shape: the different parts should occur in the same order, and there should be loops for the alphabets of $\vec{B}$. Thus, the patterns for PT are more complicated, but determining whether they exist has a lower complexity.

Chapter 5

Unambiguous languages

5.1 Characterizations of unambiguous languages 68

5.1.1 Logical characterization 68

Ehrenfeucht-Fraïssé games for $\mathsf{FO}^2(<)$ 69

5.1.2 Algebraic characterization 70

5.2 Separation by unambiguous languages 70

5.2.1 Fixpoint algorithm to compute $\mathsf{FO}^2(<)$-indistinguishable pairs . . . 72

5.2.2 Correctness of the fixpoint algorithm 73

5.2.3 Completeness of the fixpoint algorithm 74

Decompositions for (B, p)-patterns 75

Completeness result using (B, p)-decompositions 77

5.2.4 Proof of the separation theorem for unambiguous languages 81

5.3 Complexity of separation by unambiguous languages 82

In this chapter, we present our results on the separation problem for the class of unambiguous languages. In particular, we show that this is a decidable problem. As we have seen in Section 2.2.1, this result yields that the 2-pointlike sets for the variety corresponding to the class of unambiguous languages are computable. To our knowledge, this is a result shown for the first time in the paper [PvRZ13b], on which this chapter is based.

In Section 5.1, we first describe the class of languages that we study in this chapter. Section 5.2 forms the main part of this chapter. In this section, we introduce a fixpoint algorithm and we show how the information computed by this algorithm solves the separation problem for this class of languages. We also prove in this section that this algorithm is indeed correct and complete. As a byproduct, we obtain a description of a separator, if it exists. Finally, in Section 5.3, we briefly discuss the complexity of the separation problem for the class of unambiguous languages.

5.1 Characterizations of unambiguous languages

The class of languages that we study in this chapter is the class of unambiguous languages, i.e. the class of languages defined using unambiguous products. A product

$$L = B_0^* a_1 B_1^* \cdots B_{k-1}^* a_k B_k^*$$

is called *unambiguous* if every word of L admits exactly one factorization witnessing its membership in L. The number k is called the *size* of the product. An *unambiguous language* is a finite union of disjoint unambiguous products. The class of unambiguous languages is related to the class of piecewise testable languages studied in Chapter 4. Indeed, piecewise testable languages are boolean combinations of languages of the form $A^*b_1A^* \cdots A^*b_kA^*$. The latter languages are unambiguous, as is witnessed by the product

$$(A \setminus \{b_1\})^* b_1 (A \setminus \{b_2\})^* \cdots (A \setminus \{b_k\})^* b_k A^*.$$

Moreover, Schützenberger proved in [Sch76] that the class of unambiguous languages is closed under boolean operations. Therefore, piecewise testable languages form a subclass of the unambiguous languages. This inclusion is strict, as can be seen, for example, from the language A^*a. This is an unambiguous language (for any alphabet A), but for $A \supseteq \{a, b\}$, it cannot be defined in terms of pieces: for every n, there are words inside and outside of A^*a that have the same pieces up to size n, for example the words $(ab)^n$ and $(ab)^n a$.

Many characterizations for unambiguous languages have been found, both in terms of logical fragments and in terms of algebraic properties. We refer to [TT02, DGK08] for a detailed overview of these characterizations.

5.1.1 Logical characterization

From a logical point of view, unambiguous languages can be defined as a fragment of first-order logic both in terms of quantifier alternations and in terms of the number of variables of a formula. Also, characterizations have been found as fragments of temporal logic: the fragment using only the unary temporal operators 'next', 'previously', 'sometime in the future', and 'sometime in the past' [EVW97, EVW02], and the fragment of so-called ranker languages [WI09].

We will now describe the characterizations as fragments of first-order logic in more detail. A $\Sigma_2(<)$-formula is a first-order formula of the form

$$\exists x_1 \ldots \exists x_n \, \forall y_1 \ldots \forall y_m \; \varphi(x_1, \ldots, x_n, y_1, \ldots, y_m),$$

where φ is quantifier-free. A $\Pi_2(<)$-formula is a formula whose negation is a $\Sigma_2(<)$-formula. Finally, the class $\Delta_2(<)$ is the intersection of these two classes. Thus, a language is $\Delta_2(<)$-*definable* if it can be defined both by a $\Sigma_2(<)$-formula and the negation of a $\Sigma_2(<)$-formula. It was shown in [PW97] that a language is unambiguous if and only if it is $\Delta_2(<)$-definable.

Another logical characterization of the unambiguous languages was given in [TW98]. In this paper, it is shown that the fragment $\Delta_2(<)$ corresponds to the fragment of those first-order logic formulas that use only two (reusable) variable names. This fragment of first-order logic is denoted by $\mathsf{FO}^2(<)$.

Example 5.1. Consider the language A^*bA^*a. The unambiguous product $(A/b)^*bA^*a$ witnesses that this is an unambiguous language. It is defined by the following $\mathsf{FO}^2(<)$-formula,

$$\exists x\, \exists y.\ \Big(b(x) \wedge a(y) \wedge \forall x.\ \big(\neg(y < x)\big)\Big).$$

It is also defined by the following $\Sigma_2(<)$-formula,

$$\exists x\, \exists y\, \forall z.\ \Big(b(x) \wedge a(y) \wedge \neg(y < z)\Big),$$

and the following $\Pi_2(<)$-formula,

$$\forall x\, \exists y\, \exists z.\ \Big((a(x) \vee x < y) \wedge b(z)\Big),$$

which shows that it is $\Delta_2(<)$-definable.

It turns out that the characterization of unambiguous languages as $\mathsf{FO}^2(<)$-definable languages provides a suitable framework for our proofs. In particular, this formulation allows us to use Ehrenfeucht-Fraïssé games for $\mathsf{FO}^2(<)$ in our proofs. We will define these games now.

Ehrenfeucht-Fraïssé games for $\mathsf{FO}^2(<)$

Ehrenfeucht-Fraïssé games form a widely applicable method from model theory to show that two structures are equivalent, or not, with respect to a certain logic.

The game is played by two players, called Spoiler and Duplicator. Given a logical fragment, the intuition is that Spoiler wants to show that the words can be distinguished by this fragment, and Duplicator wants to show that this is not the case. That is, Duplicator wants to show that they look the same with respect to the logical fragment. Here, we only introduce a version of the games instantiated for $\mathsf{FO}^2(<)$.

Definition 5.2. An *Ehrenfeucht-Fraïssé game* for $\mathsf{FO}^2(<)$ (or short, *EF game*) is a game played by two players, called Spoiler and Duplicator. The game is played on two words. In front of each of the words, initially one pebble is placed. The game lasts for at most k rounds, where k is fixed beforehand. Each of the rounds is played according to the following rules:

- Spoiler chooses one of the words, picks up the pebble placed on or in front of this word and moves it, to the left or to the right, to a position in the same word.
- Duplicator answers by moving the other pebble in the other word, mimicking both the direction and the letter chosen by Spoiler.

If Duplicator is unable to answer, she loses. She wins if she is able to answer in each of the k rounds.

As before, we define a sequence of congruence relations on words in A^*, that indicates how similar the words are from the perspective of the logic $\mathsf{FO}^2(<)$. For $u, v \in A^*$, we have

$$u \sim_k v \ \Leftrightarrow\ u \text{ and } v \text{ satisfy the same } \mathsf{FO}^2(<)\text{-formulas up to quantifier depth } k.$$

The EF games are particularly useful as they provide a tool to show that two words are $\sim_k$-equivalent, or not, as follows from the following well-known theorem.

Theorem 5.3 ([Imm82, Imm99])**.** *Duplicator has a winning strategy in the k-round EF game for $\mathsf{FO}^2(<)$ on the words w, w' if and only if $w \sim_k w'$.*

Example 5.4. Consider the words aba and $abba$. Clearly, whichever letter Spoiler choses to place a pebble on in the first round, Duplicator is able to answer by placing a pebble on the same letter in the other word. It follows that $aba \sim_1 abba$. However, if Spoiler places a pebble on the first b in the word $abba$ in the first round, Duplicator has to place a pebble on the b in aba. Spoiler can then move the pebble from $abba$ to the right and again place it on a b, but Duplicator cannot mimic this move in aba. Thus, $aba \nsim_2 abba$.

5.1.2 Algebraic characterization

The variety of finite monoids for which every regular $\mathcal{D}$-class (that is, every $\mathcal{D}$-class that contains an idempotent) is an aperiodic semigroup, is called DA. A monoid M is a member of DA if and only if it satisfies the identity $(xy)^\omega y (xy)^\omega = (xy)^\omega$. The variety DA forms a proper subclass of the variety A of aperiodic monoids.

In [Sch76], it is shown that unambiguous languages are precisely the DA-recognizable languages.

5.2 Separation by unambiguous languages

To show that the separation problem for $\mathsf{FO}^2(<)$-definable languages is decidable, we follow the approach discussed in Section 2.2.3 and work with $\mathsf{FO}^2(<)$-indistinguishable pairs of monoid elements. We will provide a fixpoint algorithm that, given a monoid M and a surjective morphism $\alpha : A^* \to M$, computes the $\mathsf{FO}^2(<)$-indistinguishable pairs of M, that is, the 2-pointlike sets of M for the variety DA. These pairs of course do not depend on the morphism α, but the algorithm needs such a fixed morphism to perform its computations. We therefore denote these pairs by $\mathrm{I}^{\mathsf{FO}^2(<)}(\alpha)$. Since this chapter only deals with the logic $\mathsf{FO}^2(<)$, we also write $\mathrm{I}(\alpha)$ for this set of pairs.

We will use the fact that the logic can be stratified according to the quantifier depth of a formula. The fragment of $\mathsf{FO}^2(<)$ that consists of formulas of quantifier depth up to k is denoted by $\mathsf{FO}^2(<)[k]$. For every $k \in \mathbb{N}$, we define the following congruence relation, which is of finite index. For words $u, v \in A^*$,

$$u \sim_k v \;\Leftrightarrow\; u \text{ and } v \text{ satisfy the same } \mathsf{FO}^2(<)\text{-formulas up to quantifier depth } k.$$

The set of pairs of monoid elements that are $\mathsf{FO}^2(<)[k]-$indistinguishable is denoted by $\mathrm{I}_k^{\mathsf{FO}^2(<)}(\alpha)$, or simply $\mathrm{I}_k(\alpha)$. That is,

$$\mathrm{I}_k(\alpha) = \{(s, s') \in M^2 \mid \exists w \in \alpha^{-1}(s).\ \exists w' \in \alpha^{-1}(s').\ w \sim_k w'\}.$$

For reasons that will become apparent in Sections 5.2.2 and 5.2.3, we need to keep track of the alphabet of words that map onto the monoid elements. Thus, for $\alpha : A^* \to M$ and $s, t \in M$, we define

$$(s, s') \in \mathrm{I}_k(\alpha, B) \quad \Leftrightarrow \quad \begin{array}{l}\text{there are } w \in \alpha^{-1}(s), w' \in \alpha^{-1}(s') \text{ such that} \\ \mathsf{alph}(w) = \mathsf{alph}(w') = B \text{ and } w \sim_k w'.\end{array}$$

Since a formula of rank 1 is already able to distinguish between words that have a different alphabet, it follows that $\mathrm{I}_k(\alpha) = \bigcup_{B \subseteq A} \mathrm{I}_k(\alpha, B)$. And, as always, $\mathrm{I}(\alpha) = \bigcap_{k \in \mathbb{N}} \mathrm{I}_k(\alpha)$. The following inclusions hold by definition,

$$\mathrm{I}(\alpha) = \bigcap_{n \in \mathbb{N}} \mathrm{I}_n(\alpha) \subseteq \ldots \subseteq \mathrm{I}_{k+1}(\alpha) \subseteq \mathrm{I}_k(\alpha) \subseteq \ldots \subseteq \mathrm{I}_1(\alpha).$$

The difficulty in computing $\mathrm{I}(\alpha)$ lies in knowing the limit behavior of the sequence $(\mathrm{I}_k(\alpha))_{k \in \mathbb{N}}$. Note that all elements of this sequence are in the finite set $\mathcal{P}(M \times M)$, and since the sequence is decreasing with respect to the inclusion order, there must be an index for which the sequence stabilizes. It turns out that we can compute such a stabilization index for this sequence, which depends on the size of the monoid M and the size of the alphabet A.

Let us briefly sketch the content of the rest of this section. In Section 5.2.1, we will provide a fixpoint algorithm of which we claim that it computes the $\mathsf{FO}^2(<)$-indistinguishable pairs. Given a surjective morphism $\alpha : A^* \to M$, the output of this algorithm is denoted by $\mathtt{Alg}(\alpha)$. In Section 5.2.2, we will prove that the presented algorithm is indeed correct: all of the pairs that the algorithm outputs are $\mathsf{FO}^2(<)$-indistinguishable pairs, i.e. $\mathtt{Alg}(\alpha) \subseteq \mathrm{I}(\alpha)$. Finally, we prove in Section 5.2.3 that for $\kappa = (2|M|^2 + 3)|A|^2$, the $\mathsf{FO}^2(<)[\kappa]-$indistinguishable pairs occur as output of the algorithm. Completeness of the algorithm follows, since by definition, $\mathrm{I}(\alpha) \subseteq \mathrm{I}_\kappa(\alpha)$. To summarize, we then have the following inclusions.

$$\mathrm{I}(\alpha) \subseteq \mathrm{I}_\kappa(\alpha) \overset{5.2.3}{\subseteq} \mathtt{Alg}(\alpha) \overset{5.2.2}{\subseteq} \mathrm{I}(\alpha). \tag{5.1}$$

The value $\kappa = (2|M|^2 + 3)|A|^2$ thus serves as a stabilization index for the sequence mentioned above. In Section 5.2.4, we will prove the following theorem that contains the main results of this chapter.

Theorem 5.5. *Let M be a monoid, let P and Q be subsets of M, and let $\alpha : A^* \to M$ be a surjective morphism. Let $\kappa = (2|M|^2 + 3)|A|^2$. Then, the following conditions are equivalent.*

(1) The languages $\alpha^{-1}(P)$ and $\alpha^{-1}(Q)$ are $\mathsf{FO}^2(<)$-separable,

(2) The languages $\alpha^{-1}(P)$ and $\alpha^{-1}(Q)$ are $\mathsf{FO}^2(<)[\kappa]$-separable,

(3) The language $[\alpha^{-1}(P)]_{\sim\kappa}$ separates $\alpha^{-1}(P)$ from $\alpha^{-1}(Q)$,

(4) $P \times Q \,\cap\, \mathrm{I}(\alpha) = \varnothing$.

While a proof of this theorem is provided in Section 5.2.4, let us for now just mention that the implications (3) ⇒ (2) ⇒ (1) are trivial, and that the implication (1) ⇒ (4) is immediate from the definition of $\mathsf{FO}^2(<)$-indistinguishable pairs. The difficult direction of the theorem is (4) ⇒ (3). Note that this direction would follow from the inclusion $\mathrm{I}_\kappa(\alpha) \subseteq \mathrm{I}(\alpha)$, announced in (5.1). Indeed, if $P \times Q \,\cap\, \mathrm{I}(\alpha) = \varnothing$, the inclusion gives that also $P \times Q \,\cap\, \mathrm{I}_\kappa(\alpha) = \varnothing$. This means that there are no words $w \in \alpha^{-1}(P), w' \in \alpha^{-1}(Q)$ such that $w \sim_\kappa w'$. Thus,

$[\alpha^{-1}(P)]_{\sim\kappa} \cap \alpha^{-1}(Q) = \varnothing$. To prove Theorem 5.5, it thus suffices to show that $\mathrm{I}_\kappa(\alpha) \subseteq \mathrm{I}(\alpha)$. This inclusion will follow from the results in Sections 5.2.2 and 5.2.3.

When starting from two regular languages L_1 and L_2, recall that one can always construct a monoid that recognizes both languages. It follows that the languages are $\mathsf{FO}^2(<)$-separable if and only if they are $\mathsf{FO}^2(<)[\kappa]-$separable, where κ is calculated as in the theorem. In case the languages are separable, a description of a separator would be $[L_1]_{\sim\kappa}$, the saturation of the first language by the $\sim_\kappa$-equivalence. To test $\mathsf{FO}^2(<)$-separability, one could thus use a brute-force approach and test all of the finitely many $\mathsf{FO}^2(<)[\kappa]-$definable languages. This yields Corollary 5.6. However, it turns out that exploiting Condition (4) yields a better complexity result. Besides providing a means to prove the inclusion $\mathrm{I}_\kappa(\alpha) \subseteq \mathrm{I}(\alpha)$, this exploitation of Condition (4) is another purpose of the fixpoint algorithm that we present in Section 5.2.1.

Corollary 5.6. *It is decidable whether two regular languages can be separated by an* $\mathsf{FO}^2(<)$*-definable language.*

Remark. In Theorem 5.5, we provide a bound κ on the quantifier rank of $\mathsf{FO}^2(<)$-formulas that need to be considered to define a potential separator. It turns out that the same value κ also works to bound unambiguous products: there exists an $\mathsf{FO}^2(<)$-separator if and only if there exists a separator defined by a boolean combination of unambiguous products of size κ. This approach of bounding the size of unambiguous products rather than the quantifier rank of $\mathsf{FO}^2(<)$-formulas was taken in the paper [PvRZ13b].

5.2.1 Fixpoint algorithm to compute $\mathsf{FO}^2(<)$-indistinguishable pairs

For a monoid M, and a surjective morphism $\alpha : A^* \to M$, we define the following fixpoint algorithm that computes a set $\mathtt{Alg}(\alpha) \subseteq M \times M$.

Initialize by adding, for all $a \in A$, the pair $(\alpha(a), \alpha(a))$ to $\mathtt{Alg}(\alpha, \{a\})$. Then, saturate the set $\mathtt{Alg}(\alpha) = \bigcup_{B \subseteq A} \mathtt{Alg}(\alpha, B)$ with the following two operations.

(1) If $(s, s') \in \mathtt{Alg}(\alpha, B)$ and $(t, t') \in \mathtt{Alg}(\alpha, C)$, then add $(st, s't')$ to $\mathtt{Alg}(\alpha, B \cup C)$.

(2) If both (s, s') and (t, t') belong to $\mathtt{Alg}(\alpha, B)$, and, furthermore, there exist $w, w' \in B^*$ such that $\alpha(w) = r, \alpha(w') = r'$, then add $(s^\omega r t^\omega, s'^\omega r' t'^\omega)$ to $\mathtt{Alg}(\alpha, B)$.

Note that there are finitely many subsets B of A, and that during the execution of the algorithm, for every such B, the set $\mathtt{Alg}(\alpha, B)$ only gets larger with respect to inclusion. As the sets are all bounded from above by $M \times M$, the algorithm will terminate.

Remark. Operation (1) takes care of the fact that the $\sim_k$-relations are congruences. Such an operation will be needed in any fixpoint algorithm calculating the indistinguishable pairs of a monoid with respect to a variety of languages. Operation (2), however, is specific for the class of unambiguous languages.

5.2.2 Correctness of the fixpoint algorithm

In this section, we show that the fixpoint algorithm only outputs pairs of elements that are indeed $\mathsf{FO}^2(<)$-indistinguishable. It will follow from the fact that the $\sim_k$-relations are congruences that Operation (1), applied to two $\mathsf{FO}^2(<)$-indistinguishable pairs, yields an $\mathsf{FO}^2(<)$-indistinguishable pair. To show that the pairs generated by Operation (2) from $\mathsf{FO}^2(<)$-indistinguishable pairs are again $\mathsf{FO}^2(<)$-indistinguishable, we will use Lemma 5.8. Let us first state a related simpler result.

Lemma 5.7. *Let $u, v, w \in B^*$ and let $k \in \mathbb{N}$. If $\mathsf{alph}(u) = \mathsf{alph}(v) = B$, then $u^k w v^k \sim_k u^k v^k$.*

Proof. We prove this using the formalism of EF games for $\mathsf{FO}^2(<)$. The following strategy for Duplicator is a winning strategy: if Spoiler plays inside w, Duplicator plays a position in the last copy of u or the first copy of v, labeled by the same letter as the one that Spoiler chose. In this way, she will still have at least $k-1$ occurrences of each letter of B both to her left and to her right. It follows that she is able to survive all k rounds of the game. And, if Spoiler does not play inside w, clearly, Duplicator can mimic his moves in the other word. It follows from Theorem 5.3 that $u^k w v^k \sim_k u^k v^k$. □

We now slightly generalize Lemma 5.7.

Lemma 5.8. *Let $u_1, u_2, v_1, v_2, w_1, w_2$ be words in B^* and let $k \in \mathbb{N}$. If $u_1 \sim_k u_2$, $v_1 \sim_k v_2$, and $\mathsf{alph}(u_1) = \mathsf{alph}(u_2) = \mathsf{alph}(v_1) = \mathsf{alph}(v_2) = B$, then, for all $p \geq k$, $u_1^p w_1 v_1^p \sim_k u_2^p w_2 v_2^p$.*

Proof. Lemma 5.7 implies that $u_1^k w_1 v_1^k \sim_k u_1^k v_1^k \sim_k u_1^k w_2 v_1^k$. Since $\sim_k$ is a congruence and since $u_1 \sim_k u_2$, $v_1 \sim_k v_2$, it follows that $u_1^k w_1 v_1^k \sim_k u_2^k w_2 v_2^k$. And, for the same reasons, we then have $u_1^p w_1 v_1^p \sim_k u_2^p w_2 v_2^p$. □

The correctness of the fixpoint algorithm is proved in the next proposition. In the proof, we make use of the fact that the fixpoint algorithm keeps track of information about the alphabet. This information allows us to perform an induction on the number of applications of the operations.

Proposition 5.9. *The fixpoint algorithm described in Section 5.2.1 is correct.*

Proof. One has to show that $\mathtt{Alg}(\alpha) \subseteq \mathrm{I}(\alpha)$. That is, that for all $(m, n) \in \mathtt{Alg}(\alpha)$, it holds that for all $k \in \mathbb{N}$, there exist $w_1 \in \alpha^{-1}(m), w_2 \in \alpha^{-1}(n)$ such that $w_1 \sim_k w_2$. Note that $\mathtt{Alg}(\alpha) = \bigcup_{B \subseteq A} \mathtt{Alg}(\alpha, B)$, and thus, for all $(m, n) \in \mathtt{Alg}(\alpha)$, there exists $B \subseteq A$ such that $(m, n) \in \mathtt{Alg}(\alpha, B)$.

Therefore, the following claim is stronger than the proposition, as it also states something about the alphabet of the words in the preimages. We will prove this stronger statement.

Claim. For every $B \subseteq A$, for all $(m, n) \in \mathtt{Alg}(\alpha, B)$, and for all $k \in \mathbb{N}$, there exist $w_1 \in \alpha^{-1}(m), w_2 \in \alpha^{-1}(n)$ such that $w_1 \sim_k w_2$ and $\mathsf{alph}(w_1) = \mathsf{alph}(w_2) = B$.

We prove this claim by induction on the number of applications of the operations used to obtain that $(m, n) \in \mathtt{Alg}(\alpha)$. If no application of the operations is used to obtain that

$(m,n) \in \mathtt{Alg}(\alpha)$, then (m,n) was added during the initialization phase. This means there is $a \in A$ such that $(m,n) \in \mathtt{Alg}(\alpha, \{a\})$, and $m = \alpha(a) = n$. Clearly, for all $k \in \mathbb{N}$, $a \sim_k a$ and $\mathsf{alph}(a) = \{a\}$.

If operations were applied, suppose first that the last operation applied to obtain $(m,n) \in \mathtt{Alg}(\alpha)$ was Operation (1), and that $(m,n) \in \mathtt{Alg}(\alpha, B)$. Then, there are $s,t,s',t' \in M$ such that $m = st, n = s't'$ and, $(s,s') \in \mathtt{Alg}(\alpha, C), (t,t') \in \mathtt{Alg}(\alpha, C')$, for some C, C', such that $C \cup C' = B$. By the induction hypothesis, for all $k \in \mathbb{N}$, there are

$$\begin{array}{ll} u \in \alpha^{-1}(s), & u' \in \alpha^{-1}(s'), \\ v \in \alpha^{-1}(t), & v' \in \alpha^{-1}(t'), \end{array}$$

such that $\mathsf{alph}(u) = \mathsf{alph}(u') = C$, $\mathsf{alph}(v) = \mathsf{alph}(v') = C'$, $\mathsf{alph}(u) \cup \mathsf{alph}(v) = B$, and $u \sim_k u'$, $v \sim_k v'$. Then $uv \in \alpha^{-1}(st) = \alpha^{-1}(m)$, and $u'v' \in \alpha^{-1}(s't') = \alpha^{-1}(n)$. Since $\sim_k$ is a congruence, $uv \sim_k u'v'$. Also, $\mathsf{alph}(uv) = \mathsf{alph}(u'v') = B$. This shows the claim for this case.

Finally, suppose the last operation applied to obtain $(m,n) \in \mathtt{Alg}(\alpha)$ was Operation (2), and that $(m,n) \in \mathtt{Alg}(\alpha, B)$. Then, there are $w, w' \in B^*$ with $\alpha(w) = r, \alpha(w') = r'$, and there are $s,t,s',t' \in M$ such that $m = s^\omega r t^\omega, n = s'^\omega r' t'^\omega$ and, $(s,s'), (t,t') \in \mathtt{Alg}(\alpha, B)$. By the induction hypothesis, for all $k \in \mathbb{N}$, there are

$$\begin{array}{ll} u \in \alpha^{-1}(s), & u' \in \alpha^{-1}(s'), \\ v \in \alpha^{-1}(t), & v' \in \alpha^{-1}(t'), \end{array}$$

such that $\mathsf{alph}(u) = \mathsf{alph}(u') = \mathsf{alph}(v) = \mathsf{alph}(v') = B$, and $u \sim_k u'$, $v \sim_k v'$. Fix $k \in \mathbb{N}$ and let $p = k \cdot |M|!$. For every $x \in M$, we then have $x^p = x^\omega$. It follows that $\alpha(u^p w v^p) = s^\omega r t^\omega$, and $\alpha(u'^p w' v'^p) = s'^\omega r' t'^\omega$. Both words have alphabet B, and by Lemma 5.8, $u^p w v^p \sim_k u'^p w' v'^p$. This concludes the proof of the claim and of the proposition. □

5.2.3 Completeness of the fixpoint algorithm

This section is devoted to proving the completeness of the fixpoint algorithm presented in Section 5.2.1. In order to do this, we prove a stronger statement. Namely, we fix a specific $\kappa \in \mathbb{N}$ and prove that for this value of κ, all $\mathsf{FO}^2(<)[\kappa]$-indistinguishable pairs occur as output of the fixpoint algorithm. In this section, we first work with a fixed subalphabet B. In Proposition 5.16, we show how the desired general statement easily follows from the other results.

Roughly, our approach to show that the $\mathsf{FO}^2(<)[\kappa]$-indistinguishable pairs occur as output of the fixpoint algorithm, is the following. We first define a notion of pattern on words. Then, we show that if the images of two words form an $\mathsf{FO}^2(<)[\kappa]$-indistinguishable pair, they will either both contain a big pattern, or both contain only a small pattern. We decompose the words along these patterns, in such a way that the factors use only a strict subalphabet, and are pairwise equivalent up to some quantifier rank. We then use induction on the size of the alphabet. If the words only contained a small pattern, Operation (1) will suffice to recompose the factors and find that the original pair was in $\mathtt{Alg}(\alpha)$. If the words contained a big pattern, the proof is more complicated, as we will have to mold the factors in the right shape to be able to apply Operation (2). An ingredient used to obtain this is the pigeonhole principle, which entails that a sufficiently long product of monoid elements contains a factor of consecutive elements that can be repeated without changing the value of the product.

Decompositions for (B, p)-patterns

The notion of (B, p)-pattern, that we introduced in Chapter 4, also plays a central role in the proofs of this section. Let us first recall this notion.

Definition 5.10. Let $B = \{b_1, \ldots, b_n\}$ be a finite alphabet, ordered as $b_1 < \ldots < b_n$, and let $p \in \mathbb{N}$. A word $w \in B^*$ is a (B, p)*-pattern* if $w \in (B^* b_1 B^* \cdots b_n B^*)^p$. Or, in other words, if this w contains the subword $(b_1 \cdots b_n)^p$.

Using the concept of (B, p)-patterns, we will introduce a notion of decomposition for words that fits well with our purposes, due to their following useful properties. First of all, $\mathsf{FO}^2(<)$-formulas of sufficiently large quantifier rank can express the property of allowing a certain decomposition. We are thus able to detect the decompositions that words allow. Also, if two words both can be decomposed in a certain way, the factors of the decompositions will be pairwise related. Furthermore, the decompositions break the words up into factors that use a strictly smaller alphabet. These properties permit us to prove the completeness of the fixpoint algorithm using induction on the size of the alphabet. We will make this more precise in the remainder of this section.

The alphabet $B = \{b_1, \ldots, b_n\}$, ordered as $b_1 < \ldots < b_n$, is fixed for the rest of this section. Let w be a (B, p)-pattern, that is, $w \in (B^* b_1 B^* \cdots b_n B^*)^p$. Then, w can be decomposed into a product of factors witnessing its membership in $(B^* b_1 B^* \cdots b_n B^*)^p$. For example, as

$$w = \Big(\prod_{i=1}^{pn} w_i \cdot b_{i \bmod n} \Big) \cdot w_{pn+1},$$

where, for ease of notation, if n divides i, we write $b_{i \bmod n}$ to denote the letter b_n. We apply this notation throughout this section.

For a given word w, we are interested in the left-most decomposition of w along the marked product $(B^* b_1 B^* \cdots b_n B^*)^p \, b_1 B^* \cdots b_{\ell \bmod n} B^*$, where p and ℓ are as big as possible.

Definition 5.11. If $w \in B^*$, then there is a unique number $\ell \in \mathbb{N}$, such that

(1) $w = \Big(\prod_{i=1}^{\ell} w_i \cdot b_{i \bmod n} \Big) \cdot w_{\ell+1}$, and

(2) for all $i \in \{1, \ldots, \ell + 1\}$, $b_{i \bmod n} \notin \mathsf{alph}(w_i)$.

Let $p \in \mathbb{N}$ be such that $pn \leq \ell < (p+1)n$. Then, the decomposition of Condition (1) is called *the* (B, p)*-decomposition* of w. In this case, we also say that w *admits* a (B, p)-decomposition. The number ℓ is called the *length* of the decomposition.

Note that there is just one value of p for which a word $w \in B^*$ admits a (B, p)-decomposition. This is precisely the value of p for which w is a (B, p)-pattern, but not a $(B, p + 1)$-pattern.

Let us illustrate these notions by giving two examples.

Example 5.12. The word $w = bcacbbcccaccbaa$ over $B = \{a, b, c\}$, ordered as $a < b < c$, admits a $(B, 1)$-decomposition, since it is a $(B, 1)$-pattern, but not a $(B, 2)$-pattern. The length of its $(B,1)$-decomposition is 5:

$$\underbrace{bc}_{w_1}\ \mathbf{a}\ \underbrace{c}_{w_2}\ \mathbf{b}\ \underbrace{b}_{w_3}\ \mathbf{c}\ \underbrace{cc}_{w_4}\ \mathbf{a}\ \underbrace{cc}_{w_5}\ \mathbf{b}\ \underbrace{aa}_{w_6}$$

The word $w = aacbacbaabcaaa$ over the same ordered alphabet admits a $(B,2)$-decomposition. The length of this decomposition is 7:

$$\underbrace{\varepsilon}_{w_1}\ \mathbf{a}\ \underbrace{ac}_{w_2}\ \mathbf{b}\ \underbrace{a}_{w_3}\ \mathbf{c}\ \underbrace{b}_{w_4}\ \mathbf{a}\ \underbrace{a}_{w_5}\ \mathbf{b}\ \underbrace{\varepsilon}_{w_6}\ \mathbf{c}\ \underbrace{\varepsilon}_{w_7}\ \mathbf{a}\ \underbrace{aa}_{w_8}$$

As announced before, a useful result about (B,p)-decompositions is that they can be detected using an $\mathsf{FO}^2(<)$-formula of sufficiently big quantifier rank. We prove this result in the following lemma. Recall that B is a fixed ordered alphabet of size n.

Lemma 5.13. *Let $p,k \in \mathbb{N}$ be such that $(p+1)|B| \leq k$. Let u,v be words such that $u \sim_k v$. Then, u admits a (B,p)-decomposition if and only if v admits a (B,p)-decomposition. Moreover, the associated (B,p)-decompositions (in the sense of Definition 5.11) have the same length.*

Proof. Let us first prove that whenever u admits a (B,p)-decomposition, v does too. The converse of this statement then follows by symmetry. We prove this using EF games for $\mathsf{FO}^2(<)$. Since $u \sim_k v$, it follows from Theorem 5.3 that Duplicator has a winning strategy in the k-round EF game on u and v.

Assume that u admits a (B,p)-decomposition, and that the decomposition is

$$u = \left(\prod_{i=1}^{\ell} u_i \cdot c_i\right) \cdot u_{\ell+1},$$

where, for all i, c_i denotes the first occurrence of the letter $b_{i \bmod n}$ after u_i. The number of c_i's in this decomposition is $\ell < (p+1)|B| \leq k$. Let Spoiler subsequently play on $c_1, c_2, \ldots, c_\ell$ in u. Since $\ell < (p+1)|B| \leq k$, Duplicator can answer in v to each of Spoiler's moves in u, thus identifying $c_1 \cdots c_\ell$ as a scattered subword in v. It follows that v is a (B,p)-pattern. If v were also to contain the scattered subword $c_1 \cdots c_\ell c_{\ell+1}$, then Spoiler could next move the pebble in v to this $c_{\ell+1}$, but Duplicator would then not be able to answer this move in u, since the length of the (B,p)-decomposition of u is just ℓ. Since $\ell + 1 \leq k$, this would contradict the fact that $u \sim_k v$. It follows that v is not a $(B,p+1)$-pattern, and that the length of its (B,p)-decomposition is ℓ. □

Not only can (B,p)-decompositions be detected using an $\mathsf{FO}^2(<)$-formula of sufficiently big quantifier rank, but one can also obtain information about the $\mathsf{FO}^2(<)$-formulas that are satisfied by the factors in the (B,p)-decomposition. This is made precise in the next lemma.

Lemma 5.14. *Let $p,k,k' \in \mathbb{N}$ be such that $(p+1)|B| + k' \leq k$. Let u,v be words such that $u \sim_k v$, and such that both admit a (B,p)-decomposition that has length ℓ:*

$$\begin{aligned} u &= \left(\textstyle\prod_{i=1}^{\ell} u_i \cdot b_{i \bmod n}\right) \cdot u_{\ell+1}, \\ v &= \left(\textstyle\prod_{i=1}^{\ell} v_i \cdot b_{i \bmod n}\right) \cdot v_{\ell+1}. \end{aligned}$$

Then, for all $i \in \{1, \dots, \ell + 1\}$, $u_i \sim_{k'} v_i$.

Proof. As before, we denote the marked occurrences of $b_{i \bmod n}$ by c_i, for all i. Recall that by definition of the decompositions, for each j, $c_j \notin \mathsf{alph}(u_j)$ and $c_j \notin \mathsf{alph}(v_j)$. Let i be in $\{1, \dots, \ell + 1\}$. We will use EF games to show that $u_i \sim_{k'} v_i$. Let Spoiler play inside u_i and v_i during k' rounds in the EF game on u and v. As $u \sim_k v$, we know that Duplicator is able to answer to these moves. In order to conclude that indeed $u_i \sim_{k'} v_i$, we need to show that she is able to answer by playing within u_i and v_i. In fact, we will show that if she would play outside of this area, she would lose. Since she has a winning strategy, it then follows that she is able to play within u_i and v_i.

Suppose that at a certain point Duplicator plays on a position outside of u_i and v_i, to the left, say on v. Then Spoiler can move the pebble in u subsequently to $c_{i-1}, c_{i-2}, \dots, c_1$ (using, in total, at most $i - 1 + k' \leq \ell + k' < (p+1)|B| + k' \leq k$ moves). Since for all j, we have $c_j \notin \mathsf{alph}(v_j)$, and since the pebble in v is already located to the left of v_i, Duplicator will subsequently have to play in (possibly strict prefixes of) $v_1 c_1 v_2 \cdots c_{i-2}$, $v_1 c_1 v_2 \cdots c_{i-3}$, and so on. It follows that she is not able to answer when Spoiler plays on c_1.

Now suppose that Duplicator plays on a position outside of u_i and v_i, to the right, say on v. Then Spoiler can move the pebble in v subsequently to $c_i, c_{i-1}, c_{i-2}, \dots, c_1$ (using, in total, at most $i + k' \leq k$ moves). For similar reasons as before, Duplicator will not be able to answer in u to each of these moves.

It follows that during the k' rounds that Spoiler plays inside u_i and v_i, Duplicator does too. Thus, $u_i \sim_{k'} v_i$. □

Completeness result using (B, p)-decompositions

The next proposition contains the essential ingredients to prove the completeness of the fixpoint algorithm described in Section 5.2.1. Here, we consider two words that are $\mathsf{FO}^2(<)[k]$-equivalent for a specific k. We show how one can use the (B, p)-decompositions to break these words into suitable factors, to which one can apply induction on the size of the alphabet in order to obtain that pairs of the images of these factors are in the set $\mathtt{Alg}(\alpha)$, the output of the fixpoint algorithm. We then show how recombining these factors, using Operations (1) and (2) of the algorithm, yields that the pair of images of the original words is also in $\mathtt{Alg}(\alpha)$.

Proposition 5.15. *Let M be a monoid, and $\alpha : A^* \to M$ a surjective morphism. Let $f(n) = (2|M|^2 + 3)n^2$. For all $X \subseteq A$, and all $u, v \in X^*$ such that $\mathsf{alph}(u) = \mathsf{alph}(v) = X$ and $u \sim_{f(|X|)} v$, it holds that $(\alpha(u), \alpha(v)) \in \mathtt{Alg}(\alpha)$.*

Proof. We prove this by induction on the size of the alphabet. Let $B \subseteq A$. Our induction hypothesis is that for all $X \subseteq A$ such that $|X| < |B|$, the statement holds. Define $\kappa := f(|B|) = (2|M|^2 + 3)|B|^2$, and $\kappa' := f(|B| - 1) = (2|M|^2 + 3)(|B| - 1)^2$. Let $u, v \in B^*$ be such that $\mathsf{alph}(u) = \mathsf{alph}(v) = B$ and $u \sim_\kappa v$. We want to prove that $(\alpha(u), \alpha(v)) \in \mathtt{Alg}(\alpha)$.

Note that for all $p \leq 2(|M|^2 + 1)$, we have

$$(p+1)|B| \;\leq\; (2|M|^2 + 3)|B| \;\leq\; (2|M|^2 + 3)|B|^2 \;=\; \kappa.$$

Thus, by Lemma 5.13, for all $p \leq 2(|M|^2+1)$, u admits a (B,p)-decomposition if and only if v admits a (B,p)-decomposition. There are thus two cases: either there is a $p \leq 2(|M|^2+1)$ such that u, v both admit a (B,p)-decomposition, or both u and v do not admit a (B,p)-decomposition for any $p \leq 2(|M|^2+1)$. We treat these cases separately.

Case 1. There is a $p \leq 2(|M|^2+1)$ such that both u and v admit a (B,p)-decomposition.

Again by Lemma 5.13, both (B,p)-decompositions have the same length ℓ, and they are given by

$$\begin{aligned} u &= \left(\prod_{i=1}^{\ell} u_i \cdot b_{i \bmod n}\right) \cdot u_{\ell+1}, \\ v &= \left(\prod_{i=1}^{\ell} v_i \cdot b_{i \bmod n}\right) \cdot v_{\ell+1}. \end{aligned}$$

By definition of (B,p)-decompositions, for all $i \in \{1,\dots,\ell+1\}$, we have $b_{i \bmod n} \notin \mathsf{alph}(u_i)$ and $b_{i \bmod n} \notin \mathsf{alph}(v_i)$. We may apply Lemma 5.14, since, as $|B| \geq 1$,

$$\begin{aligned} (p+1)|B| + \kappa' &\leq (2|M|^2+3)|B| + (2|M|^2+3)(|B|-1)^2 \\ &= (2|M|^2+3)(|B|^2-|B|+1) \leq (2|M|^2+3)|B|^2 = \kappa. \end{aligned}$$

It follows that for all i, $u_i \sim_{\kappa'} v_i$. In particular, u_i and v_i have the same alphabet, which is strictly smaller than B since the letter $b_{i \bmod n}$ is not in it. Since $\kappa' = f(|B|-1)$, we can thus use the induction hypothesis to conclude that $(\alpha(u_i), \alpha(v_i)) \in \mathtt{Alg}(\alpha, \mathsf{alph}(u_i))$. Recall from Section 5.2.1 that during the initialization phase, in particular, all pairs of the shape $(\alpha(b_{i \bmod n}), \alpha(b_{i \bmod n}))$ are added to $\mathtt{Alg}(\alpha, \{b_{i \bmod n}\})$. Using that α is a morphism, multiple application of Operation (1) now yields that $(\alpha(u), \alpha(v)) \in \mathtt{Alg}(\alpha, B) \subseteq \mathtt{Alg}(\alpha)$.

Case 2. Neither u nor v admits a (B,p)-decomposition for any $p \leq 2(|M|^2+1)$.

Since $\mathsf{alph}(u) = \mathsf{alph}(v) = B$, u and v are both $(B,0)$-patterns. The fact that they do not admit a (B,p)-decomposition, for any $p \leq 2(|M|^2+1)$, thus means that they are at least $(B, 2|M|^2+3)$-patterns.

We define the words u_l and v_l as the minimal prefix of u resp. v that contains the subword $(b_1 b_2 \cdots b_n)^{|M|^2+1}$. Thus, by construction, u_l and v_l are $(B, |M|^2+1)$-patterns and admit a $(B, |M|^2+1)$-decomposition, which is of length $|B| \cdot (|M|^2+1)$. These decompositions are

$$\begin{aligned} u_l &= \prod_{i=1}^{\ell} u_{l,i} \cdot b_{i \bmod n}, \\ v_l &= \prod_{i=1}^{\ell} v_{l,i} \cdot b_{i \bmod n}, \end{aligned}$$

where $\ell = |B| \cdot (|M|^2+1)$ and for all $i \in \{1,\dots,\ell+1\}$, we have $b_{i \bmod n} \notin \mathsf{alph}(u_{l,i})$ and $b_{i \bmod n} \notin \mathsf{alph}(v_{l,i})$.

Let $\bar{k} = \kappa - |B| \cdot (|M|^2+1)$. We claim that $u_l \sim_{\bar{k}} v_l$. This can be proved using EF games, in a similar fashion as Lemma 5.14. Let Spoiler play inside u_l and v_l during $\bar{k}$ rounds, in the EF game on u and v. Since $u \sim_\kappa v$, Duplicator is able to answer to each of these moves. We show that if she plays outside of u_l and v_l, she will lose. It thus follows that she is able to answer inside u_l and v_l. Suppose that Duplicator plays outside u_l and v_l, i.e. to the right, say on word u. Now Spoiler can keep moving the pebble on u to the left by playing subsequently on $b_{|B|(|M|^2+1) \bmod n}, \dots, b_{1 \bmod n}$. When Spoiler starts playing this sequence of moves, the pebble on v is at, or to the left of, $b_{|B|(|M|^2+1) \bmod n}$. Since, for all i, $b_{i \bmod n} \notin \mathsf{alph}(u_{l,i})$, she will lose. Note that at most $\bar{k} + |B|(|M|^2+1) = \kappa$ rounds have been played. It follows that

$$u_l \sim_{\bar{k}} v_l. \tag{5.2}$$

We define u_r and v_r in a symmetric way, as the minimal suffix of u resp. v that contains the subword $(b_1 b_2 \cdots b_n)^{|M|^2+1}$. Again, by construction, these are $(B, |M|^2+1)$-patterns. Note that one can adapt our notion of (B,p)-decomposition to define a right-most decomposition. By construction of u_r and v_r, their right-most decompositions are

$$\begin{aligned} u_r &= \Big(\textstyle\prod_{i=1}^{\ell} b_{i \bmod n} \cdot u_{r,i}\Big), \\ v_r &= \Big(\textstyle\prod_{i=1}^{\ell} b_{i \bmod n} \cdot v_{r,i}\Big), \end{aligned}$$

where, $\ell = |B| \cdot (|M|^2+1)$, and, for all i, $b_{i \bmod n} \notin \mathsf{alph}(u_{r,i})$. Using this notion, we can apply an argument symmetric to the one above, and obtain, for $\bar{k} = \kappa - |B| \cdot (|M|^2+1)$, that

$$u_r \sim_{\bar{k}} v_r. \tag{5.3}$$

Now, since u and v are $(B, 2|M|^2+3)$-patterns, there must be words u_c and v_c such that $u = u_l u_c u_r$ and $v = v_l v_c v_r$ (i.e. the defined prefix and suffix do not overlap). Since $\mathsf{alph}(u) = \mathsf{alph}(v) = B$, we have that $\mathsf{alph}(u_c) \subseteq B$ and $\mathsf{alph}(v_c) \subseteq B$.

Our approach now is the following. We want to divide u and v into parts on which we can use the induction hypothesis, and which are suitable to apply the operations of the fixpoint algorithm to. Contrary to Case 1, we no longer have an upper bound on the decompositions that u and v admit. Thus, it no longer suffices to just use Operation (1), as worked in Case 1. We will need to use Operation (2), and, therefore, we will break the words up as shown in Figure 5.1, in such a way that we can harmlessly repeat u_e, u_f, v_e and v_f, that is, without changing the image of the words under α.

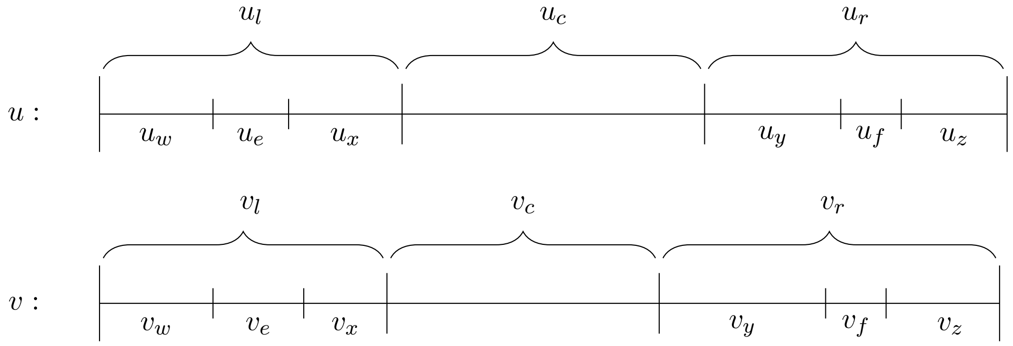


Figure 5.1: Desired factorizations of u and v.

Consider the decompositions of u_l and v_l. By construction, these look like

$$\begin{aligned} u_l &= \textstyle\prod_{i=1}^{\ell} u_{l,i} \cdot b_{i \bmod n}, \\ v_l &= \textstyle\prod_{i=1}^{\ell} v_{l,i} \cdot b_{i \bmod n}, \end{aligned}$$

where $\ell = |B| \cdot (|M|^2 + 1)$. Denote $\alpha(u_{l,i} b_{i \bmod n})$ by s_i and $\alpha(v_{l,i} b_{i \bmod n})$ by t_i. We consider the following two sequences of $|M|^2 + 1$ monoid elements, which are the images of longer and longer prefixes of the decompositions.

$$\begin{array}{llllll} s_1 \cdots s_{|B|}, & s_1 \cdots s_{|B| \cdot 2}, & s_1 \cdots s_{|B| \cdot 3}, & \ldots, & s_1 \cdots s_{|B|(|M|^2+1)}, & \text{and} \\ t_1 \cdots s_{|B|}, & t_1 \cdots t_{|B| \cdot 2}, & t_1 \cdots t_{|B| \cdot 3}, & \ldots, & t_1 \cdots t_{|B|(|M|^2+1)}. & \end{array}$$

Note that there are only $|M|^2$ different pairs of elements of M. It follows that there must be g, h such that $g < h$, and both

$$\begin{array}{rcl} s_1 \cdots s_{|B| \cdot g} & = & s_1 \cdots s_{|B| \cdot g} s_{|B| \cdot g+1} \cdots s_{|B| \cdot h}, \text{ and} \\ t_1 \cdots t_{|B| \cdot g} & = & t_1 \cdots t_{|B| \cdot g} t_{|B| \cdot g+1} \cdots t_{|B| \cdot h}. \end{array}$$

Applying these equalities multiple times gives that

$$\begin{array}{rcl} s_1 \cdots s_{|B| \cdot g} & = & s_1 \cdots s_{|B| \cdot g} (s_{|B| \cdot g+1} \cdots s_{|B| \cdot h})^{\omega}, \text{ and} \\ t_1 \cdots t_{|B| \cdot g} & = & t_1 \cdots t_{|B| \cdot g} (t_{|B| \cdot g+1} \cdots t_{|B| \cdot h})^{\omega}. \end{array}$$

The factors that give rise to $s_{|B| \cdot g+1} \cdots s_{|B| \cdot h}$ resp. $t_{|B| \cdot g+1} \cdots t_{|B| \cdot h}$ will be our u_e resp. v_e. That is, we define,

$$\begin{array}{rclrcl} u_w & = & \prod_{i=1}^{|B| \cdot g} u_{l,i} \cdot b_{i \bmod n}, & v_w & = & \prod_{i=1}^{|B| \cdot g} v_{l,i} \cdot b_{i \bmod n}, \\ u_e & = & \prod_{i=|B| \cdot g+1}^{|B| \cdot h} u_{l,i} \cdot b_{i \bmod n}, & v_e & = & \prod_{i=|B| \cdot g+1}^{|B| \cdot h} v_{l,i} \cdot b_{i \bmod n}, \\ u_x & = & \prod_{i=|B| \cdot h+1}^{\ell} u_{l,i} \cdot b_{i \bmod n}, & v_x & = & \prod_{i=|B| \cdot h+1}^{\ell} v_{l,i} \cdot b_{i \bmod n}. \end{array}$$

Note that the factors u_e and v_e contain the following distinguished letters,

$$b_{|B| \cdot g+1 \bmod n}, \; b_{|B| \cdot g+2 \bmod n}, \; \ldots, \; b_{|B| \cdot h \bmod n}.$$

Since $g < h$, this means that they contain all letters from B, and it follows that $\mathsf{alph}(u_e) = \mathsf{alph}(v_e) = B$.

In (5.2), we saw that $u_l \sim_{\bar{k}} v_l$. We want to use Lemma 5.14 to see how the factors of u_l and v_l are related. Let us first verify that Lemma 5.14 may be applied to u_l and v_l. Indeed, for $\kappa' = f(|B| - 1) = (2|M|^2 + 3)(|B| - 1)^2$, since $|B| \geq 1$, we have

$$\begin{array}{rcl} (|M|^2 + 2)|B| + \kappa' & = & (|M|^2 + 2)|B| + (2|M|^2 + 3)(|B| - 1)^2 \\ & = & (2|M|^2 + 3)|B| - |B|(|M|^2 + 1) + (2|M|^2 + 3)(|B| - 1)^2 \\ & = & -|B|(|M|^2 + 1) + (2|M|^2 + 3)|B|^2 + (2|M|^2 + 3)(1 - |B|) \\ & \leq & -|B|(|M|^2 + 1) + (2|M|^2 + 3)|B|^2 = \bar{k}. \end{array}$$

Thus, for all $i \in \{1, \ldots, |B| \cdot (|M|^2 + 1)\}$, Lemma 5.14 gives that $u_{l,i} \sim_{\kappa'} v_{l,i}$, and in particular that $\mathsf{alph}(u_{l,i}) = \mathsf{alph}(v_{l.i})$. By definition, these factors of the decompositions use an alphabet strictly smaller than B. Therefore, for all $i \in \{1, \ldots, |B| \cdot (|M|^2 + 1)\}$, we have by induction hypothesis that $(\alpha(u_{l,i}), \alpha(v_{l,i})) \in \mathtt{Alg}(\alpha, \mathsf{alph}(u_i))$.

Similar to Case 1, the pairs $(\alpha(u_{l,i}), \alpha(v_{l,i}))$ can then be recombined by multiple application of Operation (1), to obtain that $(\alpha(u_w), \alpha(v_w)) \in \mathtt{Alg}(\alpha, \mathsf{alph}(u_w))$ and $(\alpha(u_e), \alpha(v_e)) \in \mathtt{Alg}(\alpha, B)$.

Since u_r and v_r are also $(B, |M|^2+1)$-patterns, and since we saw in (5.3) that $u_r \sim_{\bar{k}} v_r$, we can construct in a similar way the factorizations $u_r = u_y u_f u_z$ and $v_r = v_y v_f v_z$, such that $\alpha(u_y)\alpha(u_f)^\omega = \alpha(u_y)$, $\alpha(v_y)\alpha(v_f)^\omega = \alpha(v_y)$, $(\alpha(u_z), \alpha(v_z)) \in \mathtt{Alg}(\alpha, \mathsf{alph}(u_z))$ and $(\alpha(u_f), \alpha(v_f)) \in \mathtt{Alg}(\alpha, B)$.

Clearly, we also have that $\mathsf{alph}(u_x) \cup \mathsf{alph}(v_x) \cup \mathsf{alph}(u_y) \cup \mathsf{alph}(v_y) \cup \mathsf{alph}(u_c) \cup \mathsf{alph}(v_c) \subseteq B$. Therefore, we can apply Operation (2) to $(\alpha(u_e), \alpha(v_e))$, $u_x u_c u_y$, $v_x v_c v_y$, and $(\alpha(u_f), \alpha(v_f))$. This yields that

$$(\alpha(u_e)^\omega \alpha(u_x u_c u_y)\alpha(u_f)^\omega, \alpha(v_e)^\omega \alpha(v_x v_c v_y)\alpha(v_f)^\omega) \in \mathtt{Alg}(\alpha, B).$$

Finally, Operation (1) gives

$$(\alpha(u_w)\alpha(u_e)^\omega \alpha(u_x u_c u_y)\alpha(u_f)^\omega \alpha(u_z), \alpha(v_w)\alpha(v_e)^\omega \alpha(v_x v_c v_y)\alpha(v_f)^\omega \alpha(v_z)) \in \mathtt{Alg}(\alpha, B).$$

By the above, this term is equal to $(\alpha(u), \alpha(v))$, and thus, $(\alpha(u), \alpha(v)) \in \mathtt{Alg}(\alpha)$. □

It is just a very small step from the previous proposition to the completeness result of the fixpoint algorithm. This is described in the following proposition.

Proposition 5.16. *The fixpoint algorithm described in Section 5.2.1 is complete.*

Proof. We have to show that $\mathrm{I}(\alpha) \subseteq \mathtt{Alg}(\alpha)$. Let $(s,t) \in \mathrm{I}(\alpha)$. Then, for every $k \in \mathbb{N}$, there are $u \in \alpha^{-1}(s), v \in \alpha^{-1}(t)$ such that $u \sim_k v$, and there is $B \subseteq A$ such that $\mathsf{alph}(u) = \mathsf{alph}(v) = B$. This holds in particular for $k = (2|M|^2+3)|A|^2$. Let u and v be the words corresponding to this k. Note that for f as in Proposition 5.15, $k \geq f(|B|)$. Thus, $u \sim_k v$ implies that $u \sim_{f(|B|)} v$, and Proposition 5.15 then yields that $(s,t) = (\alpha(u), \alpha(v)) \in \mathtt{Alg}(\alpha)$. □

5.2.4 Proof of the separation theorem for unambiguous languages

We are now ready to prove Theorem 5.5, the main theorem of this chapter. Let us first restate the theorem.

Theorem 5.5. *Let M be a monoid, let P and Q be subsets of M, and let $\alpha : A^* \to M$ be a surjective morphism. Let $\kappa = (2|M|^2+3)|A|^2$. Then, the following conditions are equivalent.*

(1) The languages $\alpha^{-1}(P)$ and $\alpha^{-1}(Q)$ are $\mathsf{FO}^2(<)$-separable,

(2) The languages $\alpha^{-1}(P)$ and $\alpha^{-1}(Q)$ are $\mathsf{FO}^2(<)[\kappa]$-separable,

(3) The language $[\alpha^{-1}(P)]_{\sim\kappa}$ separates $\alpha^{-1}(P)$ from $\alpha^{-1}(Q)$,

(4) $P \times Q \ \cap\ \mathrm{I}(\alpha) = \varnothing$.

Proof. As we noted before, the implications (3) ⇒ (2) ⇒ (1) are trivial. Also, implication (1) ⇒ (4) follows by definition of $\mathsf{FO}^2(<)$-indistinguishable pairs: if $\alpha^{-1}(P)$ and $\alpha^{-1}(Q)$ are $\mathsf{FO}^2(<)$-separable, then there is some $n \in \mathbb{N}$, such that there is an $\mathsf{FO}^2(<)[n]$-formula that defines a separator. Then there can be no elements of P and Q that form an $\mathsf{FO}^2(<)[n]$-indistinguishable pair. Since, by definition, $\mathrm{I}(\alpha) \subseteq \mathrm{I}_n(\alpha)$, it follows that $P \times Q \ \cap\ \mathrm{I}(\alpha) = \varnothing$.

The only implication that remains to be proven is (4) $\Rightarrow$ (3). It follows from Proposition 5.9 that $\mathtt{Alg}(\alpha) \subseteq \mathrm{I}(\alpha)$, and from the proof of Proposition 5.16 that $\mathrm{I}_\kappa(\alpha) \subseteq \mathtt{Alg}(\alpha)$, for $\kappa = (2|M|^2+3)|A|^2$. Thus, $P \times Q \ \cap \ \mathrm{I}_\kappa(\alpha) \subseteq P \times Q \ \cap \ \mathrm{I}(\alpha)$, which is empty by assumption. It follows that there are no words $w \in \alpha^{-1}(P), w' \in \alpha^{-1}(Q)$ such that $w \sim_\kappa w'$. Thus, $[\alpha^{-1}(P)]_{\sim\kappa} \cap \alpha^{-1}(Q) = \varnothing$. □

5.3 Complexity of separation by unambiguous languages

Our objective in this chapter was to show that the separation problem for the class of unambiguous languages is decidable. We proved this by providing an algorithm that solves this question. The output of this fixpoint algorithm, described in Section 5.2.1, can be seen as a subset of $M \times M \times \mathcal{P}(A)$. This set is of exponential size with respect to the size of the alphabet, and polynomial size with respect to the size of the monoid. Therefore, the fixpoint algorithm will terminate in at most exponentially many steps. Thus, the separation problem can be decided in Exptime with respect to the size of an NFA recognizing the languages. Note that if we take the alphabet as a fixed parameter, the algorithm runs in Ptime with respect to the size of the monoid.

Finding a lower bound for the complexity of this problem, as well as finding a sharp upper bound, are questions that we did not pose ourselves. It is likely that the upper bound that we found can still be improved.

Chapter 6

Locally testable and locally threshold testable languages

6.1 Locally testable and locally threshold testable languages 84

6.1.1 Locally testable languages 86

6.1.2 Locally threshold testable languages 87

6.2 Separation for a fixed counting threshold 88

6.2.1 Common d-patterns 89

6.2.2 Separation theorem for a fixed counting threshold 90

Intermezzo: Relation with the concept of indistinguishable pairs . . 92

6.2.3 A common d-pattern yields equivalent words for all profile sizes . . . 92

6.2.4 From a common d-pattern in M to a common d-pattern in $\mathcal{A}$ 94

6.2.5 Bounding the profile size 95

6.2.6 Decidability of separation by locally testable languages 99

6.3 Separation for full locally threshold testable languages 100

6.3.1 Decidability of separation by locally threshold testable languages . . 100

6.3.2 Bounding the counting threshold 102

6.3.3 Optimality of the bound on the counting threshold 105

6.4 Complexity of LT- and LTT-separability 107

6.4.1 Upper complexity bounds 107

Reduction to the case of $k = 1$ 108

Deciding LT- and LTT-separability for $k = 1$ 112

Results on upper complexity bounds 113

6.4.2 Lower complexity bounds 113

6.5 Separating context-free languages by LT and LTT languages . . . 115

In this chapter, which is based on the papers [PvRZ13a, PvRZ14], we investigate the separation problem for locally testable and locally threshold testable languages. A language is locally testable (LT) if membership of a word in the language only depends on the *set* of infixes, prefixes and suffixes up to some fixed length that occur in the word. For a language that is locally threshold testable (LTT), membership may also depend on the *number* of occurrences of such infixes, which may be counted up to some fixed threshold.

As we have seen in Section 2.2.1, it was proved in [Alm99] that solving the separation problem for a class of separators amounts to computing the 2-pointlike sets for the algebraic variety corresponding to this class. It has been shown that both the varieties corresponding to locally testable languages and to locally threshold testable languages have computable pointlike sets. This is a consequence of [CN09, Cos01] for LT, and of [BP91, Str85, Ste98, Ste01] for LTT. These results prove the decidability of the separation problem for these classes. For LTT, however, this is done in an indirect way. Besides working in an algebraic setting, one uses that LTT corresponds to the variety $\mathsf{Acom}*\mathsf{D}$. The decidability then follows from the transfer result that states that computability of pointlikes is preserved under the operation $\mathsf{V} \mapsto \mathsf{V}*\mathsf{D}$.

Our approach to show that the LT- and LTT-separation problems are decidable is by reduction to fixed parameters. The class LTT comes with two parameters: a first parameter for the length of the factors that determine membership, and a second one for the threshold up to which these factors are counted. The class LT only has the first parameter. In Section 6.1, we provide definitions for both classes. Our approach is then to establish bounds, which depend on the input languages, on the parameters holding for potential separators. For fixed parameters there are only finitely many languages, and thus providing such bounds proves that the separation problems are decidable. Contrary to the algebraic approach, this method also gives a description of a separator in case it exists, in terms of these parameters.

Recently, some of the results in this chapter have been reproved in a generic way in the paper [PZ14c], which gives transfer results for the separation problem for different fragments of FO, to which the successor relation has been added as a predicate. The class LTT corresponds to the logical fragment $\mathsf{FO}(=)$ with the successor relation added, and is one of the fragments to which this transfer result applies.

In Section 6.2, we first restrict ourselves to the class of LTT languages with a fixed threshold. For this situation, we are able to provide a bound on the length of the factors. As a consequence, this solves the LT-separation problem, since the class LT is equal to LTT with fixed threshold 1. It turns out, as we show in Section 6.3, that the same bound still works for the full class LTT. In this section, we also provide a bound on the threshold. In Section 6.4, we give upper and lower complexity bounds for the LT- and LTT-separation problems. The upper bounds are based on forbidden patterns in the NFA or monoid recognizing the input languages. Finally, in Section 6.5, we consider separation by LT and LTT languages when the class of input languages is the class of context-free languages, rather than the class of regular languages.

6.1 Locally testable and locally threshold testable languages

In order to introduce the classes of languages that we consider in this chapter, let us first make some definitions. For a word $w \in A^*$, recall that we denote the *length* or *size* of w as $|w|$. When w is nonempty, we view w as a sequence of $|w|$ positions labeled over A. In this chapter, we number the positions from 0, for the leftmost position, to $|w|-1$, for the rightmost position. Recall also that an *infix* of a word $w \in A^*$ is a word w' such that $w = u \cdot w' \cdot v$ for some $u, v \in A^*$, and that in this case, we say that w' is a *prefix* (resp. a *suffix*) of w, if $u = \varepsilon$ (resp. if $v = \varepsilon$).

Let $0 \leq x < y \leq |w|$. We write $w[x, y]$ for the infix of w starting at position x and ending at position $y - 1$. For example, if $w = abcdabc$, then $w[2, 5] = cda$. By convention, we also define $w[x, x] = \varepsilon$. Observe that by definition, when $x \leq y \leq z$, we have $w[x, z] = w[x, y] \cdot w[y, z]$.

The following definition allows us to work with one single notion instead of the three notions of prefix, infix and suffix.

Definition 6.1. For $k \in \mathbb{N}$, let $k_\ell = \lfloor k/2 \rfloor$ and $k_r = k - k_\ell$. A *k-profile* is a pair of words (w_ℓ, w_r), with $|w_\ell| \leq k_\ell$ and $|w_r| \leq k_r$. Given $w \in A^*$ and a position x of w, *the k-profile of x* is the pair (w_ℓ, w_r) defined as follows: $w_\ell = w[\max(0, x - k_\ell), x]$ and $w_r = w[x, \min(x + k_r, |w|)]$.

For example, in $w = abcdabc$, the 3-profile of $x = 2$ is (b, cd). See also Figure 6.1. The set of k-profiles over A is denoted by A_k. Note that its size is $|A_k| = |A|^{O(k)}$.

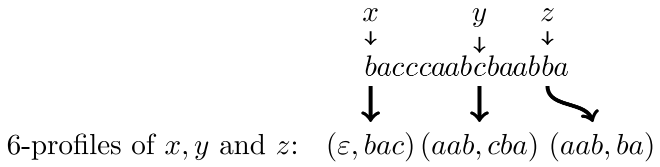


Figure 6.1: Illustration of the notion of k-profile for $k = 6$.

A k-profile is thus a description of an infix of w that is centered at position x. As can be seen in Figure 6.1, the k-profiles of positions close to the beginning or to the end of the word are shorter. Thus, we may use the k-profiles that occur in a word to identify the prefixes and suffixes of length $\leq k - 1$ of this word.

To this end, suppose that $|w| \geq k - 1$ and consider the k-profile of the position $k_\ell - 1$. This is (w_ℓ, w_r), for $w_\ell = w[0, k_\ell - 1]$ and $w_r = w[k_\ell - 1, k - 1]$. Then, $w_\ell \cdot w_r = w[0, k - 1]$ is the prefix of w of length $k - 1$. Similarly, we obtain the suffix of length $k - 1$ from the k-profile of the position $|w| - k_r + 1$. In this case, $w_\ell \cdot w_r = w\big[|w| - (k - 1), |w|\big]$ is the suffix of length $k - 1$.

Note that, if $k = 2$, then the position $|w| - k_r + 1 = |w|$, and therefore lies outside of the word. Since we also want to be able to read the suffix of length 1 from the set of 2-profiles of a word, we introduce a dummy position $|w|$, and define its k-profile as (w_ℓ, w_r), for $w_\ell = w[\max(0, |w| - k_\ell), |w|]$ and $w_r = w[|w|, \min(|w|, |w| + k_r)] = \varepsilon$.

Definition 6.2. A k-profile (w_ℓ, w_r) *occurs in a word w* if there exists some position $x \in \{0, 1, \ldots, |w|\}$ whose k-profile is (w_ℓ, w_r). If n is a natural number, we say that (w_ℓ, w_r) *occurs n times in w* if there are n distinct positions in $\{0, 1, \ldots, |w|\}$ where (w_ℓ, w_r) occurs. In this case, we write $|w|_{(w_\ell, w_r)} = n$.

The set of all k-profiles over the alphabet A is denoted by A_k. Its size is $|A_k| = |A|^{O(k)}$.

Definition 6.3. We say that two numbers $n, m \in \mathbb{N}$ are *equal up to threshold* d if $n = m$, or both $n, m \geq d$. For two words w, w', we write $w \equiv_k^d w'$, if for every k-profile (w_ℓ, w_r), it holds that the numbers $|w|_{(w_\ell,w_r)}$ and $|w'|_{(w_\ell,w_r)}$ are equal up to threshold d.

One can verify that $\equiv_k^d$ is an equivalence relation (and actually a congruence) of finite index. The $\equiv_k^d$-equivalence class of a word w is denoted as $[w]_k^d$.

6.1.1 Locally testable languages

A language is called *locally testable* if, indeed, it can be tested locally whether a word belongs to the language or not. To be more precise, membership of a word in a locally testable language can be tested by inspecting its prefixes, suffixes and infixes up to some length (which depends on the language).

Definition 6.4. A language is *locally testable* (LT) if it is a finite boolean combination of languages of the following form.

1. $uA^* = \{w \mid u \text{ is a prefix of } w\}$, for some $u \in A^*$.
2. $A^*u = \{w \mid u \text{ is a suffix of } w\}$, for some $u \in A^*$.
3. $A^*uA^* = \{w \mid u \text{ is an infix of } w\}$, for some $u \in A^*$.

The next lemma allows us to describe the class of locally testable languages in terms of k-profiles rather than prefixes, infixes and suffixes.

Lemma 6.5 (adaptation of [BP91, Proposition 2.1])**.** *The class of locally testable languages is the class of languages that are unions of $\equiv_k^1$-classes, for some $k \in \mathbb{N}$.*

Proof. Let L be a locally testable language. Let i be the maximal length of the infixes occurring in the boolean expression for L, and let p resp. s be the maximal length of prefixes resp. suffixes occurring in this expression. Define $k = \max(i, p+1, s+1)$. Let $w \in L$ and let $w \equiv_k^1 w'$. By definition of the $\equiv_k^1$-equivalence, w and w' have the same prefixes of length $\leq k-1 \leq p$, the same suffixes of length $\leq k-1 \leq s$ and the same infixes of length $\leq k \leq i$. It follows that $w' \in L$. For the converse direction, it suffices to show that every class of the form $[w]_k^1$ is a finite boolean combination of languages as above. To this end, define $P_w = \{u \in A^{\leq k-1} \mid \exists v \in A^*.\ w = uv\}$, $S_w = \{u \in A^{\leq k-1} \mid \exists v \in A^*.\ w = vu\}$, $I_w = \{u \in A^{\leq k} \mid \exists v, v' \in A^*.\ w = vuv'\}$, and $J_w = \{u \in A^{\leq k} \mid \forall v, v' \in A^*.\ w \neq vuv'\}$. Now,

$$[w]_k^1 = \Big(\bigcap_{u \in P_w} uA^* \cap \bigcap_{u \in S_w} A^*u \cap \bigcap_{u \in I_w} A^*uA^* \Big) \setminus \bigcup_{u \in J_w} A^*uA^*. \qquad \square$$

Since every LT language is a finite boolean combination of languages of the form uA^*, A^*uA^* and A^*u, the class LT is a subclass of the FO(<)-definable languages. For example, uA^*, for $u = u_1 \cdots u_n$ with all u_i's in A, is defined by the following FO(<)-formula,

$$\exists x_1. \dots \exists x_n. \Big(\forall y.\ \neg(y < x_1) \wedge \bigwedge_{i=2,\dots,n} \big(x_{i-1} < x_i \wedge \forall y.\ \neg(x_{i-1} < y \wedge y < x_i)\big) \wedge u_1(x_1) \wedge \dots u_n(x_n)\Big).$$

However, no simple description of LT in terms of first-order logic is known. In terms of linear temporal logic, LT languages are exactly those defined by formulas involving only the operators F (eventually) and X (next), with no nesting of F operators.

For all $k \in \mathbb{N}$, we denote the set of languages that are unions of $\equiv_k^1$-classes by $\mathsf{LT}[k]$. Thus, $\mathsf{LT} = \bigcup_k \mathsf{LT}[k]$. Given $L \subseteq A^*$ and $k \in \mathbb{N}$, the smallest $\mathsf{LT}[k]$-language containing L is

$$[L]_k^1 = \{w \in A^* \mid \exists u \in L \text{ such that } u \equiv_k^1 w\}.$$

However, there is in general no smallest LT language containing a given regular language. For example consider the language $L = (aa)^*$ over $A = \{a\}$. Let L' be an LT language that contains L. Then, L' is an $\mathsf{LT}[k]$-language for some k. Now, for n such that $|(aa)^n| \geq k$, we have that, for every $m \in \mathbb{N}$, $(aa)^n \equiv_k^1 (aa)^n a^m$. Thus, L' is of the shape $(aa)^n a^* \cup F$, with F finite. There is no smallest LT language of this shape: one can always remove a^{2n+1} from L' to obtain a smaller LT language that still contains $(aa)^*$.

The membership problem for the class of locally testable languages was raised by McNaughton and Papert [MP71], and solved independently by Zalcstein, Brzozowski and Simon, and McNaughton [Zal72, BS73, McN74]. This was done by characterizing the syntactic semigroups of locally testable languages. A language is locally testable if and only if its syntactic semigroup is in LSl. This is the variety of all finite locally idempotent and commutative semigroups, also called local semilattices. This means that a semigroup S belongs to LSl if and only if, for every idempotent $e \in S$, the semigroup eSe belongs to Sl. Therefore, LSl consists of all finite semigroups that satisfy, for all $e \in E(S)$ and for all $s, t \in S$, $esese = ese$ and $esete = etese$.

If the input language is given by a deterministic automaton, the membership problem for LT is in PTIME [KMM89].

6.1.2 Locally threshold testable languages

An extension of the class of locally testable languages is the class of locally threshold testable languages. In this class, the occurrence of infixes is counted up to some threshold.

Definition 6.6. A language is *locally threshold testable* (LTT) if it is a finite boolean combination of languages of the following form.

1. $uA^* = \{w \mid u \text{ is a prefix of } w\}$, for some $u \in A^*$.
2. $A^*u = \{w \mid u \text{ is a suffix of } w\}$, for some $u \in A^*$.
3. $\{w \mid u \text{ occurs at least } d \text{ times as an infix of } w\}$, for some $u \in A^*$ and $d \in \mathbb{N}$.

Again, since the k-profiles of a word not only determine the infixes of length k, but also determine the prefixes and suffixes of length $\leq k-1$ of the word, an LTT language can be defined purely in terms of k-profiles counted up to a threshold. Similar to Lemma 6.5, one can prove that this class is exactly the class of languages that are unions of $\equiv_k^d$-classes, for some $k, d \in \mathbb{N}$ depending on the language. The natural number d is called the *counting threshold*.

For $k, d \in \mathbb{N}$, let us denote by $\mathsf{LTT}[k, d]$ the set of the finitely many languages that are unions of $\equiv_k^d$-classes. By definition, we have $\mathsf{LTT} = \bigcup_{k,d} \mathsf{LTT}[k, d]$. Given $L \subseteq A^*$, the smallest $\mathsf{LTT}[k, d]$-language containing L is

$$[L]_k^d = \{w \in A^* \mid \exists u \in L \text{ such that } u \equiv_k^d w\}.$$

As before, there is in general no smallest LTT language containing a given regular language. The same example with $L = (aa)^*$ over $A = \{a\}$ also works to show this for the class of LTT languages.

The class of LTT languages can be defined in terms of first-order logic: a language is LTT if and only if it can be defined by an $\mathsf{FO}(=, +1)$-formula, i.e., a first-order logic formula using predicates for the equality and the successor relation, but not for the linear order. See [BP91, Tho82]. It was shown in [TW85] that the fragment $\mathsf{FO}(=, +1)$ corresponds to the class of languages recognized by the semidirect product $\mathsf{Acom} * \mathsf{D}$. This gives a decidable characterization for the class of locally threshold testable languages. Membership can be tested in PTIME [Pin96, Pin05, Tra01a], if the input language is given by a deterministic automaton.

6.2 Separation for a fixed counting threshold

Before studying the full LTT-separation problem in Section 6.3, we first restrict ourselves to a simpler problem. It turns out that the results for this simpler problem will be very useful when dealing with the full LTT-separation problem. In this section, we fix $d \in \mathbb{N}$ and look at the separation problem for the class of locally threshold testable languages with counting threshold d. We prove that this is a decidable problem. That is, we prove that, for a fixed $d \in \mathbb{N}$ and two regular languages, it is decidable whether there exists a $k \in \mathbb{N}$ such that the languages are separable by an $\mathsf{LTT}[k, d]$-language. In particular, this proves the decidability of the separation problem for LT, as this class corresponds to LTT with counting threshold $d = 1$. All results in this section are for an arbitrary fixed d. Theorem 6.12 states the main results of this section. It contains the following two contributions.

(1) First, we establish a bound k on the size of profiles, such that it suffices to consider only profiles up to this size in order to see whether the input languages can be separated. This bound only depends on the size of a monoid recognizing these languages, and it can be computed. One can use this bound in a brute-force algorithm that tests separability by all the finitely many $\mathsf{LTT}[k, d]$-languages.

(2) The second contribution is a criterion on the input languages to check whether there exists a k such that the languages are $\mathsf{LTT}[k, d]$-separable. This criterion can be defined equivalently on an automaton or a monoid recognizing the input languages, in terms of the absence of common patterns. Using this criterion, we bypass the brute force algorithm and obtain a better complexity result. We will discuss this complexity result in Section 6.4.1.

We will see in Section 6.3 that the bound k on the size of the profiles from (1) actually also works for the full LTT-separation problem.

In Section 6.2.1, we define the criterion on automata or monoids recognizing the input languages. This criterion identifies the so-called d-indistinguishable pairs of monoid elements, or of pairs of states. We will come back to this in Section 6.2.2. Our separation theorem for the class of LTT languages with a fixed counting threshold is also stated in Section 6.2.2. In Sections 6.2.3, 6.2.4, and 6.2.5, we prove the different implications of this theorem.

6.2.1 Common d-patterns

In this section, we define a criterion that must be satisfied by two languages in order for these to be $\mathsf{LTT}[k,d]$-separable for some k. One can equivalently define the criterion on an automaton or on a monoid recognizing the languages. Here, we present both of these definitions. The criterion states the absence of common patterns of a certain shape, determined by d. Recall that d is fixed throughout Section 6.2. The patterns will identify factors that can be pumped without changing the syntactical value of a word, such that a common pattern leads to witnesses of non-$\mathsf{LTT}[k,d]$-separability for arbitrarily large k. To make this more precise, we first introduce the relevant notions.

Definition 6.7. A *block* is a triple of words $\mathfrak{b} = (v_\ell, u, v_r)$ where $v_\ell, v_r \neq \varepsilon$. A *prefix block* is a pair of words $\mathfrak{p} = (u, v_r)$ with $v_r \neq \varepsilon$, and a *suffix block* is a pair of words $\mathfrak{s} = (v_\ell, u)$ with $v_\ell \neq \varepsilon$.

Definition 6.8. Let $d \in \mathbb{N}$. A *d-pattern* $\mathcal{P}$ is either a word w, or is a triple $(\mathfrak{p}, f, \mathfrak{s})$ where $\mathfrak{p}$ and $\mathfrak{s}$ are respectively a prefix and a suffix block, and f is a function from the set of blocks to the set $\{0, \dots, d\}$, such that all but finitely many blocks are mapped to 0.

An example of a 3-pattern is $\mathcal{P} = ((a,b), f, (b,a))$, with f sending (b,a,b) to 3, and all other blocks to 0.

Definition 6.9. Let w be a word and let $\mathcal{P}$ be a d-pattern. We say that w *admits a $\mathcal{P}$-decomposition* if w can be decomposed as $w = u_0v_1u_1v_2 \cdots v_nu_n$ with $n \geq 0$ and such that either $n = 0$ and $\mathcal{P} = u_0 = w$, or $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$ and the following conditions are verified,

1. $\mathfrak{p} = (u_0, v_1)$ and $\mathfrak{s} = (v_n, u_n)$,
2. for every block $\mathfrak{b}$, if $f(\mathfrak{b}) < d$, then $\big|\{i \mid (v_i, u_i, v_{i+1}) = \mathfrak{b}\}\big| = f(\mathfrak{b})$,
3. for every block $\mathfrak{b}$, if $f(\mathfrak{b}) = d$, then $\big|\{i \mid (v_i, u_i, v_{i+1}) = \mathfrak{b}\}\big| \geq d$.

We may say $\mathcal{P}$-decomposition to mean a $\mathcal{P}$-decomposition *of some word*.

For example, consider the 3-pattern $\mathcal{P} = ((a,b), f, (b,a))$, with f sending (b,a,b) to 3, and all other blocks to 0. All words of the shape $w = a(bab)^{\geq 3}a$ admit a $\mathcal{P}$-decomposition. To see this, take, for every i, $u_i = a$ and $v_i = b$. Then, $w = u_0v_1u_1v_2 \cdots v_nu_n$ for some n, and $(u_0, v_1) = (a,b)$, $(v_n, u_n) = (b,a)$, and $\big|\{i \mid (v_i, u_i, v_{i+1}) = (b,a,b)\}\big| \geq 3$.

In the following definition we use d-patterns to express that some factors of a word may be pumped without changing the syntactical value of the word.

Definition 6.10. Let $\alpha : A^* \to M$ be a morphism into a monoid M, and let $s \in M$. A $\mathcal{P}$-decomposition of a word w is said to be (α, s)*-compatible* if $\alpha(w) = s$ and, for all $1 \leq i \leq n$, $\alpha(u_0 \cdots v_i) = \alpha(u_0 \cdots v_i) \cdot \alpha(v_i)$. Similarly, if p, q are two states of an automaton $\mathcal{A}$, a $\mathcal{P}$-decomposition of the word w is (p, q)*-compatible* if there is a run from p to q for w, such that for all $1 \leq i \leq n$, each infix v_i labels a loop in the run, as depicted in Figure 6.2. Here, edges denote transition sequences.

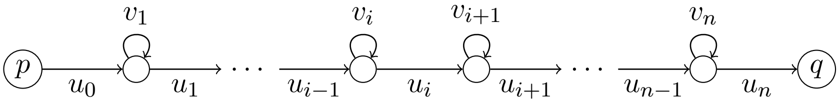


Figure 6.2: A (p, q)-compatible $\mathcal{P}$-decomposition of $w = u_0 v_1 \cdots v_n u_n$.

The intuition behind the d-patterns is the following. If a d-pattern is compatible with elements of a recognizing set of a monoid, or with initial-final pairs of states of an automaton, then it defines a subset of the regular language recognized by the monoid or the automaton. For automata, this subset is the language recognized by the automaton of Figure 6.2, with p as an initial and q as a final state. This language thus gives information about the prefix, suffix and infixes, occurring up to threshold d, present in words of the language. For instance, the (p, q)-compatible $\mathcal{P}$-decomposition of Figure 6.2 implies that for every $m \in \mathbb{N}$, there is a word in the language that has prefix $u_0 v_1^m$, a word that has suffix $v_n^m u_n$, and the same for infixes (with counting) corresponding to the d-pattern.

Since two languages are $\mathsf{LTT}[k, d]$-separable if and only if there are no words in the two respective languages that share the same prefix and suffix of length $k - 1$ and infixes of length k up to threshold d, it is interesting to compare the d-patterns of the two languages. To this end, we define the following notion.

Definition 6.11. Let $d \in \mathbb{N}$ and $\alpha : A^* \to M$ be a morphism into a finite monoid. We say that a pair $(s_1, s_2) \in M \times M$ has a *common d-pattern for α* if there exist a d-pattern $\mathcal{P}$ and two $\mathcal{P}$-decompositions of (possibly different) words that are respectively (α, s_1)-compatible and (α, s_2)-compatible. Usually, the morphism α is clear from the context and we just speak about a *common d-pattern*, without mentioning α. Also, if $\mathcal{A}$ is an automaton, and p_1, q_1, p_2, q_2 are states of $\mathcal{A}$, we say that the pair $\big((p_1, q_1), (p_2, q_2)\big)$ has a *common d-pattern* if there exist a d-pattern $\mathcal{P}$ and two $\mathcal{P}$-decompositions of words that are respectively (p_1, q_1)-compatible and (p_2, q_2)-compatible.

In particular, the pair $\big((p_1, q_1), (p_2, q_2)\big)$ has a common 1-pattern if and only if there are paths in $\mathcal{A}$ of the form shown in Figure 6.2 with the same *set* of triples (v_i, u_i, v_{i+1}) and same beginning and ending, going respectively from p_1 to q_1 and from p_2 to q_2. The notion of 1-pattern is pertinent to the separation problem for the class of locally testable languages.

6.2.2 Separation theorem for a fixed counting threshold

The reason why common d-patterns are interesting when studying the separation problem for locally threshold testable languages, is the following. A d-pattern identifies factors of a word that may be pumped without changing the syntactical value of the word. A common d-pattern implies that this may be done simultaneously in both words. We will show that this means that for any k, words in the languages can be constructed that are $\equiv_k^d$-equivalent. In fact, we will show that the property of a pair $(s_1, s_2) \in M \times M$ of having a common d-pattern is necessary and sufficient for the languages $\alpha^{-1}(s_1)$ and $\alpha^{-1}(s_2)$ to *not* be separable by an $\mathsf{LTT}[k, d]$-language, for any k. A similar statement holds for common d-patterns in NFAs. The difficult part of the proof is showing that this condition is necessary.

Furthermore, having a common d-pattern is a decidable property, which makes it particularly useful for our purposes. The next theorem contains the main results of this section, and states the relation between the absence of common d-patterns and the existence of an $\ell \in \mathbb{N}$ such that the languages are $\mathsf{LTT}[\ell, d]$-separable. We devote Sections 6.2.3, 6.2.4, and 6.2.5 to the proof of this theorem.

Theorem 6.12. *Fix $d \in \mathbb{N}$. Let L_1, L_2 be regular languages. Let $\alpha : A^* \to M$ be a morphism into a finite monoid M recognizing both L_1 and L_2. Let $\mathcal{A}$ be an NFA recognizing both L_1 and L_2, with $L_i = L(\mathcal{A}, I_i, F_i)$. Let $k = 4(|M| + 1)$. Then, the following conditions are equivalent.*

(1) L_1 and L_2 are $\mathsf{LTT}[\ell, d]$-separable for some ℓ,

(2) L_1 and L_2 are $\mathsf{LTT}[k, d]$-separable,

(3) The language $[L_1]_k^d$ separates L_1 from L_2,

(4) There is no pair in $\alpha(L_1) \times \alpha(L_2)$ with a common d-pattern,

(5) There is no pair in $(I_1 \times F_1) \times (I_2 \times F_2)$ with a common d-pattern.

Observe that equivalence (1) ⇔ (2) of this theorem is like a *delay theorem* [Str85, Ste01], for separation restricted to the class of LTT with fixed d, since we prove that the size of profiles that a potential separator needs to consider can be bounded by a function of the size of the monoids recognizing the languages.

The equivalence (1) ⇔ (2) yields an algorithm to decide LTT-separability for a fixed threshold. Indeed, an algorithm that tests all the finitely many $\mathsf{LTT}[k, d]$-languages, for $k = 4(|M| + 1)$, as potential separators solves this question. This gives Corollary 6.13. However, this brute-force approach yields a very costly procedure. It turns out that a more practical algorithm can be obtained from Conditions (4) and (5). We postpone the presentation of this algorithm to Section 6.4.1.

Corollary 6.13. *Let $d \in \mathbb{N}$. It is decidable whether two given regular languages are $\mathsf{LTT}[\ell, d]$-separable for some $\ell \in \mathbb{N}$.*

Instantiating Theorem 6.12 for $d = 1$ gives five equivalent conditions for LT-separability, and in particular yields an algorithm to decide LT-separability. Since this is an interesting result in itself, we focus on this result in Section 6.2.6. After a brief intermezzo about indistinguishable pairs, we will prove Theorem 6.12 for arbitrary, fixed, d. Note that the implications (3) ⇒ (2) ⇒ (1) are immediate by definition. In Sections 6.2.3, 6.2.4, and 6.2.5, we will subsequently prove the implications (1) ⇒ (5) ⇒ (4) ⇒ (3).

Intermezzo: Relation with the concept of indistinguishable pairs

Before proving the separation theorem for LTT languages with fixed counting threshold d, let us briefly mention the relation between the approach to studying the LTT-separation problem that we take here, and our approach from Section 2.2.3.

Let $\alpha : A^* \to M$ be a morphism. If a pair $(s_1, s_2) \in M \times M$ has a common d-pattern for α, it follows from Theorem 6.12, which we will prove in the following sections, that the languages $\alpha^{-1}(s_1)$ and $\alpha^{-1}(s_2)$ are not $\mathsf{LTT}[\ell, d]$-separable, for any $\ell \in \mathbb{N}$. Using our terminology from Section 2.2.3, we can thus call such a pair *d-indistinguishable for α*, or simply *d-indistinguishable*. We denote the set of such pairs by $\mathrm{I}_d(\alpha)$. One could similarly define the d-indistinguishable pairs of pairs of states in $Q^2 \times Q^2$. If a pair is d-indistinguishable for every $d \in \mathbb{N}$, we say that it is an indistinguishable pair. We denote the set of these pairs by $\mathrm{I}(\alpha)$.

Note that a d-pattern $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$ gives rise to a $(d-1)$-pattern $\mathcal{P}' = (\mathfrak{p}, f', \mathfrak{s})$, simply by defining

$$f'(\mathfrak{b}) \quad = \begin{cases} f(\mathfrak{b}) & \text{if } f(\mathfrak{b}) < d-1 \\ d-1 & \text{if } f(\mathfrak{b}) \geq d-1 \end{cases}$$

It is immediate that a $\mathcal{P}$-decomposition of a word w is also a $\mathcal{P}'$-decomposition. Recall that a pair $(s_1, s_2) \in M \times M$ has a common d-pattern if there is a d-pattern $\mathcal{P}$ and words w_1, w_2 that have (α, s_1)- resp. (α, s_2)-compatible $\mathcal{P}$-decompositions. This yields that a d-indistinguishable pair (s_1, s_2) is also $(d-1)$-indistinguishable, and we thus have the following inclusions.

$$\mathrm{I}(\alpha) = \bigcap_{n \in \mathbb{N}} \mathrm{I}_n(\alpha) \subseteq \ldots \subseteq \mathrm{I}_d(\alpha) \subseteq \mathrm{I}_{d-1}(\alpha) \subseteq \ldots \subseteq \mathrm{I}_1(\alpha). \tag{6.1}$$

When dealing with LTT for a fixed counting threshold d, we are interested in computing one single level of this sequence. However, in Section 6.3, we consider the full class of LTT languages and we are interested in the limit behavior of this sequence. From the fact that this sequence is growing with respect to the inclusion order, while all elements of the sequence are in the finite set $\mathcal{P}(M \times M)$, it follows that there must be an index for which the sequence stabilizes. In Section 6.3.2, we establish a bound on d which gives such a stabilization index for the sequence. This bound depends on the size of the recognizing monoid or the number of states in the automaton, and the size of the alphabet.

6.2.3 A common d-pattern yields equivalent words for all profile sizes

We now prove the implication (1) $\Rightarrow$ (5) of Theorem 6.12, by contraposition. That is, we prove that if there exists a pair $\big((p_1, q_1), (p_2, q_2)\big) \in (I_1 \times F_1) \times (I_2 \times F_2)$ that has a common d-pattern, then there does not exist any $\ell \in \mathbb{N}$ such that L_1 and L_2 are $\mathsf{LTT}[\ell, d]$-separable. In fact, we prove that if $\big((p_1, q_1), (p_2, q_2)\big)$ has a common d-pattern, then precisely these states will, for every ℓ, give rise to words that are too closely related to be $\mathsf{LTT}[\ell, d]$-separable. Let us formulate this in the following proposition.

Proposition 6.14. *Let $d \in \mathbb{N}$ and let $\mathcal{A}$ be an NFA. Let p_1, q_1, p_2, q_2 be states of $\mathcal{A}$. If $\big((p_1, q_1), (p_2, q_2)\big)$ has a common d-pattern, then, for all $\ell \in \mathbb{N}$, there exist $w_1 \in L(\mathcal{A}, \{p_1\}, \{q_1\})$, $w_2 \in L(\mathcal{A}, \{p_2\}, \{q_2\})$, such that $w_1 \equiv_\ell^d w_2$.*

Proof. Let $L_1 = L(\mathcal{A}, \{p_1\}, \{q_1\})$ and $L_2 = L(\mathcal{A}, \{p_2\}, \{q_2\})$. The pair $\big((p_1, q_1), (p_2, q_2)\big)$ has a common d-pattern, thus by definition, there exists a d-pattern $\mathcal{P}$, and words $z_1 \in L_1$, $z_2 \in L_2$ that admit (p_1, q_1)- resp. (p_2, q_2)-compatible $\mathcal{P}$-decompositions. If $\mathcal{P} = w \in A^*$, then, $z_1 = w = z_2$, and, clearly, $z_1 \equiv_\ell^d z_2$, for all $\ell \in \mathbb{N}$.

Otherwise, $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$. Let $z_1 = u_0 v_1 u_1 v_2 \cdots v_n u_n$ and $z_2 = x_0 y_1 x_1 y_2 \cdots y_m x_m$ be the (p_1, q_1)- resp. (p_2, q_2)-compatible $\mathcal{P}$-decompositions of z_1 and z_2. For $\ell \in \mathbb{N}$, define

$$\begin{aligned} w_1 &= u_0 v_1^{\ell(d+1)} u_1 v_2^{\ell(d+1)} \cdots v_n^{\ell(d+1)} u_n, \\ w_2 &= x_0 y_1^{\ell(d+1)} x_1 y_2^{\ell(d+1)} \cdots y_m^{\ell(d+1)} x_m. \end{aligned}$$

It follows from the definition of compatibility that $w_1 \in L_1$ and $w_2 \in L_2$. We claim that $w_1 \equiv_\ell^d w_2$.

Since the $\mathcal{P}$-decompositions of z_1 and z_2 use the same d-pattern $\mathcal{P}$, we deduce that $u_0 = x_0$, $v_1 = y_1$, $v_n = y_m$, and $u_n = x_m$. Recall that by definition of a d-pattern, all v_i's and y_i's are nonempty. Thus, in particular, w_1 and w_2 have the same prefix of length $\ell - 1$, and the same suffix of length $\ell - 1$.

To show the claim, we furthermore have to show that each word of length at most ℓ occurs the same number of times, up to threshold d, as an infix in w_1 and in w_2. Let u be an infix of length at most ℓ of, say, w_1. There are two cases.

(1) There is an index i such that u is an infix of $v_i^{\ell(d+1)}$. Then, u occurs at least d times in $v_i^{\ell(d+1)}$, that is, at least d times in w_1. Since the decompositions of z_1, z_2 are $\mathcal{P}$-decompositions, there exists j such that $y_j = v_i$. Therefore, u occurs at least d times as an infix in $y_j^{\ell(d+1)}$, hence also in w_2.

(2) The word u is not an infix of any of the $v_i^{\ell(d+1)}$'s. Then, it must use one of the u_i's. Since $|u| \leq \ell$ and, for all i, $|v_i| \geq 1$, this means that it is either an infix of $u_0 v_1^{\ell(d+1)}$, of $v_n^{\ell(d+1)} u_n$, or there is an index i such that u is an infix of $v_i^{\ell(d+1)} u_i v_{i+1}^{\ell(d+1)}$. If it is an infix of $u_0 v_1^{\ell(d+1)}$, then also of $x_0 y_1^{\ell(d+1)} = u_0 v_1^{\ell(d+1)}$, and if it is an infix of $v_n^{\ell(d+1)} u_n$, then also of $y_m^{\ell(d+1)} x_m = v_n^{\ell(d+1)} u_n$.

   Now assume that u is an infix of some $v_i^{\ell(d+1)} u_i v_{i+1}^{\ell(d+1)}$. Since the decompositions of z_1, z_2 are $\mathcal{P}$-decompositions, the number of triples (v_i, u_i, v_{i+1}) in the decomposition of w_1 and the number of triples (y_j, x_j, y_{j+1}) in that of w_2 which are equal to a given triple is the same, up to threshold d. Thus, u occurs the same number of times up to threshold d in both w_1 and w_2.

It follows that $w_1 \equiv_\ell^d w_2$. □

The implication (1) ⇒ (5) of Theorem 6.12 is a direct consequence of this proposition. Let $\big((p_1, q_1), (p_2, q_2)\big) \in (I_1 \times F_1) \times (I_2 \times F_2)$ be such that this pair has a common d-pattern. By Proposition 6.14, for all $\ell \in \mathbb{N}$, there exist $w_1 \in L(\mathcal{A}, \{p_1\}, \{q_1\})$, $w_2 \in L(\mathcal{A}, \{p_2\}, \{q_2\})$, such that $w_1 \equiv_\ell^d w_2$. Since, for $i = 1, 2$, $L(\mathcal{A}, \{p_i\}, \{q_i\}) \subseteq L(\mathcal{A}, I_i, F_i)$, it follows that these languages are not $\mathsf{LTT}[\ell, d]$-separable, for any ℓ.

6.2.4 From a common d-pattern in M to a common d-pattern in $\mathcal{A}$

This section is devoted to the proof of the implication (5) ⇒ (4) of Theorem 6.12. We will prove this by contraposition. That is, we prove that if there is a pair of monoid elements $(s_1, s_2) \in \alpha(L_1) \times \alpha(L_2)$ that has a common d-pattern, then there is also a pair of pairs of states $\big((p_1, q_1), (p_2, q_2)\big) \in (I_1 \times F_1) \times (I_2 \times F_2)$ that has a common d-pattern. This will follow from the following proposition, which states that the presence of a pair $(s_1, s_2) \in \alpha(L_1) \times \alpha(L_2)$ that has a common d-pattern does not depend on the choice of the recognizing monoid or on the choice of the recognizing monoid morphism.

Proposition 6.15. *Let $d \in \mathbb{N}$. Let $\alpha : A^* \to M$ and $\beta : A^* \to N$ be monoid morphisms recognizing both L_1 and L_2. If there exists $(s_1, s_2) \in \alpha(L_1) \times \alpha(L_2)$ that has a common d-pattern, then there exists $(t_1, t_2) \in \beta(L_1) \times \beta(L_2)$ that also has a common d-pattern.*

Proof. Let $\alpha : A^* \to M$ and $\beta : A^* \to N$ be morphisms recognizing both L_i, for $i = 1, 2$, and let $F_i = \alpha(L_i)$ and $G_i = \beta(L_i)$. Let $\mathcal{P}$ be a common d-pattern of $(s_1, s_2) \in F_1 \times F_2$. If $\mathcal{P} = w \in A^*$, then, by definition, $w \in \alpha^{-1}(F_1) \cap \alpha^{-1}(F_2) = L_1 \cap L_2 = \alpha^{-1}(G_1) \cap \alpha^{-1}(G_2)$, so $\mathcal{P}$ is a common d-pattern of $(\beta(w), \beta(w)) \in G_1 \times G_2$.

Otherwise, $\mathcal{P}$ is of the form $(\mathfrak{p}, f, \mathfrak{s})$. There exist $w_1, w_2 \in A^*$ that admit (α, s_1)- resp. (α, s_2)-compatible $\mathcal{P}$-decompositions. Let these $\mathcal{P}$-decompositions be the following.

$$\begin{array}{rcl} w_1 & = & u_0 v_1 u_1 \cdots v_n u_n, \\ w_2 & = & x_0 y_1 x_1 \cdots y_m x_m. \end{array} \tag{6.2}$$

By construction, $\mathfrak{p} = (u_0, v_1) = (x_0, y_1)$ and $\mathfrak{s} = (v_n, u_n) = (y_m, x_m)$.

From $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$, we define a new d-pattern $\mathcal{P}'$ that will be common to some $(t_1, t_2) \in G_1 \times G_2$. Define $\omega = |N|!$. Then, for all $s \in N$, s^ω is idempotent. For a block $\mathfrak{b} = (v_\ell, u, v_r)$, we write $\mathfrak{b}^\omega$ for $(v_\ell^\omega, u, v_r^\omega)$. Note that in contrast to the ω-power of a monoid element, the ω-power of a word is just a number of repetitions of the word and does not get reduced. Thus one can always retrieve the original word. It follows that the mapping $\mathfrak{b} \mapsto \mathfrak{b}^\omega$ is injective.

Let $\mathcal{P}' = (\mathfrak{p}', f, \mathfrak{s}')$ be the d-pattern defined as follows.

- $\mathfrak{p}' = (u_0, v_1^\omega)$ and $\mathfrak{s}' = (v_n^\omega, u_n)$,
- For all blocks $\mathfrak{b}$, if there exists $\mathfrak{c}$ such that $\mathfrak{c}^\omega = \mathfrak{b}$, then $f'(\mathfrak{b}) = f(\mathfrak{c})$, and else, $f'(\mathfrak{b}) = 0$.

Note that f' is well defined, since $\mathfrak{b} \mapsto \mathfrak{b}^\omega$ is an injective mapping.

Now, consider the words

$$\begin{array}{rcl} z_1 & = & u_0 v_1^\omega u_1 \cdots v_n^\omega u_n, \\ z_2 & = & x_0 y_1^\omega x_1 \cdots y_m^\omega x_m. \end{array} \tag{6.3}$$

Define $t_1 = \beta(z_1)$ and $t_2 = \beta(z_2)$. Since the decompositions of w_1 resp. w_2 in (6.2) were (α, s_1)- resp. (α, s_2)-compatible, duplicating v_i's resp. y_j's in these decompositions does not change the image under α. It follows that $z_1 \in L_1$, thus $\beta(z_1) \in G_1$, and similarly, $\beta(z_2) \in G_2$.

We will prove that the decompositions given in (6.3) are (β, t_1)- resp. (β, t_2)-compatible $\mathcal{P}'$-decompositions. Clearly, the conditions on $\mathfrak{p}'$ and $\mathfrak{s}'$ are satisfied. To see that $f'(\mathfrak{b})$ indeed counts the number of occurrences of $\mathfrak{b}$, up to threshold d, in z_i (for $i = 1, 2$), we use the following. By construction, each of the $f(\mathfrak{c})$ occurrences in the $\mathcal{P}$-decomposition of w_i gives rise to a specific occurrence of $\mathfrak{b}$ in z_i, and each occurrence of $\mathfrak{b}$ in z_i must come from a factor $\mathfrak{c}$ occurring in w_i. It follows that the decompositions from (6.3) are indeed $\mathcal{P}'$-decompositions. It remains to show that these decompositions are (β, t_i)-compatible. We show this for $i = 1$. For all $l \in \{1, \dots, n\}$, since $\omega = |N|!$,

$$\beta(u_0 \cdots v_l^\omega) \cdot \beta(v_l^\omega) = \beta(u_0 \cdots u_{l-1})\beta(v_l)^\omega \cdot \beta(v_l)^\omega = \beta(u_0 \cdots u_{l-1})\beta(v_l)^\omega = \beta(u_0 \cdots v_l^\omega).$$

Thus, we have shown that $(t_1, t_2) \in G_1 \times G_2$ has a common d-pattern. □

We now show how the implication (5) ⇒ (4) of Theorem 6.12 follows from Proposition 6.15. Assume that $(s_1, s_2) \in \alpha(L_1) \times \alpha(L_2)$ has a common d-pattern. Let N be the transition monoid of $\mathcal{A}$, and let $\beta : A^* \to N$ be the associated morphism. Since β recognizes L_1 and L_2, it follows from Proposition 6.15 that there exists $(t_1, t_2) \in \beta(L_1) \times \beta(L_2)$ with a common d-pattern. By definition of a transition monoid, it is then immediate to build, from (t_1, t_2), a pair $\big((p_1, q_1), (p_2, q_2)\big) \in (I_1 \times F_1) \times (I_2 \times F_2)$ that has a common d-pattern.

6.2.5 Bounding the profile size

This section deals with implication (4) ⇒ (3) of Theorem 6.12. We will prove the following statement, which is the contraposition of this implication: if, for $k = 4(|M|+1)$, the language $[L_1]_k^d$ does not separate L_1 from L_2, then there exists a pair $(s_1, s_2) \in \alpha(L_1) \times \alpha(L_2)$ that has a common d-pattern. As this is the first time in our proofs of the implications of Theorem 6.12 that the bound k on the size of the profiles comes into play, this implication forms an important part of the theorem, and it is not surprising that this implication is the most involved one to prove.

Our approach is the following. The fact that $[L_1]_k^d$ does not separate L_1 from L_2 implies that there exist words $w_1 \in L_1$ and $w_2 \in L_2$ such that $w_1 \equiv_k^d w_2$. From these two words, we will show how to construct a d-pattern $\mathcal{P}$ and two words $w_1' \in L_1$, $w_2' \in L_2$ that admit $\mathcal{P}$-decompositions that are (α, s_1)- resp. (α, s_2)-compatible, for some $s_1, s_2 \in M$. The construction of w_1', w_2' amounts to duplicating certain infixes in w_1, w_2 that verify special properties. We first define these special infixes, called *k-loops.* Throughout this section, we use the value $4(|M|+1)$ for k.

Definition 6.16. Let $w \in A^*$, let x be a position in w, and let (w_ℓ, w_r) be the $k/2$-profile of x. We say that x *admits a k-loop for α* if there exists a nonempty prefix u of w_r such that $\alpha(w_\ell) = \alpha(w_\ell \cdot u)$. In this case, the smallest such u is called *the k-loop of x for α.* See Figure 6.3. Usually, the morphism α is clear from the context and we do not mention it explicitly.

For our construction to work, three specific properties of k-loops are important. The first two properties are immediate from the definition: k-loops are determined by profiles, and k-loops can be duplicated without modifying the image of the word under α.

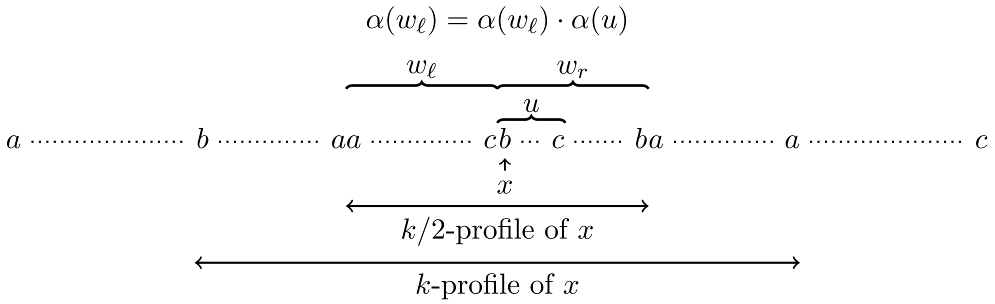

$\alpha(w_\ell) = \alpha(w_\ell) \cdot \alpha(u)$
w_ℓ
w_r
u
a b aa $cb \cdots c$ ba a c
x
$k/2$-profile of x
k-profile of x

Figure 6.3: A position x admitting a k-loop u, that is, $\alpha(w_\ell) = \alpha(w_\ell \cdot u)$.

Fact 6.17. Let x be a position of a word. Whether x admits a k-loop, and if so, which k-loops x admits, only depends on the $k/2$-profile of x. In particular, *the* k-loop of two positions with the same $k/2$-profile is the same.

Fact 6.18. Let w be a word and let x be a position of w that admits a k-loop u. Then, we have $\alpha(w[0,x]) = \alpha(w[0,x]) \cdot \alpha(u)$.

The last property we need is that k-loops occur frequently enough in words, which means for us that for $k = 4(|M|+1)$, at least one of $|M|+1$ consecutive positions must admit a k-loop. We will show this in the following lemma, using pumping arguments.

Lemma 6.19. *Let $w \in A^*$, let $\alpha : A^* \to M$ be a morphism, and let $k = 4(|M|+1)$. Let $x_1, \ldots, x_{|M|+1}$ be $|M|+1$ consecutive positions in w. Then, there exists at least one position x_i, with $1 \leq i < |M|+1$, that admits a k-loop for α.*

Proof. Consider the sequence $\alpha(w[x_1,x_1]), \alpha(w[x_1,x_2]), \ldots, \alpha(w[x_1,x_{|M|+1}])$, which consists of $|M|+1$ elements of M. By the pigeonhole principle, there are $i,j \in \mathbb{N}$ with $1 \leq i < j \leq x_{|M|+1}$, such that $\alpha(w[x_1,x_i]) = \alpha(w[x_1,x_j])$. We will prove that x_i admits a k-loop. The $k/2$-profile of x_i is (w_ℓ, w_r) with $w_\ell = w[\max(0, x_i - (|M|+1)), x_i]$ and $w_r = w[x_i, \min(x_i + |M|+1, |w|)$. Since $|w[x_1,x_i]| < |M|+1$, $w[x_1,x_i]$ is a suffix of w_ℓ. Write v for the word in A^* such that $w_\ell = v \cdot w[x_1,x_i]$. Let $u = w[x_i,x_j]$. Then, $\alpha(w_\ell) = \alpha(v) \cdot \alpha(w[x_1,x_i]) = \alpha(v) \cdot \alpha(w[x_1,x_j]) = \alpha(w_\ell \cdot u)$. Also, since $|u| < |M|+1$, u is a prefix of w_r. Therefore x_i admits u as a k-loop.

Note that u is not necessarily *the* k-loop of x_i, as there might be a smaller word that satisfies the definition as well. □

The construction of w'_1 and w'_2 will make use of the following notions.

Definition 6.20. Let $w, u \in A^*$, and let x be a position of w. The *word constructed from w by inserting u at position x* is the word $w[0,x] \cdot u \cdot w[x,|w|]$.

For example, inserting *cab* at position 3 of the word *abdabd* gives the word *abdcababd*.

It follows from Fact 6.18 that inserting the k-loop of a position at that position does not change the image under α of the word. Because of this fact, the following definition will play a role when showing that we can construct words, in the respective languages, that are

$\equiv_k^d$-equivalent.

Definition 6.21. Let $w \in A^*$. Let x be a position of w that admits a k-loop, and let z_x be the k-loop of x. Let w' be the word constructed from w by simultaneously inserting in w, for all such positions x, the infixes z_x. The word w' is called the *k-unfolding of w*.

We will prove the implication (4) ⇒ (3) of Theorem 6.12 by contraposition. Recall that by definition of $[L_1]_k^d$, we have that if this language does not separate L_1 from L_2, then there exist $w_1 \in L_1$, $w_2 \in L_2$ with $w_1 \equiv_k^d w_2$. Define $s_1 = \alpha(w_1)$ and $s_2 = \alpha(w_2)$. The following proposition now shows that the pair (s_1, s_2) then, indeed, has a common d-pattern.

Proposition 6.22. *Let $\alpha : A^* \to M$ be a morphism, let $k = 4(|M| + 1)$ and let $d \in \mathbb{N}$. Let w_1, w_2 be words such that $w_1 \equiv_k^d w_2$. Then, there exists a d-pattern $\mathcal{P}$, a word with an $(\alpha, \alpha(w_1))$-compatible $\mathcal{P}$-decomposition, and a word with an $(\alpha, \alpha(w_2))$-compatible $\mathcal{P}$-decomposition.*

Proof. If $w_1 = w_2$, one can use the d-pattern $\mathcal{P} = w_1$. The words w_1 and w_2 then give the desired $(\alpha, \alpha(w_1))$- resp. $(\alpha, \alpha(w_2))$-compatible $\mathcal{P}$-decompositions.

We can thus assume that $w_1 \neq w_2$. We will construct two new words w_1', w_2' from w_1, w_2 and prove that there exists a d-pattern $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$, for which w_1' admits an $(\alpha, \alpha(w_1))$-compatible $\mathcal{P}$-decomposition and w_2' admits an $(\alpha, \alpha(w_2))$-compatible $\mathcal{P}$-decomposition. To this end, for $i = 1, 2$, define w_i' as the k-unfolding of w_i. Since $w_1 \neq w_2$ and $w_1 \equiv_k^d w_2$, it follows in particular that $|w_1|, |w_2| > k/4 = |M| + 1$. By Lemma 6.19, at least one insertion has thus occurred both in the construction of w_1', and in the construction of w_2'.

Let us now define the d-pattern that we will use. By construction, the word w_1' can be decomposed as $w_1' = u_0 v_1 u_1 v_2 \cdots v_n u_n$, where $w_1 = u_0 u_1 \cdots u_n$ and the words v_j are the k-loops inserted during the construction. Since at least one insertion was made, it holds that $n \geq 1$ and we can define $\mathfrak{p} = (u_0, v_1)$, $\mathfrak{s} = (v_n, u_n)$. We define f as the function that maps a block (v_ℓ, u, v_r) to the number of times that it occurs in the decomposition of w_1', up to threshold d. That is, f sends the block (v_ℓ, u, v_r) to $\left|\{1 \leq i < n \mid (v_i, u_i, v_{i+1}) = (v_\ell, u, v_r)\}\right|$, counted with threshold d. Define $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$. By definition, $u_0 v_1 u_1 v_2 \cdots v_n u_n$ is a $\mathcal{P}$-decomposition for w_1'. Also, by Fact 6.18, it is $(\alpha, \alpha(w_1))$-compatible. It remains to prove that, for $\mathcal{P}$ as just defined, w_2' admits an $(\alpha, \alpha(w_2))$-compatible $\mathcal{P}$-decomposition.

By construction, the word w_2' can be decomposed in a similar way as w_1', that is, $w_2' = u_0' v_1' u_1' v_2' \cdots v_m' u_m'$, where $w_2 = u_0' u_1' \cdots u_m'$ and the words v_j' are the k-loops inserted during the construction of w_2'. If we prove that this is a $\mathcal{P}$-decomposition, then by Fact 6.18, it will be $(\alpha, \alpha(w_2))$-compatible.

Let us first see that, indeed, $(u_0', v_1') = \mathfrak{p}$ and $(v_m', u_m') = \mathfrak{s}$. We will use the fact that, since $w_1 \equiv_k^d w_2$, the words w_1 and w_2 have the same prefix of length $k - 1$, and the same suffix of length $k - 1$. Recall that the positions of w_1 are numbered from 0 to $|w_1| - 1$. Let x be the first position of w_1 that admits a k-loop. By construction, the k-loop of x is v_1. By Lemma 6.19, $x < k/4$. The $k/2$-profile of x is

$$\big(w_1[\max(0, x - k/4), x], w_1[x, \min(x + k/4, |w_1|)]\big).$$

Note that $x + k/4 < k/2$. Thus, $x + k/4 \leq k/2 - 1 = 2|M| + 1 \leq 4|M| + 2 = k - 2$, which is the last position of the prefix $w[0, k-1]$. It follows that the $k/2$-profile of x lies within the prefix of length $k - 1$. Thus, the corresponding position x' in the prefix of w'_2 has the same $k/2$-profile as x, and by Fact 6.17, it follows that the k-loop of x' is v_1. Note that, automatically, x' is the first position in w_2 that admits a k-loop. Otherwise, by symmetry, the same argument would yield an earlier position in w_1 admitting a k-loop, which would yield a contradiciton. It follows that $(u'_0, v'_1) = (u_0, v_1) = \mathfrak{p}$. In a similar way, we obtain that $(v'_m, u'_m) = \mathfrak{s}$.

Now let x be a position of w_1, that is inside some u_i, for $i \notin \{0, n\}$. Let x_ℓ and x_r be positions that admit a k-loop and that are the closest such positions to x, to the left (for x_ℓ) resp. to the right (for x_r). By construction, the k-loop of x_ℓ is v_i, and the k-loop of x_r is v_{i+1}. By Lemma 6.19, it follows that $x - x_\ell < k/4$ and $x_r - x \leq k/4$. The k-profile of x is

$$\big(w_1[\max(0, x - k/2), x], w_1[x, \min(x + k/2, |w_1|)]\big).$$

Since $x - x_\ell < k/4$, we have $x - k/2 < x_\ell - k/4$, and it follows that the $k/2$-profile of x_ℓ, that is,

$$\big(w_1[\max(0, x_\ell - k/4), x_\ell], w_1[x_\ell, \min(x_\ell + k/4, |w_1|)]\big),$$

is contained in the k-profile of x. Similarly, from $x_r - x \leq k/4$, we obtain that $x_r + k/4 \leq x + k/2$ and it follows that the $k/2$-profile of x_r is contained in the k-profile of x. For a sketch of the situation, see Figure 6.4.

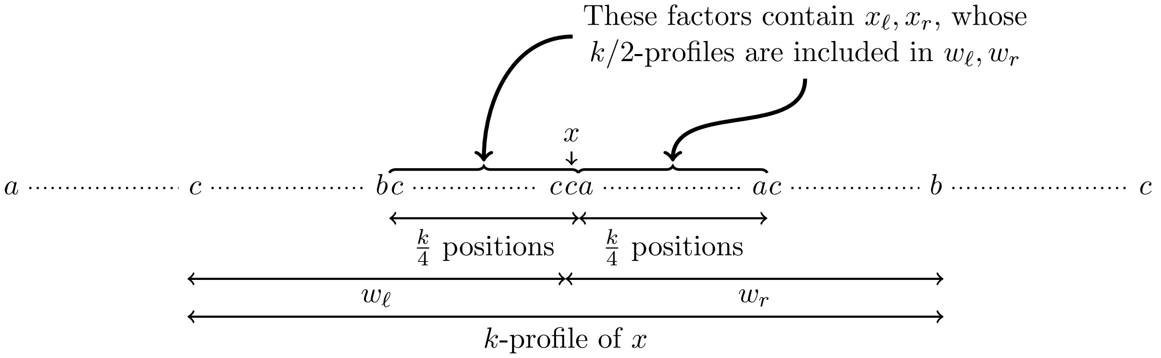


Figure 6.4: Construction in Proposition 6.22.

Let y be a position, either of w_1 or of w_2, that has the same k-profile as x. Say that y is in w_2, in some factor u'_j. Let x'_ℓ, x'_r be the corresponding positions relative to y (that is, the positions that are at the same distance to y as x_ℓ, x_r are to x). By the above, the $k/2$-profiles of x'_ℓ and x'_r are inside the k-profile of y, and are the same as the $k/2$-profiles of x_ℓ resp. x_r. Thus, using Fact 6.17, x'_ℓ, x'_r have k-loops v_i resp. v_{i+1}. As before, it follows that there are no positions closer to y that admit a k-loop. It follows that $(v_i, u_i, v_{i+1}) = (v'_j, u'_j, v'_{j+1})$. Note that this also says that the k-profile of a position x determines the positions of x_ℓ and x_r, relative to x, and thus determines the relative position of x inside u. This means that for each factor u_i and u'_j, all positions of the factor have different k-profiles.

Thus, when taking the k-unfolding of a word, only the k-profile of a position x of this word determines the block $\mathfrak{b} = (v_\ell, u, v_r)$ in which this x will end up inside u, in the factor $v_\ell \cdot u \cdot v_r$. For simplicity, we say in this case that position x ends up in $\mathfrak{b}$.

Consider a block $\mathfrak{b} = (v_\ell, u, v_r)$. Let $S_\mathfrak{b}$ be the set of all k-profiles, occurring in w_1 and w_2, of positions that end up in $\mathfrak{b}$. If u_i is a factor whose positions end up in $\mathfrak{b}$, all $|u_i| = |u|$ positions of this factor have a different k-profile. Thus, if $S_\mathfrak{b} \neq \varnothing$, then $|S_\mathfrak{b}| \geq |u|$. Since $w_1 \equiv_k^d w_2$, we have for every k-profile in $S_\mathfrak{b}$ that the number of positions that have this k-profile is equal, up to threshold d, in w_1 and w_2. Let b_i be the number of times that $\mathfrak{b}$ occurs in the decomposition of w'_i (so, $f(\mathfrak{b}) = b_1$, up to d). Note that if a position x in some factor u_j ends up in $\mathfrak{b}$, then all positions in u_j end up in $\mathfrak{b}$, and $u_j = u$. It follows that the number of positions in w_i that have a k-profile from $S_\mathfrak{b}$ is $b_i \cdot |u|$. By the above, $b_1 \cdot |u| = b_2 \cdot |u|$, up to d.

We want to show that $b_1 = b_2$, up to threshold d. If both are greater than or equal to d, this is true. Also, if $S_\mathfrak{b} = \varnothing$, then it is clear that $b_1 = b_2 = 0$. Therefore, suppose that $b_1 < d$ and that $|S_\mathfrak{b}| \geq |u|$. This means that the number of positions in w_1 that have a k-profile from $S_\mathfrak{b}$ is $b_1 \cdot |u| < d \cdot |u|$. By the pigeonhole principle, this means that there are at most $|u| - 1$ k-profiles that occur $\geq d$ times in w_1. All other profiles occur exactly the same number of times in w_2. Since each occurrence of $\mathfrak{b}$ in w'_2 gives rise to the occurrence of $|u|$ different k-profiles from $S_\mathfrak{b}$ in w_2, and since w_1 and w_2 differ in the occurrence of at most $|u| - 1$ k-profiles from $S_\mathfrak{b}$, it follows that each k-profile from $S_\mathfrak{b}$ occurs equally often in w_1 and in w_2. Thus, $b_1 < d$ implies that $b_1 = b_2$, and by symmetry, $b_2 < d$ also implies that $b_1 = b_2$. It follows that $b_1 = b_2$, up to threshold d.

In other words, $f(\mathfrak{b})$ is the number of times the block $\mathfrak{b}$ occurs in the decomposition of w'_2, and it follows that $u'_0 v'_1 u'_1 v'_2 \cdots v'_m u'_m$ is a $\mathcal{P}$-decomposition for w'_2. □

6.2.6 Decidability of separation by locally testable languages

Let us now focus on the class of locally testable languages. As mentioned before, decidability of the separation problem for locally testable languages follows immediately from Theorem 6.12 (one simply instantiates the theorem for $d = 1$). In view of the relevance of this class, we explicitly state the conditions equivalent to being LT-separable in the following theorem.

Theorem 6.23. *Let L_1, L_2 be regular languages. Let $\alpha : A^* \to M$ be a morphism into a finite monoid M recognizing both L_1 and L_2. Let $\mathcal{A}$ be an NFA recognizing both L_1 and L_2, with $L_i = L(\mathcal{A}, I_i, F_i)$. Let $k = 4(|M| + 1)$. Then, the following conditions are equivalent.*

(1) L_1 and L_2 are LT-separable,

(2) L_1 and L_2 are $\mathsf{LT}[k]$-separable,

(3) The language $[L_1]_k$ separates L_1 from L_2,

(4) There is no pair in $\alpha(L_1) \times \alpha(L_2)$ with a common 1-pattern,

(5) There is no pair in $(I_1 \times F_1) \times (I_2 \times F_2)$ with a common 1-pattern.

As before, this theorem yields an algorithm to decide LT-separability, by testing all the finitely many $\mathsf{LT}[k]$-languages, for $k = 4(|M| + 1)$, as potential separators. Again, this brute-force approach yields a very costly procedure. In Section 6.4.1, we will see that Conditions (4) and (5) can be exploited to obtain a more practical algorithm.

Corollary 6.24. *It is decidable whether two given regular languages are* LT*-separable.*

6.3 Separation for full locally threshold testable languages

In Section 6.3.1, we focus on showing that the separation problem for locally threshold testable languages is decidable. We will show that the bound on k, found in Theorem 6.12 when looking at LTT restricted to a fixed d, also works for the class of full LTT. This enables us to adapt an algorithm from [Boj07] and show that the separation problem for LTT is decidable. This result, however, does not give any insight about an actual separator yet. Next, in Section 6.3.2, we provide a bound on the counting threshold d. This result, together with the bound on k that we already had, yields another (brute-force) algorithm to test LTT-separability, and, more importantly, it yields a description of a separator in case it exists. Finally, in Section 6.3.3, we discuss the optimality of our bound on the counting threshold.

6.3.1 Decidability of separation by locally threshold testable languages

From Theorem 6.12, it follows in particular that for two regular languages L_1, L_2, both recognized by some monoid M, for $d \in \mathbb{N}$ and for $k = 4(|M|+1)$, the following equivalence holds.

$$\exists \ell \in \mathbb{N}.\ \ L_1, L_2 \text{ are } \mathsf{LTT}[\ell, d]\text{-separable} \quad\Leftrightarrow\quad L_1, L_2 \text{ are } \mathsf{LTT}[k, d]\text{-separable.}$$

Since two regular languages are LTT-separable if and only if there exist $d, \ell \in \mathbb{N}$ such that the languages are $\mathsf{LTT}[\ell, d]$-separable, the above equivalence yields, for $k = 4(|M|+1)$,

$$L_1, L_2 \text{ are } \mathsf{LTT}\text{-separable} \quad\Leftrightarrow\quad \exists d \in \mathbb{N}.\ \ L_1, L_2 \text{ are } \mathsf{LTT}[k, d]\text{-separable.} \tag{6.4}$$

Our proof for the decidability builds on the above formulation of LTT-separability, and on an adaptation of an algorithm to decide membership for LTT, presented in [Boj07]. In this paper, it was proved that for a fixed k, one can use Parikh's theorem [Par66] to translate the property of a language of being in a class $\mathsf{LTT}[k, d]$, for some d, to a computable Presburger formula. Presburger arithmetic is decidable [Pre29, Sko31]. In order to use this fact to prove that membership in LTT is decidable, two ingredients are used in [Boj07],

- a bound on k,
- the translation to Presburger arithmetic.

By the formulation of LTT-separability in (6.4), we already have a bound for k that works in our setting. It turns out that the translation to Presburger arithmetic, as applied in the paper, can easily be adapted to express separability rather than membership in a Presburger formula. The reason why this easy adaptation is possible is that the formula testing *membership* was actually already formulated as testing *separability* between the input language and its complement. In the rest of this section, we will generalize the arguments from [Boj07] and explain how to construct the suitable Presburger formula. The notion of commutative image, also called *Parikh image*, is important in this context. Let us first recall this notion.

Definition 6.25. Let $A = \{a_1, a_2, \ldots, a_n\}$ be an alphabet, ordered as $a_1 < a_2 < \ldots < a_n$. Let $w \in A^*$. The *commutative image* of w, denoted by $\pi(w)$, is a vector $(m_1, m_2, \ldots, m_n)$ of natural numbers, such that for all i, m_i is the number of occurrences of a_i in w. For a language L, the set $\pi(L) = \{\pi(w) \mid w \in L\}$ is called the commutative image of L.

For our purposes, it will be useful to count profiles occurring in a word, rather than just letters. Recall that the set of all k-profiles is denoted by A_k. We assume this set is ordered, for example, by an order induced by the order on A.

Definition 6.26. Let $k \in \mathbb{N}$. The *k-image* of w, denoted by $\pi_k(w)$, is the A_k-indexed vector of natural numbers that counts, for every k-profile (w_ℓ, w_r), the number of positions in w with this k-profile. If L is a language, we define the *k-image* of L as the set $\{\pi_k(w) \mid w \in L\}$, which we will denote by $\pi_k(L)$.

By definition of the relation $\equiv_k^d$, we have the following fact.

Fact 6.27. Let $w, w' \in A^*$ and let $k, d \in \mathbb{N}$. Then $w \equiv_k^d w'$ if and only if $\pi_k(w)$ and $\pi_k(w')$ are componentwise equal up to threshold d.

A well-known result about commutative images is Parikh's theorem, stated in [Par66], which says that if L is context-free (thus, in particular, if L is regular), then $\pi(L)$ is semilinear. Recall that a set of vectors is semilinear if it is a finite union of linear sets of vectors, and that a set of vectors is linear if it is of the form $\{\overline{x}_0 + i_1\overline{x}_1 + \ldots + i_m\overline{x}_m \mid i_1, \ldots, i_m \in \mathbb{N}, \overline{x}_0, \ldots, \overline{x}_m \in \mathbb{N}^n\}$. By [GS66, Theorem 1.3], the fact that $\pi(L)$ is semilinear implies that $\pi(L)$ is Presburger definable. As explained in [Boj07], Parikh's theorem extends without difficulty to k-images. Let us prove this in the following theorem.

Theorem 6.28. *Let L be a context-free language and let $k \in \mathbb{N}$. Then, $\pi_k(L)$ is semilinear. Moreover, a Presburger formula for this semilinear set can be computed from L.*

Proof. If $k = 1$, then $\pi_k(L) = \pi(L)$, and by Parikh's theorem, this set is semilinear and can be effectively calculated. When $k > 1$, consider the following language L' over the alphabet A_k of k-profiles: a word w' is in L' if and only if there exists $w \in L$ of the same length (measured in terms of their own respective alphabets) and such that a position in w' is labeled by the k-profile of the same position in w. The language L' is context-free and, by construction, the k-image $\pi_k(L)$ of L is the commutative image $\pi(L')$ of L'. By Parikh's theorem, $\pi(L')$ is semilinear and can be effectively calculated. It then follows from [GS66, Theorem 1.3] that one can compute a Presburger formula for the set $\pi_k(L)$. □

We can now explain how to decide LTT-separability.

Theorem 6.29. *It is decidable whether two regular languages are* LTT*-separable.*

Proof. Given two regular languages L_1 and L_2, and a monoid M recognizing both of these languages, we let $k = 4(|M| + 1)$. By the formulation in (6.4), L_1, L_2 are LTT-separable if and only if there exists $d \in \mathbb{N}$ such that they are LTT$[k, d]$-separable. This is the case if and only if there exists $d \in \mathbb{N}$ such that there are no words $w_1 \in L_1, w_2 \in L_2$ with $w_1 \equiv_k^d w_2$.

By Fact 6.27, this can be expressed in terms of k-images in the following way: there exists $d \in \mathbb{N}$ such that there do not exist any vectors $\overline{x}_1 \in \pi_k(L_1), \overline{x}_2 \in \pi_k(L_2)$ that are equal, componentwise, up to threshold d.

By Theorem 6.28, there are computable Presburger formulas for the sets $\pi_k(L_1)$ and $\pi_k(L_2)$. It follows that the above statement can be expressed as a computable Presburger formula. From the decidability of Presburger arithmetic [Pre29, Sko31], it then follows that LTT-separability is decidable. □

Besides the decidability of the LTT-separation problem, we are also interested in finding an efficient algorithm that tests LTT-separability, and in finding an LTT-separator if it exists. Finding an LTT-separator is the subject of the next section.

6.3.2 Bounding the counting threshold

In this section, we focus on finding a bound on the counting threshold d, which is such that two languages, recognized by a monoid M, are LTT-separable if and only if they are LTT$[k, d]$-separable, for $k = 4(|M| + 1)$ and for the bound on d. Finding such a bound would yield another proof of the fact that the separation problem for LTT is decidable. An important advantage, however, of this approach, rather than the one taken in Section 6.3.1, is that it also yields a description of a separator, if it exists.

Recall that A_k denotes the set of all k-profiles. Let n be either $|M| + 1$ or $|Q| + 1$, for M a monoid and $\mathcal{A} = (A, Q, \delta)$ an automaton that recognize both languages. We claim that $d = (|A_k|n)^{|A_k|}$ has the desired property. It will follow from Theorem 6.32, which is a separation theorem for the full class of LTT languages, that the proposed bound on d is indeed correct.

We first look at the following lemma, that shows that whenever k-profiles occur 'many' times in a word w of a language L, one can obtain, by pumping appropriate factors of w, an equivalent word that is still in L and in which these k-profiles occur as often as one wants. This result will be useful later on, when we want to use witnesses of non-LTT$[k, d]$-separability (for d fixed as above), to construct witnesses of non-LTT$[k, d']$-separability, for every d'.

Lemma 6.30. *Let L be a language and let $\alpha : A^* \to M$ be a morphism into a finite monoid M recognizing L. Let $k = 4(|M| + 1)$ and let $n = |M| + 1$. Let $h, h' \in \mathbb{N}$, such that $h \geq 1$ and $h' \geq n \cdot |A_k| \cdot h$. Let $w \in L$. Then, for all $c \in \mathbb{N}$, one can construct a word $w' \in L$ such that $w' \equiv_k^h w$ and such that every k-profile that occurs h' or more times in w, occurs c or more times in w'.*

Proof. A naive approach would be the following. As soon as a k-profile occurs at more than $n = |M| + 1$ positions in w, the same monoid element occurs twice in the sequence of the images of the prefix of w up to these positions. Therefore, one can pump the corresponding infix of w that occurs between these positions to generate c copies of the k-profile, without affecting membership in L. However, we also want to maintain that $w' \equiv_k^h w$. Thus, one needs to be more careful and avoid duplicating k-profiles that occur strictly less than h times in w. This is why we use the much higher constant h', rather than n, to find the infixes that can be pumped in order to construct w'.

If there is no k-profile that occurs more than h' times in w, then it suffices to take $w' = w$. Else, let (w_ℓ, w_r) be such a k-profile that occurs more than h' times in w. We will explain how, by pumping factors in w, one can obtain a word w' that contains more than c copies of (w_ℓ, w_r), while at the same time $w' \equiv_k^h w$. This construction can then be repeated to treat all k-profiles that occur more than h' times in w, in order to get the desired w'.

Let $x_1 < \ldots < x_{h'}$ be h' positions where (w_ℓ, w_r) occurs. Note that there are at most $|A_k|(h-1)$ positions in w such that the k-profile at this position occurs strictly less than h times in w. We look at the positions that are in between consecutive positions from the list $x_1, \ldots, x_{h'}$. By choice of h', there are at least $n|A_k|h - 1$ such regions of positions that are between x_i and x_{i+1} for some $1 \leq i < h'$. Since there are at most $|A_k|(h-1)$ positions in w that have a k-profile which occurs strictly less than h times in w, it follows that there exist at least n consecutive positions in the list, say $x_i, \ldots, x_{i+(n-1)}$, such that no intermediate position between x_i and $x_{i+(n-1)}$ has a k-profile that occurs less than h times in w. (For, if there would not be such n positions, we would need one position with a k-profile that occurs less than h times for each sequence of n consecutive regions. Since there are at least $n|A_k|h-1$ regions, this means we would need at least $|A_k|h-1$ positions in w that have a k-profile which occurs strictly less than h times in w. But as $|A_k| > 1$, we have that $|A_k|h - 1 > |A_k|(h-1)$, giving a contradiction).

Now look at the following list of n monoid elements,

$$\alpha(w[0, x_i]),\ \ \alpha(w[0, x_{i+1}]),\ \ \ldots,\ \ \alpha(w[0, x_{i+(n-1)}]).$$

Since $n = |M| + 1$, there are positions x_p, x_q from the list, such that $x_i \leq x_p < x_q \leq x_{i+(n-1)}$ and $\alpha(w[0, x_p]) = \alpha(w[0, x_q])$. This means that the infix $w[x_p, x_q]$ can be repeated to generate c copies of (w_ℓ, w_r), without affecting membership in L. Furthermore, since none of the positions in between $x_i, \ldots, x_{i+(n-1)}$ had a k-profile occurring less than h times in w, the pumping did not duplicate any such k-profile. Therefore, the resulting word w' also has the desired property $w' \equiv_k^h w$. □

Note that in the lemma above, one could also have taken $n = |Q| + 1$, for $\mathcal{A} = (A, Q, \delta)$, instead of $n = |M| + 1$. The same reasoning would then yield that two positions among $x_i, \ldots, x_{i+(n-1)}$ would visit the same state in the run of w in $\mathcal{A}$. Pumping the corresponding infix would then give the same result.

The following proposition forms, together with equivalence (6.4), the key ingredient for our separation theorem for the class of full LTT. It states that one only needs to consider LTT languages with counting threshold $d = (|A_k|n)^{|A_k|}$ in order to check whether the input languages are LTT$[k, d']$-separable for some d'.

Proposition 6.31. *Let L_1, L_2 be regular languages. Let $\alpha : A^* \to M$ be a morphism into a finite monoid M recognizing both L_1 and L_2. Let $\mathcal{A} = (A, Q, \delta)$ be an NFA recognizing both L_1 and L_2. Let n be either $|M| + 1$ or $|Q| + 1$. Let $k = 4(|M| + 1)$ and let $d = (|A_k|n)^{|A_k|}$. If there exists $d' \in \mathbb{N}$ such that L_1 and L_2 are LTT$[k, d']$-separable, then the language $[L_1]_k^d$ separates L_1 from L_2.*

Proof. We will prove the contraposition of the statement, that is, if $[L_1]_k^d$ does not separate L_1 from L_2 for the values of k and d defined in the proposition, then, for all $d' \in \mathbb{N}$, L_1 and L_2 are not $\mathsf{LTT}[k, d']$-separable.

Assume that $[L_1]_k^d$ is not a separator. By definition, this means that there exist $w_1 \in L_1$ and $w_2 \in L_2$ such that $w_1 \equiv_k^d w_2$. Fix $d' \in \mathbb{N}$. Using Lemma 6.30, we will show that one can construct words $w_1' \in L_1, w_2' \in L_2$ such that $w_1' \equiv_k^{d'} w_2'$. By definition of $\equiv_k^{d'}$, this means that L_1 and L_2 are not $\mathsf{LTT}[k, d']$-separable.

Recall that $n = |M|+1$ or $n = |Q|+1$. For simplicity, we only treat the case that $n = |M|+1$. The other case, however, can be proved in the same way, since Lemma 6.30 works for both of these cases. Let $m = |A_k|n$, so that $d = m^{|A_k|}$. Let $\ell \in \mathbb{N}$. We denote the following statement by $\mathrm{P}(\ell)$.

$\mathrm{P}(\ell)$ For all $u_1 \in L_1, u_2 \in L_2$, and $d' \in \mathbb{N}$, if $u_1 \equiv_k^{m^\ell} u_2$ and the number of k-profiles that do *not* occur $\geq d'$ times in both u_1 and u_2 is smaller than ℓ,
then, there exist words $u_1' \in L_1$ and $u_2' \in L_2$ such that $u_1' \equiv_k^{d'} u_2'$.

We want to prove $\mathrm{P}(|A_k|)$. Let us first note that by definition of $d = m^{|A_k|}$, we have that $w_1 \equiv_k^{m^{|A_k|}} w_2$. Also, we have for all $d' \in \mathbb{N}$ that the number of k-profiles that do not occur $\geq d'$ times in both w_1 and w_2 is smaller than the number of *all* k-profiles, i.e. smaller than $|A_k|$. Therefore, $\mathrm{P}(|A_k|)$ entails that, for all d', there exist words w_1', w_2' such that the desired property $w_1' \equiv_k^{d'} w_2'$ holds. This exactly means that for all $d' \in \mathbb{N}$, the languages L_1 and L_2 are not $\mathsf{LTT}[k, d']$-separable. We will prove, by induction on ℓ, that $\mathrm{P}(\ell)$ holds for all $\ell \leq |A_k|$.

First, let $\ell = 0$. If u_1, u_2 and d' verify the premise in $\mathrm{P}(\ell)$, then, in particular, the number of k-profiles that do *not* occur more than d' times in both u_1 and u_2 is 0. Thus, all k-profiles in u_1, u_2 occur more than d' times in both words, and therefore $u_1 \equiv_k^{d'} u_2$.

Now, assume that $\ell > 0$. Let u_1, u_2 and d be such that the premise in $\mathrm{P}(\ell)$ is verified. If $u_1 \equiv_k^{d'} u_2$, then it suffices to take $u_1' = u_1$ and $u_2' = u_2$. Otherwise, since $u_1 \equiv_k^{m^\ell} u_2$, there must exist at least one k-profile (w_ℓ, w_r) that occurs more than m^ℓ times in both u_1 and u_2 but strictly less than d' times in at least one of the two words.

We apply Lemma 6.30 to both u_1 and u_2, for $h = m^{\ell-1}$ and $h' = m^\ell$. Note that, by definition of m, indeed, $h' \geq n|A_k|h$. The lemma yields that there are $u_1'' \in L_1$, $u_2'' \in L_2$ such that $u_1'' \equiv_k^{m^{\ell-1}} u_1$, $u_2'' \equiv_k^{m^{\ell-1}} u_2$, and, every k-profile that occurs more than m^ℓ times in u_1 resp. u_2, occurs more than d' times in u_1'' resp. u_2''. First note that, since also $u_1 \equiv_k^{m^\ell} u_2$, we obtain that $u_1'' \equiv_k^{m^{\ell-1}} u_2''$. Furthermore, (w_ℓ, w_r) now occurs more than d' times in both u_1'' and u_2''. Therefore, the number of k-profiles that do not occur more than d' times in both u_1'' and u_2'' is smaller than $\ell - 1$. Hence, we can apply the induction hypothesis to u_1'', u_2'' and d', and it follows that the desired u_1' and u_2' exist. □

We are now ready to prove our separation theorem for the class of locally threshold testable languages.

Theorem 6.32. *Let L_1, L_2 be regular languages. Let $\alpha : A^* \to M$ be a morphism into a finite monoid M recognizing both L_1 and L_2. Let $\mathcal{A} = (A, Q, \delta)$ be an NFA recognizing both*

L_1 and L_2, such that $L_i = L(\mathcal{A}, I_i, F_i)$. Let n be either $|M|+1$ or $|Q|+1$. Let $k = 4(|M|+1)$ and let $d = (|A_k|n)^{|A_k|}$. Then, the following conditions are equivalent.

(1) L_1 and L_2 are LTT-separable,

(2) There exists $d' \in \mathbb{N}$ such that L_1 and L_2 are $\mathsf{LTT}[k, d']$-separable,

(3) There exists $d' \in \mathbb{N}$ such that no pair in $\alpha(L_1) \times \alpha(L_2)$ has a common d'-pattern,

(4) There exists $d' \in \mathbb{N}$ such that no pair in $(I_1 \times F_1) \times (I_2 \times F_2)$ has a common d'-pattern,

(5) L_1 and L_2 are $\mathsf{LTT}[k, d]$-separable,

(6) The language $[L_1]_k^d$ separates L_1 from L_2.

Proof. The implications (6) $\Leftrightarrow$ (5) $\Rightarrow$ (2) $\Rightarrow$ (1) are immediate by definition. Also, we have already seen in (6.4) that Theorem 6.12 implies that (2) $\Leftrightarrow$ (1). Note that from (2), we obtain by Theorem 6.12 that for the same value for d', there is no pair in $\alpha(L_1) \times \alpha(L_2)$ with a common d'-pattern. Thus, Condition (2) implies Condition (3). In the same way, we obtain (3) $\Rightarrow$ (4) $\Rightarrow$ (2). Finally, we have proved implication (2) $\Rightarrow$ (6) in Proposition 6.31. □

Note that Condition (5) provides another proof of the decidability of LTT-separability: one uses a brute-force algorithm to test all the finitely many $\mathsf{LTT}[k, d]$-languages. As it was the case for a fixed counting threshold, this algorithm is slow and we will present a faster algorithm using Conditions (3) and (4) in Section 6.4.1.

Note also that, since Theorem 6.12 implies that the value of d of Condition (5) works as value of d' in Conditions (3) and (4), we have obtained the stabilization index of sequence (6.1). That is, the sequence of d-indistinguishable pairs $(\mathrm{I}_d(\alpha))_{d \in \mathbb{N}}$ stabilizes at $d = (|A_k|n)^{|A_k|}$.

6.3.3 Optimality of the bound on the counting threshold

The bound for the counting threshold in Theorem 6.32, i.e. $(|A_k|n)^{|A_k|}$ (for $n = |M|+1$ or $|Q|+1$), is exponential in the size of the set of all k-profiles, A_k. Since $k = 4(|M|+1)$, the size of A_k itself is exponential in the size of the monoid. This section is concerned with the question whether this bound on the counting threshold can be improved.

Note that our proof of Theorem 6.32 treats the bounding of k and d independently. We first provide a bound on k in Theorem 6.12. Then, instead of studying words over A, we look at the corresponding words over the alphabet A_k of k-profiles, in order to bound d. This technique ignores important properties of k-profiles. In particular, the k-profiles of adjacent positions are of course strongly related, and this fact is not exploited by our proof.

In this subsection, we show that if one would want to improve the bound on the counting threshold, this would require taking these additional properties into account. With this aim, we look at $k = 1$, since this means that the k-profile of a position is just the letter that it carries, and, contrary to the case for higher values of k, there are no relations between k-profiles of adjacent positions. We show that we can construct $\mathsf{LTT}[1, d]$-separable languages, for which the separator is required to have a counting threshold that is exponential in $|A|$.

Let us first provide an example, which we will generalize in Lemma 6.34. For readability, we write here $|a|_w$ to denote the number of times that the letter a occurs in the word w (deviating from the notation applied in the beginning of this chapter).

Example 6.33. Consider the following languages over the alphabet $\{a, b, c, d\}$.

$$\begin{array}{rcl} L_1 &=& a(bcc)^*(ddd)^*, \\ L_2 &=& (abb)^*(cdd)^*, \\ L &=& \{w \mid |a|_w < 2 \Rightarrow |b|_w = 2 \cdot |a|_w, \text{ and } |c|_w < 5 \Rightarrow |d|_w = 2 \cdot |c|_w\}. \end{array}$$

We will show that the language L separates L_2 from L_1. First of all, note that $L_2 \subseteq L$. Now, suppose that $w \in L_1 \cap L$. Using that (1). $w \in L_1$ and that (2). $w \in L$, we obtain

$$|a|_w = 1 \overset{(2)}{\Longrightarrow} |b|_w = 2 \cdot |a|_w = 2 \overset{(1)}{\Longrightarrow} |c|_w = 4 \overset{(2)}{\Longrightarrow} |d|_w = 2 \cdot |c|_w = 8.$$

But, by (1), $|d|_w$ is also a multiple of 3. It follows that $L_1 \cap L = \varnothing$. The language L is in $\mathsf{LTT}[1, 9]$. To see this, let $w \in L$, and let w' be a word such that $w' \equiv_1^9 w$. Either $|a|_{w'} \geq 2$, which is fine, or $|a|_{w'} < 2$. Then, $|a|_w = |a|_{w'} < 2$, such that $|b|_w = 2 \cdot |a|_w < 9$ and $|b|_{w'} = |b|_w = 2 \cdot |a|_{w'}$. In the same way, if $|c|_{w'} < 5$, then one obtains $|d|_{w'} = |d|_w = 2 \cdot |c|_{w'}$, since this is still smaller than the threshold 9. It follows that $w' \in L$. The languages L_1 and L_2 are not $\mathsf{LTT}[1, 8]$-separable (consider the words $w_1 = abc^2bc^2d^{24} \in L_1$ and $w_2 = ab^2cd^2cd^2cd^2cd^2 \in L_2$, for which $w_1 \equiv_1^8 w_2$ holds). Thus, the counting threshold of the $\mathsf{LTT}[1, 9]$-separator L is optimal.

Now, let us generalize this example. We want to construct, for every A, two languages that are separable by an LTT language with a counting threshold d that is exponential in $|A|$, and that are not separable by any LTT language with a counting threshold lower than d.

For convenience, we assume that the alphabet A is of even size and write $A = \{a_1, \ldots, a_{2m}\}$.

Lemma 6.34. *The languages*

$$\begin{array}{rcl} L_1 &=& a_1 \cdot (a_2a_3a_3)^*(a_4a_5a_5)^* \cdots (a_{2m-2}a_{2m-1}a_{2m-1})^* \cdot (a_{2m}a_{2m}a_{2m})^*, \\ L_2 &=& (a_1a_2a_2)^*(a_3a_4a_4)^* \cdots (a_{2m-1}a_{2m}a_{2m})^* \end{array}$$

are $\mathsf{LTT}[1, d]$*-separable for some* d*, but are not* $\mathsf{LTT}[1, 2^{2m-1}]$*-separable.*

Proof. We prove that L_1, L_2 are $\mathsf{LTT}[1, d]$-separable for $d = 2^{2m-1} + 1$. Consider the following language L.

$$L = \{w \mid \text{for all odd } i,\ |a_i| < 2^{i-1} + 1 \Rightarrow |a_{i+1}|_w = 2 \cdot |a_i|_w\}.$$

Thus, a word w belongs to L if and only if for all odd i, either w contains at least $2^{i-1} + 1$ copies of a_i, or the number of copies of a_{i+1} in w is exactly twice the number of copies of a_i in w.

To check whether a word w is a member of the language L, we will have to count a maximum number of occurrences of a letter in the case that $|a_{2m-1}|_w = 2^{2m-2}$. In this case, one needs to verify that $|a_{2m}|_w = 2 \cdot |a_{2m-1}|_w = 2^{2m-1}$. To check this equality, we need a counting threshold of $d = 2^{2m-1}+1$. With the power of this d, one is clearly also able to verify the conditions on the a_i's for smaller i. It follows that $L \in \mathsf{LTT}[1, 2^{2m-1}+1]$.

Let us now show that L is a separator. By definition, $L_2 \subseteq L$. Suppose that $w \in L \cap L_1$. Since $w \in L_1$, it contains only one copy of a_1. Then, since $w \in L$, it must contain two copies of a_2. Iterating this argument yields that w must contain 2^{2m-1} copies of a_{2m}, which is impossible since this number must be multiple of 3, by definition of L_1. Thus, $L \cap L_1 = \varnothing$, and it follows that L is a separator.

It remains to prove that L_1, L_2 are not $\mathsf{LTT}[1, 2^{2m-1}]$-separable. To this end, consider the words

$$\begin{array}{rcll} w_1 & = & a_1(a_2a_3^2)^2 \cdots (a_{2m-2}a_{2m-1}^2)^{2^{2m-3}}(a_{2m})^{3\cdot 2^{2m-1}} & \in L_1, \\ w_2 & = & (a_1a_2^2)(a_3a_4^2)^4 \cdots (a_{2m-1}a_{2m}^2)^{2^{2m-2}} & \in L_2. \end{array}$$

For every $i \in \{1, \dots, 2m-1\}$, we have $|a_i|_{w_1} = 2^{i-1} = |a_i|_{w_2}$. The letter a_{2m} occurs $3 \cdot 2^{2m-1}$ times in w_1, and 2^{2m-1} times in w_2. When counting with threshold 2^{2m-1}, these numbers are considered equal. Thus, $w_1 \equiv_1^{2^{2m-1}} w_2$. Therefore L_1, L_2 are not $\mathsf{LTT}[1, 2^{2m-1}]$-separable. $\square$

It follows from the previous lemma that without taking additional properties of k-profiles into account, one cannot find a bound on the counting threshold that is better than exponential in $|A|$.

6.4 Complexity of LT- and LTT-separability

In this section, we present lower and upper complexity bounds for the separation problem for LT and LTT languages. Both the lower and upper bounds use the pattern criteria of Theorems 6.23 and 6.32. We are able to prove that starting from an NFA or DFA recognizing the input languages, deciding separability can be achieved in co-Nexptime for LT and in 2-Expspace for LTT. This is shown in Section 6.4.1. In Section 6.4.2, we show how generalizing the reduction of Section 2.3.1 gives a co-Np lower bound for both problems.

6.4.1 Upper complexity bounds

We first look at complexity upper bounds for the separation problems for LT and LTT. The algorithms that we use rely on Condition (5) of Theorem 6.23 and on Condition (4) of Theorem 6.32. These conditions are the criteria about the absence of certain patterns in the automata. We will show that deciding whether two languages, accepted by some NFA, are LT-separable can be achieved in co-Nexptime, while deciding whether they are LTT-separable can be achieved in 2-Expspace.

Both algorithms work by reducing the problems to the special case of $k = 1$, that is, to the problem of verifying whether there exists an LT- resp. LTT-separator that considers only 1-profiles. The reduction is identical in both cases, and the proofs of the fact that the reduction is correct are similar. These proofs rely on Condition (5) in Theorem 6.23, for LT, and on Condition (4) in Theorem 6.32, for LTT. The computations involved in the reduction can be done in Exptime, and the newly constructed NFA is of size exponential in the input NFA.

The algorithms to decide LT- resp. LTT-separability for the special case of $k = 1$ are different for the two classes. We provide algorithms, for this special case of $k = 1$, which run in co-Np for LT and in Expspace for LTT.

Note that one could also consider the bound k on the size of k-profiles, presented in Theorems 6.23 and 6.32, in order to reduce the problem of LT- resp. LTT-separability to the case of $k = 1$. Indeed, once k is fixed, it suffices to modify the input NFA to work on the alphabet of k-profiles. This would also give a reduction to the case of $k = 1$. However, this technique might yield an NFA that is doubly exponential in the size of the input NFA.

We first present and prove the reduction to the case of $k = 1$. After this, we explain how to decide both problems in this special case, and we show which upper complexity bounds, for LT- resp. LTT-separability, this approach yields.

Reduction to the case of $k = 1$

Let $\mathcal{A} = (A, Q, \delta)$ be an NFA. It follows from Theorem 6.23 (resp. Theorem 6.32) that to determine whether $L(\mathcal{A}, I_1, F_1)$ and $L(\mathcal{A}, I_2, F_2)$ are *not* LT-separable (resp. *not* LTT-separable), it suffices to verify whether there exists a pair in $(I_1 \times F_1) \times (I_2 \times F_2)$ that has a common 1-pattern (resp. whether there exists $d \in \mathbb{N}$ such that there exists a pair in $(I_1 \times F_1) \times (I_2 \times F_2)$ that has a common d-pattern). This requires verifying whether there exist a pattern $\mathcal{P}$, a $\mathcal{P}$-decomposition that is compatible with a pair in $I_1 \times F_1$, and a $\mathcal{P}$-decomposition that is compatible with a pair in $I_2 \times F_2$.

A first step in our reduction to the case of $k = 1$ is to view $\mathcal{P}$-decompositions as words over the alphabet of blocks. This is illustrated in the following example.

Example 6.35. Let $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$ be the following 2-pattern. The prefix block $\mathfrak{p} = (a, b)$ and the suffix block $\mathfrak{s} = (bb, a)$. The function f sends the block (b, aa, bb) to 2, the block (bb, a, b) to 1 and all other blocks to 0. Let w be the word $abaabbabaabba$. Clearly, w is a $\mathcal{P}$-decomposition, as it can be decomposed as follows.

$$w \quad = \quad \underbrace{a}_{u_0}\underbrace{b}_{v_1}\underbrace{aa}_{u_1}\underbrace{bb}_{v_2}\underbrace{a}_{u_2}\underbrace{b}_{v_3}\underbrace{aa}_{u_3}\underbrace{bb}_{v_4}\underbrace{a}_{u_4}$$

Now, $\mathfrak{p} = (u_0, v_1)$, $\mathfrak{s} = (v_4, u_4)$ and $(v_1, u_1, v_2), (v_3, u_3, v_4) = (b, aa, bb)$ and $(v_2, u_3, v_3) = (bb, a, b)$. We now view w as the following word of blocks,

$$(a, b)(b, aa, bb)(bb, a, b)(b, aa, bb)(bb, a).$$

Note that, for such a correspondence between $\mathcal{P}$-decompositions and words over the alphabet of blocks to make sense, we should only consider words of blocks that satisfy the following compatibility condition: if (u, v') or (v, u, v') is followed by (v'', u', v''') or (v'', u'), then $v' = v''$.

The main idea behind the reduction is to construct a new NFA $\widetilde{\mathcal{A}}$, which recognizes words over the alphabet of blocks that represent special $\mathcal{P}$-decompositions. Namely, those $\mathcal{P}$-decompositions that are, for some $\mathcal{P}$, compatible with pairs of states of $I_1 \times F_1$ and of $I_2 \times F_2$

in $\mathcal{A}$. There are two issues with this idea. The first one we addressed above, that is, a word of blocks should satisfy the compatibility condition in order to correspond to a $\mathcal{P}$-decomposition. This compatibility property for consecutive blocks cannot be simply encoded in the states, since there are infinitely many words. The second issue is that the alphabet of blocks is infinite.

We take care of these issues using a similar argument for both. Let us first discuss the issue that the alphabet of blocks is infinite. Recall that we are only interested in $\mathcal{P}$-decompositions that are compatible with pairs of states of $I_1 \times F_1$ and of $I_2 \times F_2$ in $\mathcal{A}$. Observe that for a block (v, u, v') to appear in such a decomposition, there need to exist states q, q' in $\mathcal{A}$ with loops around these states labeled by v resp. v', and a path from q to q' labeled by u. Instead of dealing with a specific block, it thus suffices to deal with the set of pairs of states verifying this property. The number of such sets is bounded by $2^{|Q|^2}$, in particular, there are only finitely many such sets. Making the abstraction from blocks to these sets thus yields a finite alphabet.

The same argument can be used to deal with the compatibility issue. All words v that need to be considered for the compatibility condition need to label a loop around some state q. Instead of taking the specific word v into account, we will use the set of states having such a loop labeled by v. Again, this abstraction retains the relevant information from an infinite set (of words) and encodes it in a finite set (of subsets of states of $\mathcal{A}$). As we will show in the formal construction, this information can then be encoded in the states of $\widetilde{\mathcal{A}}$.

Before providing the formal construction of $\widetilde{\mathcal{A}}$ in Construction 6.38, we introduce some relevant notions. First, we define the notion used to encode the compatibility condition in the automaton.

Definition 6.36. Let $\mathcal{A} = (A, Q, \delta)$ be an NFA and let $R \subseteq Q$. We say that R is *synchronizable* if there exists a nonempty word $v \in A^*$ such that, for all $q \in R$, there exists a loop around q labeled by v.

The following definition introduces the ingredients of the new alphabet that will encode the relevant information about the blocks.

Definition 6.37. Let $\mathcal{A} = (A, Q, \delta)$ be an NFA and let $T \subseteq Q^2$. We denote the set of states that are left (resp. right) members of pairs in T by $\ell(T)$ (resp. by $r(T)$). We say that T is *synchronizable* if

(a) there exists a word $u \in A^*$ such that, for all $(q, q') \in T$, there exists a run from q to q' labeled by u,

(b) $\ell(T)$ is synchronizable, and

(c) $r(T)$ is synchronizable,

where the notion of synchronizable in (b) and (c) refers to Definition 6.36. In order to deal with prefixes and suffixes, we generalize the notion for T to these limit cases. We say that T is *prefix synchronizable* (resp. *suffix synchronizable*) if (a) and (c) (resp. (a) and (b)) hold. Finally, we say that T is *weakly synchronizable* if (a) holds.

We can now show how to construct the new NFA $\widetilde{\mathcal{A}}$ from $\mathcal{A} = (A, Q, \delta)$ and sets of states

I_1, F_1, I_2, F_2.

Construction 6.38. Let $\mathcal{A} = (A, Q, \delta)$ be an NFA and let $I_1, F_1, I_2, F_2 \subseteq Q$. Let B_w, B_p, B_i and B_s be the sets of weakly synchronizable, prefix synchronizable, synchronizable and suffix synchronizable sets of pairs of states, respectively. We let the alphabet B of the new automaton $\widetilde{\mathcal{A}}$ be the disjoint union of the sets B_w, B_p, B_i and B_s. The new set of states $\widetilde{Q}$ is defined as follows.

$$\widetilde{Q} = \{(r, R) \mid r \in R \subseteq Q,\ R \text{ is synchronizable}\} \cup I_1 \cup F_1 \cup I_2 \cup F_2.$$

We now define the transitions. Let $j = 1, 2$. As we saw above, a letter $b \in B_w$ is a set of pairs of states between which the same word can be read. For all $b \in B_w$, we add a transition labeled by b from $q \in I_j$ to $r \in F_j$, whenever $(q, r) \in b$. For all $b \in B_p$, we add a transition labeled by b from $q \in I_j$ to the state (r, R) if $(q, r) \in b$ and $R \subseteq r(b)$. Similarly, for all $b \in B_s$, we add a transition labeled by b from (r, R) to $q \in F_j$ if $(r, q) \in b$ and $R \subseteq \ell(b)$. Finally, for all $b \in B_i$, we add a transition labeled by b from (r, R) to (s, S) if $(r, s) \in b$, $R \subseteq \ell(b)$ and $S \subseteq r(b)$.

Observe that the size of $\widetilde{\mathcal{A}}$ is exponential in the size of $\mathcal{A}$. We now prove that the computation can be done in EXPTIME.

Lemma 6.39. *Let $\mathcal{A} = (A, Q, \delta)$ be an NFA and let $I_1, F_1, I_2, F_2 \subseteq Q$. The automaton $\widetilde{\mathcal{A}}$, defined as above from this input, can be constructed in* EXPTIME.

Proof. Testing synchronizability of a set of states, as well as testing this for a set of pairs of states, can easily be reduced to checking whether a set of NFA's has nonempty intersection. For example, given a set $R = \{r_1, \dots, r_n\} \subseteq Q$, we can define $\mathcal{A}_i$ as the NFA $\mathcal{A}$, with $I = F = \{r_i\}$. Now R is synchronizable if and only if $\bigcap_{i=1}^{n} \mathcal{A}_i \neq \varnothing$. Deciding whether a set of NFA's has a nonempty intersection is known to be a PSPACE-complete problem (this follows from [Koz77]). It follows that computing the synchronizable sets can be done in EXPTIME. It is then clear that the remaining computations can also be done in EXPTIME. □

We now prove that the construction is correct, that is, that it gives a reduction from LT- and LTT-separability to the special case that $k = 1$.

Proposition 6.40. *Let $\mathcal{A} = (A, Q, \delta)$ be an NFA, and let $I_1, F_1, I_2, F_2 \subseteq Q$. Let $\widetilde{\mathcal{A}}$ be the NFA resulting from Construction 6.38 on this input. Then,*

1. *The languages $L(\mathcal{A}, I_1, F_1), L(\mathcal{A}, I_2, F_2)$ are* LT*-separable if and only if $L(\widetilde{\mathcal{A}}, I_1, F_1)$, $L(\widetilde{\mathcal{A}}, I_2, F_2)$ are* LT[1]*-separable.*
2. *The languages $L(\mathcal{A}, I_1, F_1), L(\mathcal{A}, I_2, F_2)$ are* LTT*-separable if and only if there exists $d \in \mathbb{N}$ such that $L(\widetilde{\mathcal{A}}, I_1, F_1)$, $L(\widetilde{\mathcal{A}}, I_2, F_2)$ are* LTT$[1, d]$*-separable.*

Proof. Here, we will provide the proof for LTT, using Condition (4) from Theorem 6.32. The proof for LT is obtained similarly, using Condition (5) from Theorem 6.23.

Suppose that $L(\mathcal{A}, I_1, F_1)$ and $L(\mathcal{A}, I_2, F_2)$ are not LTT-separable. We prove that for all $d \in \mathbb{N}$, $L(\widetilde{\mathcal{A}}, I_1, F_1)$ and $L(\widetilde{\mathcal{A}}, I_2, F_2)$ are not LTT$[1, d]$-separable. Fix $d \in \mathbb{N}$. By Condition (4) of Theorem 6.32, in $\mathcal{A}$, there exists a pair in $(I_1 \times F_1) \times (I_2 \times F_2)$ that has a common d-pattern $\mathcal{P}$. That is, there are words $w_1 \in L(\mathcal{A}, I_1, F_1)$ and $w_2 \in L(\mathcal{A}, I_2, F_2)$ such that $w_1 = u_0 v_1 u_1 \dots v_n u_n$ and $w_2 = u'_0 v'_1 u'_1 \dots v'_m u'_m$ are $\mathcal{P}$-decompositions, compatible with some $(q_1, r_1) \in I_1 \times F_1$ in $\mathcal{A}$, respectively with $(q_2, r_2) \in I_2 \times F_2$ in $\mathcal{A}$. There are two cases: either $n = 0$ or $n > 0$.

In the first case, $n = 0$, which implies that $\mathcal{P} = u_0$. Then, the $\mathcal{P}$-decompositions w_1 and w_2 are both equal to u_0. It follows that $b = \{(q_1, r_1), (q_2, r_2)\} \in B_w$. Then, by construction, $b \in L(\widetilde{\mathcal{A}}, I_1, F_1) \cap L(\widetilde{\mathcal{A}}, I_2, F_2)$, thus the languages $L(\widetilde{\mathcal{A}}, I_1, F_1)$ and $L(\widetilde{\mathcal{A}}, I_2, F_2)$ are not $\mathsf{LTT}[1, d]$-separable.

In the second case, $n > 0$. Define $\widehat{w}_1 = \mathfrak{p}\mathfrak{b}_1 \cdots \mathfrak{b}_{n-1}\mathfrak{s}$, where $\mathfrak{p} = (u_0, v_1)$, $\mathfrak{s} = (v_n, u_n)$ and, for all i, $\mathfrak{b}_i = (v_i, u_i, v_{i+1})$. Similarly, we define $\widehat{w}_2 = \mathfrak{p}'\mathfrak{b}'_1 \cdots \mathfrak{b}'_{m-1}\mathfrak{s}'$. Since we started from $\mathcal{P}$-decompositions for a d-pattern $\mathcal{P}$, we have that $\mathfrak{p} = \mathfrak{p}'$, $\mathfrak{s} = \mathfrak{s}'$, and the number of times that a block $\mathfrak{b}$ occurs in w_1 and in w_2 is equal up to threshold d. It follows by construction that, over the alphabet of blocks, $\widehat{w}_1 \equiv_1^d \widehat{w}_2$. We will use these words to construct $\widetilde{w}_1, \widetilde{w}_2 \in B^*$ such that $\widetilde{w}_1 \in L(\widetilde{\mathcal{A}}, I_1, F_1)$, $\widetilde{w}_2 \in L(\widetilde{\mathcal{A}}, I_2, F_2)$, and $\widetilde{w}_1 \equiv_1^d \widetilde{w}_2$.

Let $\mathfrak{b}$ be a block that appears in $\widehat{w}_1, \widehat{w}_2$. Define $T_\mathfrak{b} \subseteq Q^2$ as the set of all pairs of states in $\mathcal{A}$ that are used to read occurrences of $\mathfrak{b}$ in the runs of w_1 and w_2. Since the $\mathcal{P}$-decompositions are compatible with their respective initial and final states, by definition, $T_\mathfrak{b}$ is synchronizable. Similarly, if $\mathfrak{p}$ (resp. $\mathfrak{s}$) is a prefix (resp. suffix) block occurring in $\widehat{w}_1, \widehat{w}_2$, we get a prefix (resp. suffix) synchronizable set $T_\mathfrak{p}$ (resp. $T_\mathfrak{s}$). We now define the following words over the alphabet B: $\widetilde{w}_1 = T_\mathfrak{p} T_{\mathfrak{b}_1} \cdots T_{\mathfrak{b}_{n-1}} T_\mathfrak{s}$ and $\widetilde{w}_2 = T_{\mathfrak{p}'} T_{\mathfrak{b}'_1} \cdots T_{\mathfrak{b}'_{m-1}} T_{\mathfrak{s}'}$. By definition of $\widetilde{\mathcal{A}}$ and the fact that the $\mathcal{P}$-decompositions w_1 and w_2 are (q_1, r_1)- resp. (q_2, r_2)-compatible, $\widetilde{w}_1 \in L(\widetilde{\mathcal{A}}, I_1, F_1)$ and $\widetilde{w}_2 \in L(\widetilde{\mathcal{A}}, I_2, F_2)$. Moreover, since $\widehat{w}_1 \equiv_1^d \widehat{w}_2$, we have $\widetilde{w}_1 \equiv_1^d \widetilde{w}_2$. We conclude that $L(\widetilde{\mathcal{A}}, I_1, F_1)$ and $L(\widetilde{\mathcal{A}}, I_2, F_2)$ are not $\mathsf{LTT}[1, d]$-separable.

Conversely, assume that for all $d \in \mathbb{N}$, $L(\widetilde{\mathcal{A}}, I_1, F_1)$ and $L(\widetilde{\mathcal{A}}, I_2, F_2)$ are not $\mathsf{LTT}[1, d]$-separable. We prove that for all $d \in \mathbb{N}$, there exists a pair in $(I_1 \times F_1) \times (I_2 \times F_2)$ that has a common d-pattern $\mathcal{P}$. Let $d \in \mathbb{N}$. By assumption, there exist $\widetilde{w}_1 \in L(\widetilde{\mathcal{A}}, I_1, F_1)$ and $\widetilde{w}_2 \in L(\widetilde{\mathcal{A}}, I_2, F_2)$, such that $\widetilde{w}_1 \equiv_1^d \widetilde{w}_2$. Again, we have two cases. By definition of $\widetilde{\mathcal{A}}$, either $\widetilde{w}_1, \widetilde{w}_2 \in B_w$ or $\widetilde{w}_1, \widetilde{w}_2 \in B_p(B_i)^* B_s$. In the first case, $\widetilde{w}_1 \equiv_1^d \widetilde{w}_2$ implies that $\widetilde{w}_1 = \widetilde{w}_2$ and this means, by definition of B_w, that there is a word w that can be read both between a pair in $I_1 \times F_1$ and between a pair in $I_2 \times F_2$. Then it suffices to take $\mathcal{P} = w$.

Otherwise, $\widetilde{w}_1 = b_p b_1 \cdots b_n b_s$ and $\widetilde{w}_2 = b_p b'_1 \cdots b'_m b_s$. By definition of B_p, B_i, B_s, to each label appearing in $\widetilde{w}_1, \widetilde{w}_2$ we can associate a block, prefix block or suffix block. Since $\widetilde{w}_1$ and $\widetilde{w}_2$ label runs in $\widetilde{\mathcal{A}}$, these blocks can be chosen in such a way that the corresponding words over the alphabet of blocks satisfy the compatibility condition. We define $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$ as the following d-pattern: $\mathfrak{p}$, $\mathfrak{s}$ are the prefix and suffix blocks associated to b_p and b_s, respectively. Since $\widetilde{w}_1 \equiv_1^d \widetilde{w}_2$, for all blocks $\mathfrak{b}$, the number of occurrences of labels b such that $\mathfrak{b}$ is associated to b is the same in $\widetilde{w}_1$ and $\widetilde{w}_2$ up to threshold d. We define $f(\mathfrak{b})$ as this number.

Finally, let $q_i \in I_i$ and $r_i \in F_i$ be such that $\widetilde{w}_i$ labels a path from q_i to r_i in $\widetilde{\mathcal{A}}$. Let $\mathfrak{b} = (v, u, v')$ be the block chosen to correspond to b. A transition labeled b from (p, P) to (s, S) implies in particular that, in $\mathcal{A}$, there is a loop labeled v resp. v' around p resp. around s, and that the word u can be read from p to s. Recall that the word of blocks corresponding to $\widetilde{w}_1$ satisfies the compatibility condition. Projecting to the first coordinate of the states in the path of $\widetilde{w}_1$ thus yields a path for a $\mathcal{P}$-decomposition w_1 in $\mathcal{A}$ that is (q_1, r_1)-compatible by construction. Similarly, from $\widetilde{w}_2$, we obtain a (q_2, r_2)-compatible run for a $\mathcal{P}$-decomposition w_2 in $\mathcal{A}$. □

Deciding LT- and LTT-separability for $k = 1$

We explain how LT- and LTT-separability can be decided when the size of k-profiles is fixed to 1. Observe that in this case, the k-profile of a position is just its label. As we explained in Section 6.3.1, decidability of LTT-separability in general (and, similarly, of LT-separability) follows from Parikh's theorem and decidability of Presburger arithmetic. However, applying these results naively yields a high complexity. We explain here how, when k is fixed to 1, these arguments can be refined in order to obtain Expspace and co-Np complexities.

Lemma 6.41. *Given an NFA accepting the languages L_1 and L_2, deciding whether there exists $d \in \mathbb{N}$ such that L_1 and L_2 are* LTT$[1, d]$*-separable is in* Expspace.

Proof. Let $\pi(L_1), \pi(L_2)$ be the commutative images of L_1 and L_2. As we explained in Section 6.3.1, non-LTT$[1, d]$-separability is equivalent to the following Presburger property: 'for all $d \in \mathbb{N}$ there exist $\bar{x}_1 \in \pi(L_1)$, $\bar{x}_2 \in \pi(L_2)$ that are equal, componentwise, up to threshold d.' By [SSMH04, Theorem 1], existential Presburger formulas for $\pi(L_1), \pi(L_2)$ can be computed in linear time, with respect to the size of the NFA (see also [VSS05], where the same technique is applied to context-free grammars). Therefore, a Presburger formula for our property can also be computed in linear time. Moreover, by definition, this property has exactly one quantifier alternation. It then follows from [RL78] that it can be decided in Expspace. Thus, LTT$[1, d]$-separability can be decided in co-Expspace, which is Expspace. □

For LT, a similar proof works. However, since d is equal to 1 for this class, the situation is a bit simpler, as we will see in the following lemma.

Lemma 6.42. *Deciding whether two regular languages, given by an accepting NFA, are* LT$[1]$*-separable is in* co-Np.

Proof. Let $\pi(L_1), \pi(L_2)$ be the commutative images of the languages L_1 and L_2. Since d is now fixed to be equal to 1, non-LT$[1]$-separability is equivalent to the following Presburger property: 'there exist $\bar{x}_1 \in \pi(L_1)$, $\bar{x}_2 \in \pi(L_2)$ that are equal, componentwise, up to threshold 1.' The Presburger formula corresponding to this property is existential. It is known that existential Presburger formulas can be decided in Np (see [BT76, GS78]). We conclude that LT$[1]$-separability is in co-Np. □

However, taking a closer look at the class LT$[1]$, one notes that membership of a word in a language of this class depends only on the set of letters that the word contains. This is since, for $k = 1$, the k-profile of each position of the word has the same length, and one can thus not deduce any information from the k-profiles about the different kinds of positions (that is, whether they are in the beginning, middle or ending of the word). Thus, LT$[1]$ is equal to the class SI of alphabet-testable languages that we encountered in Section 2.3.1. We can reuse Corollary 2.20 to see that Lemma 6.42 cannot be improved.

Corollary 6.43. *It is a* co-NP*-complete problem to decide whether two regular languages, defined by two deterministic finite automata, are* $\mathsf{LT}[1]$*-separable.*

Results on upper complexity bounds

Summarizing the results of this section, we obtain the following two propositions concerning the upper complexity bounds.

Proposition 6.44. *Deciding whether two languages, accepted by an NFA, are* LTT*-separable can be achieved in* 2-Expspace.

Proof. By Lemma 6.39, the NFA $\widetilde{\mathcal{A}}$ is built from $\mathcal{A}$ in Exptime. Proposition 6.40 states that L_1 and L_2 are LTT-separable if and only if there exists a $d \in \mathbb{N}$ such that $\widetilde{L}_1$ and $\widetilde{L}_2$ are $\mathsf{LTT}[1, d]$-separable. By Lemma 6.41, this latter property can be decided in Expspace, with respect to the size of $\widetilde{\mathcal{A}}$, which is exponential in the size of $\mathcal{A}$. It follows that deciding LTT-separability can be achieved in 2-Expspace. □

And, for LT-separability, we obtain the following proposition.

Proposition 6.45. *Deciding whether two languages L_1 and L_2, accepted by an NFA, are* LT*-separable can be achieved in* co-Nexptime.

Proof. As before, Lemma 6.39 yields that the NFA $\widetilde{\mathcal{A}}$ is built from $\mathcal{A}$ in Exptime. Also, Proposition 6.40 states that L_1 and L_2 are LT-separable if and only if $\widetilde{L}_1$ and $\widetilde{L}_2$ are $\mathsf{LT}[1]$-separable. By Lemma 6.42, this latter property can be decided in co-Np, with respect to the size of $\widetilde{\mathcal{A}}$, which is exponential in the size of $\mathcal{A}$. It follows that deciding whether L_1 and L_2 are LT-separable can be achieved in co-Nexptime. □

As we saw, Lemma 6.42 cannot be improved. Improving the co-Nexptime upper bound from Proposition 6.45 would thus require improving the reduction.

For LTT, the situation is different. It is likely that a sharper analysis of the Presburger formula would yield a better complexity result in Lemma 6.41. Indeed, while deciding Presburger formulas with one quantifier alternation is very costly in general, we only consider a very specific formula. A better complexity result in Lemma 6.41 could yield a better upper bound, even without improving the reduction.

6.4.2 Lower complexity bounds

In this section, we will prove co-Np lower bounds for both LT- and LTT-separability. The bounds hold when the input languages are given as NFA's or DFA's. Our method for proving these bounds is an extension of the method that we applied in Lemma 2.19 in Section 2.3.1, in order to show that SI-separability is co-NP-complete. We will again give a reduction of **3-SAT**, but now to the problem of LT- resp. LTT-non-separability. That is, from an arbitrary instance of **3-SAT**, we construct two DFA's and prove that the corresponding languages are not LT- resp. LTT-separable if and only if the **3-SAT** instance is satisfiable.

Proposition 6.46. *The following two problems are* CO-NP*-hard.*

Input: *An alphabet $A = \{a_1, a_2, \ldots, a_n\}$ and two languages L_1, L_2, recognized by DFA's $\mathcal{A}_1$ resp. $\mathcal{A}_2$ over A.*
Question 1: *Are L_1 and L_2 LT-separable?*
Question 2: *Are L_1 and L_2 LTT-separable?*

Proof. We will prove the statement about LTT-separability. The reduction is identical for the case of LT. Let φ be a **3-SAT** formula over the variables $\{x_1, x_2, \ldots, x_n\}$. Let A be the alphabet $\{\#, x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$. We will construct two DFA's and prove that the corresponding languages are not LTT-separable if and only if the **3-SAT** instance φ is satisfiable. Let $\mathcal{A}_1$ be the automaton depicted in Figure 6.5. Let L_1 be the language that is recognized by this automaton, by the initial and final states marked in the picture.

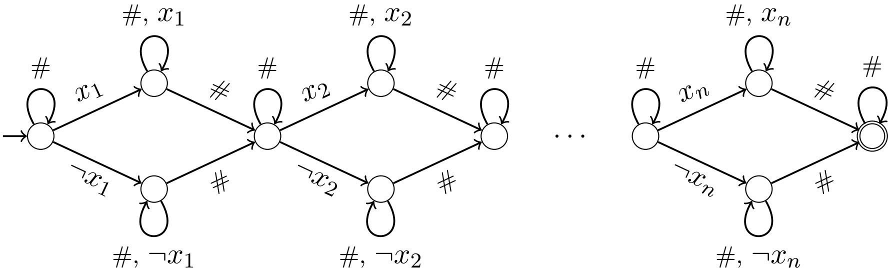


Figure 6.5: The automaton $\mathcal{A}_1$.

The automaton $\mathcal{A}_2$ is contructed as a concatenation of m subautomata, each corresponding to a clause C in φ. For every disjunct in the clause, there will be a path in the subautomaton that reads precisely that disjunct and the symbol #. This sequence of subautomata is then concatenated with a copy of the automaton $\mathcal{A}_1$. In Figure 6.6, we illustrate how to construct $\mathcal{A}_2$ precisely. The language that is recognized by the marked initial and final states, is called L_2.

We will show that φ is satisfiable if and only if L_1 and L_2 are not LTT-separable. To this end, we first prove the following claim.

Claim. There exist $u \in L_1$ and $v \in L_2$ such that $\mathsf{alph}(u) = \mathsf{alph}(v)$ if and only if for all $k, d \in \mathbb{N}$, there are $u' \in L_1$ and $v' \in L_2$ such that $u' \equiv_k^d v'$.

For the direction from right to left, let $d = k = 1$. The corresponding u' and v' then, in particular, have the same alphabet. For the converse direction, let $k, d \in \mathbb{N}$, and let u, v be as in the assumption. Inserting #-loops in the runs of u and of v sufficiently often (for example, k times for every possible loop), one can ensure that every k-profile contains at most one letter from $A \setminus \{\#\}$. Note that this also makes sure that the prefixes and suffixes of length k of the new words are the same, namely $\#^k$. Then, one inserts loops labeled by x_i resp. $\neg x_i$

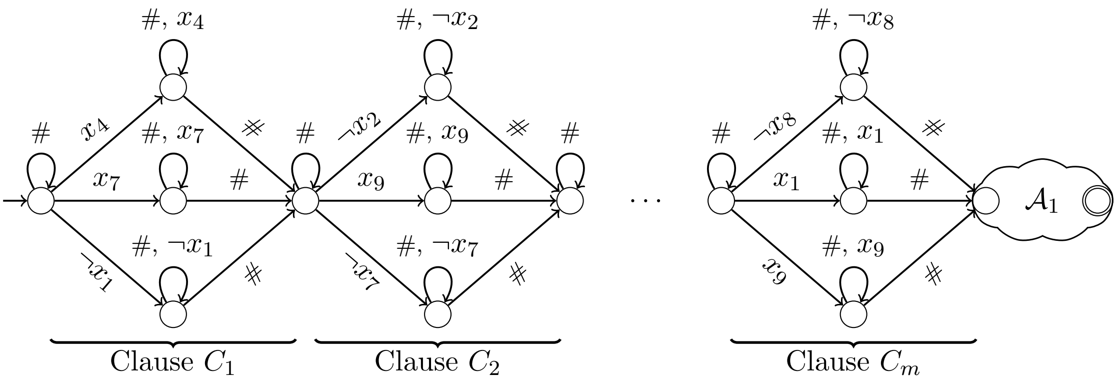


Figure 6.6: The automaton $\mathcal{A}_2$, for $\varphi = (x_4 \vee x_7 \vee \neg x_1) \wedge (\neg x_2 \vee x_9 \vee \neg x_7) \wedge (\neg x_8 \vee x_1 \vee x_9)$.

(whichever is already present in the run) in combination with #-loops, in such a way that these k-profiles occur equally often (counted up to threshold d) in both words. This process yields the desired words u' and v' such that $u' \equiv_k^d v'$.

Now, suppose that some valuation satisfies φ. One uses this valuation to define a word u in the language L_1, in the following way. Let $u = a_1\#a_2\#\dots a_n\#$. If x_i is sent to 1 by the valuation, $a_i = x_i$. Else, $a_i = \neg x_i$. Since the valuation satisfies φ, there is in each of the m clauses a disjunct d that is sent to 1. Define a word v in L_2 as $y_1\#y_2\#\dots y_m\#u$, where y_i is such a disjunct of C_i that is sent to 1. By soundness of the valuation, we have $\mathsf{alph}(u) = \mathsf{alph}(v)$. Hence, by the claim above, for all $k, d \in \mathbb{N}$, there are $u' \in L_1$ and $v' \in L_2$ such that $u' \equiv_k^d v'$. That is, L_1, L_2 are not LTT-separable.

On the other hand, suppose that L_1, L_2 are not LTT-separable. In particular, this means that there exist $u \in L_1$ and $v \in L_2$ such that $\mathsf{alph}(u) = \mathsf{alph}(v)$. Note that by construction of $\mathcal{A}_1$, we have that for all $i \in \{1, \dots, n\}$, either $x_i \in \mathsf{alph}(u)$ or $\neg x_i \in \mathsf{alph}(u)$. The word $u \in L_1$ thus defines the following valuation on the set of variables.

$$\begin{array}{rccll} val: & \{x_1, \dots x_n\} & \to & \{0,1\} & \\ & x_i & \mapsto & 1 & \text{if } x_i \in \mathsf{alph}(u) \\ & x_i & \mapsto & 0 & \text{else} \end{array}$$

The fact that $\mathsf{alph}(v) = \mathsf{alph}(u)$ thus implies that all non-# letters in v are sent to 1. It follows by the construction of $\mathcal{A}_2$ that in each of the clauses of φ, there is a disjunct that evaluates to 1. Thus, φ is satisfiable. □

6.5 Separating context-free languages by LT and LTT languages

In Section 6.3.1, we showed that LTT-separability for regular languages is decidable, using the following ingredients:

(1) Condition (2) of Theorem 6.12,

(2) Theorem 6.28,

(3) decidability of Presburger arithmetic.

Once the size k of the profiles was fixed by Condition (2) of Theorem 6.12, Theorem 6.28 allowed us to write LTT-separability as a computable Presburger formula. The decidability of LTT-separability for regular languages then followed from the decidability of Presburger arithmetic.

Theorem 6.28, which is an extension of Parikh's theorem to k-images, not only holds for regular languages, but also for context-free languages. We can thus reuse the arguments from Section 6.3.1, to prove that it is decidable whether context-free languages are separable by an LTT language with a fixed profile size k. This is shown in the following theorem. For k a fixed natural number, we write $\mathsf{LTT}[k] = \bigcup_{d \in \mathbb{N}} \mathsf{LTT}[k, d]$.

Theorem 6.47. *Let L_1, L_2 be context-free languages and $k \in \mathbb{N}$. It is decidable whether L_1, L_2 are $\mathsf{LTT}[k]$-separable.*

Proof. The proof is similar to the proof of Theorem 6.29. The languages L_1, L_2 are $\mathsf{LTT}[k]$-separable if and only if there exists $d \in \mathbb{N}$ such that they are $\mathsf{LTT}[k, d]$-separable. This is the case if and only if there exists $d \in \mathbb{N}$ such that there are no words $w_1 \in L_1, w_2 \in L_2$ with $w_1 \equiv_k^d w_2$. By Fact 6.27, this can be expressed in terms of k-images as the following statement: there exists $d \in \mathbb{N}$ such that there do not exist any vectors $\overline{x}_1 \in \pi_k(L_1), \overline{x}_2 \in \pi_k(L_2)$ that are equal, componentwise, up to threshold d.

By Theorem 6.28, there are computable Presburger formulas for the sets $\pi_k(L_1)$ and $\pi_k(L_2)$. The above statement can thus be expressed as a computable Presburger formula. Since Presburger arithmetic is decidable [Pre29, Sko31], it follows that $\mathsf{LTT}[k]$-separability of context-free languages is decidable. □

An interesting consequence of Theorem 6.47 is that $\mathsf{LTT}[1]$-separability of context-free languages is decidable. A language is in $\mathsf{LTT}[1]$ if and only if it can be defined by a first-order logic formula that can test equality between positions, but cannot test ordering. This result is surprising, since membership of a context-free language in this class is undecidable. We prove this in Theorem 6.49. Our proof follows the lines of the proof of Greibach's theorem [Gre68], which is used in particular to prove that it is undecidable whether a context-free language is regular. In order to adapt the proof of Greibach's theorem to the class of $\mathsf{LTT}[1]$, we need that this class is closed under right residuals.

Lemma 6.48. *For every $k, d \in \mathbb{N}$, the class $\mathsf{LTT}[k, d]$ is closed under right residuals. Furthermore, the classes $\mathsf{LTT}, \mathsf{LTT}[k]$, LT, and $\mathsf{LT}[k]$ are closed under right residuals.*

Proof. Note that the second statement is an immediate consequence of the first statement, as these classes are unions of classes of the form $\mathsf{LTT}[k, d]$, for certain choices of k and d. To prove the first statement, let $k, d \in \mathbb{N}$ and let $L \in \mathsf{LTT}[k, d]$. Let $v \in L/a = \{w \mid wa \in L\}$ and let $u \in A^*$ be such that $u \equiv_k^d v$. Note that $u \equiv_k^d v$ implies that u and v have the same suffix of length $k - 1$. Concatenating both u and v with the letter a thus influences the sets of k-profiles of the words in the same way: every k-profile in which this new letter occurs, contains only positions of the original word that were inside the suffix of length

$k-1$. It follows that $ua \equiv_k^d va$, and since $v \in L/a$, it follows that $u \in L/a$ as well. Thus, $L/a \in \mathsf{LTT}[k,d]$. □

Theorem 6.49. *The membership problem for context-free languages in the class* $\mathsf{LTT}[1]$ *is undecidable.*

Proof. We reduce the undecidable problem of universality of context-free languages to this membership problem. Let L be a context-free language over A and let $\# \notin A$. Let K be a context-free language that is not in $\mathsf{LTT}[1]$, and define $L_1 = (K \cdot \# \cdot A^*) \cup (A^* \cdot (\# \cdot L)^+)$. Clearly, a context-free grammar for L_1 can be computed from context-free grammars for L and K. We show that $L = A^*$ if and only if $L_1 \in \mathsf{LTT}[1]$.

If $L = A^*$, then $L_1 = (K \cdot \# \cdot A^*) \cup (A^* \cdot (\# \cdot A^*)^+) = A^* \cdot (\# \cdot A^*)^+ = (A \cup \{\#\})^* \cdot \# \cdot (A \cup \{\#\})^*$. The first-order formula $\exists x.\ \#(x)$ then witnesses that $L_1 \in \mathsf{LTT}[1]$.

For the converse direction, we use Lemma 6.48. By assumption, $L_1 \in \mathsf{LTT}[1]$. Suppose that $L \neq A^*$. Let $w \in A^* \backslash L$. Consider the quotient $L_1/\#w = \{v \mid v\#w \in L_1\}$. By definition of L_1, this quotient is equal to K. Lemma 6.48 implies that $K = L_1/\#w \in \mathsf{LTT}[1]$, which contradicts the definition of K. It follows that $L = A^*$. □

In a similar fashion, we obtain that the membership problems for context-free languages in the classes LT and LTT are undecidable as well.

Theorem 6.50. *The membership problems for context-free languages in the classes* LT *and* LTT *are undecidable.*

Proof. This can be shown by replacing each occurrence of $\mathsf{LTT}[1]$ in the proof of Theorem 6.49 by LT resp. LTT. This works since Lemma 6.48 holds for these classes too, and, whenever $L = A^*$, the language L_1 is also in LT and in LTT. □

The results of Theorem 6.47 and Theorem 6.49 may seem contradictory. Indeed, in the setting of regular languages, a language belongs to a class if and only if this class can separate the language from its complement. It follows that membership can be reduced to separability. However, the class of context-free languages is not closed under complement, such that the reduction no longer holds in this broader setting.

An interesting question is whether full LT- and LTT-separability of context-free languages is decidable as well. In view of Theorem 6.50, this would also be surprising. As we saw before, two out of three ingredients of our method from Section 6.3.1 still work in the setting of context-free languages. A possible approach to proving decidability of LT- resp. LTT-separability of context-free languages would thus be to generalize Condition (2) of Theorem 6.12. This amounts to finding a bound on the size of the profiles that a potential separator has to consider. However, as we will show in Theorem 6.51, it turns out that LT- and LTT-separability of context-free languages are undecidable. In particular, this implies that generalizing Condition (2) of Theorem 6.12 to context-free languages is not possible.

As we have mentioned in Chapter 2, it is proven in [SW76] that separability of context-free languages by regular languages is undecidable. The proof of this fact, as provided in [SW76, Theorem 4.6], is a reduction from the halting problem on Turing machines to this separation problem. It turns out that this reduction actually works for any class of regular separators that contains all languages of the form $K_1A^* \cup K_2$, where K_1, K_2 are finite languages. Since this is clearly the case for both LT and LTT, the undecidability of LT- and LTT-separability of context-free languages follows along the same lines. In Theorem 6.51 we provide a version of this proof tailored to the classes of LT and LTT.

Theorem 6.51. *The* LT*-separation problem and the* LTT*-separation problem are undecidable problems for context-free languages.*

Proof. We will reduce the halting problem on Turing machines to LT-separability and LTT-separability of context-free languages. The reduction is the same for both LT and LTT. Consider a deterministic Turing machine $\mathcal{M}$. We prove that it is possible to compute context-free languages L_1, L_2 from $\mathcal{M}$, such that $\mathcal{M}$ halts on the empty input if and only if L_1, L_2 are LT-separable, if and only if L_1, L_2 are LTT-separable.

Let A be the alphabet of $\mathcal{M}$, let Q be its set of states, and let $B = A \cup (A \times Q) \cup \{\#, \gamma\}$ where $\#, \gamma \notin A$. As usual, a configuration of $\mathcal{M}$ is encoded as a word in $A^* \cdot (A \times Q) \cdot A^* \subseteq B^*$. The word $(u, (a, q), v)$ means that $\mathcal{M}$ is in state q, the tape holds $u \cdot a \cdot v$, and the head currently scans the distinguished a position. Finally, if $w \in B^*$, we denote by w^R the mirror image of w.

We now define the context-free languages L_1, L_2 over B. The language L_1 contains all words of the form

$$c_1\#c_2^R\#c_3\#c_4^R \cdots \#c_{2k-1}\#c_{2k}^R\#\gamma^k,$$

that are such that $c_1, \dots, c_{2k}$ are encodings of configurations of $\mathcal{M}$, and for all $i \leq k$, $c_{2i-1} \vdash_{\mathcal{M}} c_{2i}$ (that is, c_{2i} is the configuration of $\mathcal{M}$ that is reached after one computation step from configuration c_{2i-1}). Similarly, L_2 contains all words of the form

$$c_1\#c_2^R\#c_3\#c_4^R \cdots \#c_{2k-1}\#c_{2k}^R\#\gamma^{2k},$$

that are such that $c_1, \dots, c_{2k}$ are encodings of configurations of $\mathcal{M}$, c_1 is the initial configuration of $\mathcal{M}$ starting with an empty input and for all $i \leq k-1$, $c_{2i} \vdash_{\mathcal{M}} c_{2i+1}$. One can verify that L_1, L_2 are indeed context-free languages and that grammars for L_1, L_2 can be computed from $\mathcal{M}$ (for example, using [Har67]). We will regard prefixes that are common to both languages. To this end, let $c_1, c_2, \dots, c_{i-1}, c_i$ be a sequence of configurations and let $w \in B^*$ be the word defined as

$$w \quad = \begin{cases} c_1\#c_2^R\#c_3\#c_4^R \cdots \#c_{2k-1}\#c_{2k}^R\#, & \text{if } i = 2k, \text{and} \\ c_1\#c_2^R\#c_3\#c_4^R \cdots \#c_{2k-1}\#c_{2k}^R\#c_{2k+1}\#, & \text{if } i = 2k+1. \end{cases}$$

Claim 1. *If w is both a prefix of a word in L_1 and of a word in L_2, then $c_1, c_2, \dots, c_i$ are the first i configurations of the run of $\mathcal{M}$ starting from the empty input. Moreover, if $i = 2k$ and*

$$w \cdot c \cdot \# \quad = \quad c_1\#c_2^R\#c_3\#c_4^R \cdots \#c_{2k-1}\#c_{2k}^R\#c\#$$

is a prefix of a word in L_2, then c is configuration $(i+1)$ in the run. And, if $i = 2k+1$ and

$$w \cdot c^R \cdot \# \quad = \quad c_1\#c_2^R\#c_3\#c_4^R \cdots \#c_{2k-1}\#c_{2k}^R\#c_{2k+1}\#c^R\#$$

is a prefix of a word in L_1, then c is configuration $(i+1)$ in the run.

The claim follows by definition of L_1 and L_2. Since w is a prefix of some word w_2 in L_2, c_1 is the initial configuration of $\mathcal{M}$ starting with an empty input. Then, since w is a prefix of some word w_1 in L_1, $c_1 \vdash_{\mathcal{M}} c_2$, and again since w is a prefix of w_2, $c_2 \vdash_{\mathcal{M}} c_3$, and so on. Thus, $c_1, c_2, \dots, c_i$ are the first i configurations of the run of $\mathcal{M}$ starting from the empty input. If $i = 2k$, and $w \cdot c \cdot \#$ is a prefix of a word in L_2, we have $c_{2k} \vdash_{\mathcal{M}} c$ by definition of L_2, and thus c is configuration $(i+1)$ in the run. The result for the case $i = 2k+1$ is obtained similarly.

We will now prove that this indeed gives a reduction, that is, that $\mathcal{M}$ halts on the empty input if and only if L_1, L_2 are LT-separable, if and only if L_1, L_2 are LTT-separable.

Assume first that $\mathcal{M}$ does not halt on the empty input. This means that the run of $\mathcal{M}$ is an infinite sequence of configurations $c_1, c_2, c_3, \dots$. Then by definition of L_1, L_2, for all $k \in \mathbb{N}$,

$$\begin{aligned} &c_1\#c_2^R\#c_3\#c_4^R \cdots \#c_{2k-1}\#c_{2k}^R\#\gamma^k && \in L_1, \\ &c_1\#c_2^R\#c_3\#c_4^R \cdots \#c_{2k-1}\#c_{2k}^R\#\gamma^{2k} && \in L_2. \end{aligned}$$

It then follows from the following claim that L_1, L_2 cannot be separated by an LT or LTT language, as they cannot even be separated by a regular language.

Claim 2. If $\mathcal{M}$ does not halt on the empty input, the languages L_1 and L_2 cannot be separated by a regular language.

This is exactly [SW76, Theorem 4.6, case 1]. Suppose there is a DFA $\mathcal{A}$ that separates L_1 and L_2. Let $n-1$ be its number of states. Let $c_1, c_2, c_3, \dots, c_{2\cdot n!}$ be the first $2 \cdot n!$ configurations of the run of $\mathcal{M}$ on the empty input. Define $z = c_1\#c_2^R\#c_3\#c_4^R \cdots \#c_{2\cdot n!-1}\#c_{2\cdot n!}^R\#$. By the above, the words $z\gamma^{n!} \in L_1$, and $z\gamma^{2\cdot n!} \in L_2$. It follows by pumping arguments that $\mathcal{A}$ cannot separate these words.

For the other direction of the reduction, assume that $\mathcal{M}$ halts on the empty input within ℓ steps, i.e., its run is $c_1 \vdash_{\mathcal{M}} c_2 \vdash_{\mathcal{M}} \dots \vdash_{\mathcal{M}} c_\ell$ and c_ℓ is the halting configuration. Before providing an LT-separator for this case, we observe that sufficiently long prefixes of words of L_1, L_2 are distinct.

Claim 3. *Let $w_1 \in L_1$ and $w_2 \in L_2$ be words that have prefixes u_1 resp. u_2 of length $\ell(\ell+1)+2\ell$. Then $u_1 \neq u_2$.*

We prove this claim by contradiction. Assume that $u_1 = u_2$, and let u be the largest prefix of $u_1 = u_2$ that is either of the form

$$\begin{aligned} u = \ & c_1\#c_2^R\#c_3\#c_4^R \cdots \#c_{i-1}\#c_i^R\#, \quad \text{or of the form} \\ u = \ & c_1\#c_2^R\#c_3\#c_4^R \cdots \#c_{i-1}^R\#c_i\#, \end{aligned}$$

depending on whether i is even or not. By definition, $u_1 = u_2 = u \cdot v$, where v is either of the form γ^j with $j \leq i/2$, or is a prefix of c or c^R for some configuration c of $\mathcal{M}$.

Assume first that $v = \gamma^j$, with $j \leq i/2$. By Claim 1, $c_1, c_2, \dots, c_{i-1}, c_i$ are the first i configurations of the run of $\mathcal{M}$ starting from the empty input. Since $\mathcal{M}$ halts in ℓ steps, this means that $i \leq \ell$ and that each of the i configurations has length $\leq \ell+1$. It follows that u is of length $\leq \ell(\ell+1)+\ell$. Therefore, $|u_1| = |u_2| \leq \ell(\ell+1)+\ell+j \leq \ell(\ell+1)+\ell+i/2 \leq \ell(\ell+1)+\ell+\ell/2$. This gives a contradiction, because by definition, u_1, u_2 are of length $\ell(\ell+1)+2\ell$.

Now assume that v is the prefix of c or c^R for some configuration c of $\mathcal{M}$. By Claim 1, c is such that $c_1, c_2, \dots, c_{i-1}, c_i, c$ are the first $i+1$ configurations of the run of $\mathcal{M}$ starting from the empty input. Since $\mathcal{M}$ halts in ℓ steps, this means that $i+1 \leq \ell$ and that each configuration has length $\leq \ell+1$. Thus, $|u| \leq (\ell-1)(\ell+1)+\ell-1$ and $|v| \leq \ell+1$. It follows that $u_1 = u_2$ is of length $\leq \ell(\ell+1)+\ell-1$, which gives again a contradiction with the definition.

We will now show how Claim 3 can be used to define an LT language that separates L_1 from L_2. Let K_1 be the language of words of L_1 of strictly smaller length than $\ell(\ell+1)+2\ell$. Let K_2 be the set of prefixes of length $\ell(\ell+1)+2\ell$ of words in L_1. Now define $L = K_1 \cup K_2 \cdot B^*$. By definition, K_1, K_2 are finite languages, hence L is clearly in LT (and therefore in LTT). We will prove that L separates L_1 from L_2.

By definition, $L_1 \subseteq L$. Now let $w \in L_2$. Clearly, w cannot be in K_1, since, by definition of L_1, all words in K_1 are of the form

$$c_1 \# c_2^R \# c_3 \# c_4^R \cdots \# c_{2k-1} \# c_{2k}^R \# \gamma^k.$$

But, as $w \in L_2$, it has twice as many letters γ at the end. And, w can neither be in $K_2 \cdot B^*$, because of the following. By definition, $K_2 \cdot B^*$ contains all words for which there is a word in L_1 that has the same prefix of length $\ell(\ell+1)+2\ell$. By Claim 3, this is not the case for w. It follows that $L \cap L_2 = \varnothing$.

We deduce that L_1, L_2 are LT-separable, and therefore also LTT-separable. □

Conclusion and perspectives

This thesis reflects the results of our study of the separation problem for various subclasses of the regular languages. For regular input languages, this problem subsumes the classical membership problem, which is considered to be one of the main tools to understand a class of languages described in terms of logic or combinatorial properties. The separation problem is a tool to study the ability of a class of languages to perceive phenomena outside of this class of languages. More precisely, it asks whether a class of languages can witness the fact that two languages, that can be outside of the class, are disjoint. Besides that solving this problem provides more information about the class than solving the membership problem does, it also seems to be a more robust notion. For example, it is known from [Aui10] that decidability of the membership problem is not preserved under the operation $\mathsf{V} \mapsto \mathsf{V} * \mathsf{D}$, while this is the case for decidability of the separation problem [Ste01].

The separation problem for classes of languages emerged first in an algebraic context in the form of pointlike sets, and in a profinite context as a topological separation problem. In Chapter 2, we discussed the result of [Alm99] that says that the separation problem for a variety V is equivalent to the algebraic problem of finding the 2-V-pointlike sets, and to the topological problem of testing whether closures of two regular languages in the free pro-V semigroup are disjoint. These problems have been studied for various classes of languages, using involved techniques from the theory of profinite semigroups.

Contributions

In this dissertation, we focused on the separation problem for the following four classes of languages: piecewise testable languages, unambiguous languages, locally testable languages and locally threshold testable languages. For the class of unambiguous languages, it was not yet known whether the separation problem was decidable. In Theorem 5.5, we show that this is indeed the case. For the other classes of languages, the decidability of the separation problem had already been proved using algebraic or topological arguments. These arguments only provide a yes/no answer to these separation problems, and do not provide separating languages. Our motivation for studying this problem was to find proofs that do not rely on the profinite theory behind these problems, but only use combinatorial arguments. We believe that these proofs provide more insight in the separation problems. Indeed, our approach also provides separating languages in case they exist.

Our contribution to this field of research is threefold. First of all, we provide combinatorial proofs of the decidability of the PT-, $\mathsf{FO}^2(<)$-, LT-, and LTT-separation problems. Our proofs only use elementary combinatorial techniques, and are therefore much simpler than the previously known proofs. We also provide some complexity results for these separation problems. For example, we showed that the PT-separation problem can be decided in Ptime with respect to both the size of a recognizing automaton and the size of the alphabet. This improved the existing upper bound, which was Ptime with respect to the size of the automaton, but Exptime with respect to the size of the alphabet.

Secondly, our approach to these problems is to establish bounds on parameters of the class of languages (such as the length of the pieces, the length of the prefixes, infixes and suffixes, or the quantifier rank). These are such that only languages whose parameters are below the bounds are relevant to decide the separation problem. An advantage of this approach is that one immediately obtains a brute force algorithm to test separability, and, more importantly, one obtains a description of a separator in case it exists.

Finally, this combinatorial approach gives an outline along which one can try to solve the separation problem for other classes as well. For example, recently, this has successfully been done for the class of languages recognizable by $\mathsf{FO}(<)$ in [PZ14b].

Recent developments and future directions

Recently, our result about the decidability of LTT-separability has been generalized in [PZ14c], to obtain a transfer result that works for natural fragments of first-order logic. This result says that the decidability of separability by a fragment of the form $\mathcal{F}(<, +1, \min, \max)$ reduces to the decidability of separability by a fragment of the form $\mathcal{F}(<)$.

Improving results

Some questions still remain open. One question concerns the description of the separators. The descriptions that we have found are in the form of a saturation of one of the two languages, with respect to a congruence relation that depends on the class of languages that we study, and on the bound established on the parameters of this class. While these descriptions of separators form a new contribution, it is usually not clear how to find a more insightful description of such a separator. One could for example try to find a recognizing automaton or semigroup for this separator to obtain more information about it.

Also, there are still some complexity gaps in our results. Most importantly, we did not try to find a lower bound for $\mathsf{FO}^2(<)$. Also, the lower bound we found for both LT and LTT is co-NP, while the upper bounds we found are co-Nexptime resp. 2-Expspace.

A different notion

Recall that if two languages are $\mathcal{S}$-separable, for a class $\mathcal{S}$, then, from the point of view of $\mathcal{S}$, these languages are sufficiently different to be perceived as disjoint. The separation problem tests the discriminative power of $\mathcal{S}$, which is more informative than the expressive power.

One could pursue this generalization one step further, and call a language L_1 $\mathcal{S}$-different from a language L_2 if these are perceived as *different* by $\mathcal{S}$. That is, if there exists a language $L \in \mathcal{S}$ such that $L_1 \cap L \neq \varnothing$ and $L_2 \cap L = \varnothing$. Clearly, L_1 is $\mathcal{S}$-different from L_2 if and only if L_1 contains a word w such that $\{w\}$ is $\mathcal{S}$-separable from L_2. If $\mathcal{S}$ contains the languages consisting of a single word, then this notion does not have much content. This is not the case, for example, for the class of languages recognized by finite groups, and there is a relation between this notion for these languages and Hall's theorem [Hal50]. It is not clear right away what, in general, the logical counterpart of this notion would be, and whether it would be interesting to study this notion.

Other classes of separators

It would be interesting to see what our approach would give for the class of group languages and compare this with existing results. As we saw in Chapter 3, the separation problem for group languages is decidable and one can even explicitly describe the separating group languages if it exists.

A possible variation of the class LTT would be to add modulo predicates. The infixes can then be counted modulo constants. It is to be expected that a bound on these constants can be established in a way similar to the bounds that we found on the size of the infixes and on the counting threshold. Solving the separation problem for this class might give insight in the separation problem for other classes to which modulo predicates are added as well, in the same fashion as studying the separation problem for LTT (which corresponds to $\mathsf{FO}(=,+1)$) gave insight in the separation problems for other fragments to which the successor relation is added.

It is also to be expected that the separation problem for the class of strongly locally testable languages, i.e. the class of languages for which membership of a word depends on the infixes of a certain length occurring in the word (and not on the prefixes or suffixes), could be solved by making adjustments to the proofs of Chapter 6. This would be especially interesting since the semigroups recognizing these languages do not form a variety.

More in general, it would be interesting to study the separation problem for classes of languages that do not form a variety, such as lattices of languages. It is shown in [GGP08] that these classes admit an equational description, and it would be interesting to see how our approach with indistinguishable pairs can be adapted to apply to such classes of separators. A step in this direction was recently made in [PZ14a], where the class $\Sigma_2(<)$ is studied. This is a class of languages that is not closed under complement, and to accommodate for this, the relation of indistinguishability is replaced by an asymmetric relation.

Besides widening the class of separators, one could also study the separation problem for more involved structures, such as regular tree languages, rather than regular language over finite words. This is expected to be much more difficult, since the theory of algebraic characterizations for these languages is less advanced.

Possible applications

Our motivation for studying this problem was purely theoretical interest. However, if the separation problem is decidable for a class $\mathcal{S}$, which usually is a simple class, then this can give a simple way to express the fact that two - possibly very complicated - languages are disjoint. This could be useful in applications as verification or database theory (see also [CMM13]), for example, if one wants to over-approximate a complicated specification by a simpler one, while avoiding a second, complicated, forbidden specification. To this end, one could implement algorithms to decide separability for simple classes of separators, for example the classes of piecewise testable languages, prefix- and suffix-testable languages.

Bibliography

[ACZ08] Jorge Almeida, José Carlos Costa, and Marc Zeitoun. Pointlike sets with respect to R and J. *J. Pure Appl. Algebra*, 212(3):486–499, 2008. Cited on pages xi, 6, 21, and 47.

[Alm91] Jorge Almeida. Implicit operations on finite $\mathcal{J}$-trivial semigroups and a conjecture of I. Simon. *J. Pure Appl. Algebra*, 69(3):205–218, 1991. Cited on page 50.

[Alm94] Jorge Almeida. *Finite semigroups and universal algebra*, volume 3 of *Series in Algebra*. World Scientific, 1994. Cited on pages 11, 21, 24, 26, 32, and 62.

[Alm99] Jorge Almeida. Some algorithmic problems for pseudovarieties. *Publ. Math. Debrecen*, 54(suppl.):531–552, 1999. Automata and formal languages, VIII (Salgótarján, 1996). Cited on pages xi, 5, 9, 21, 22, 23, 24, 26, 38, 84, and 121.

[AS75] Anatolij W. Anissimov and Franz D. Seifert. Zur algebraischen Charakteristik der durch kontext-freie Sprachen definierten Gruppen. *Elektronische Informationsverarbeitung und Kybernetik*, pages 695–702, 1975. Cited on page 40.

[AS05] Karl Auinger and Benjamin Steinberg. A constructive version of the Ribes-Zalesskiĭ product theorem. *Mathematische Zeitschrift*, 250(2):287–297, 2005. Cited on pages xi, 6, 9, 21, 37, 38, 40, and 44.

[Ash91] Chris J. Ash. Inevitable graphs: a proof of the type II conjecture and some related decision procedures. *Internat. J. Algebra Comput.*, 1:127–146, 1991. Cited on pages xi, 5, 6, 21, 37, and 39.

[Aui04] Karl Auinger. A new proof of the Rhodes type II conjecture. *International Journal of Algebra and Computation*, 14(05n06):551–568, 2004. Cited on pages xi, 6, and 21.

[Aui10] Karl Auinger. On the decidability of membership in the global of a monoid pseudovariety. *Internat. J. Algebra Comput.*, 20(2):181–188, 2010. Cited on pages 4, 6, 19, and 121.

[AZ97] Jorge Almeida and Marc Zeitoun. The pseudovariety J is hyperdecidable. *RAIRO Inform. Théor. Appl.*, 31(5):457–482, 1997. Cited on pages xi, 6, 21, 47, and 48.

[Boj07] Mikołaj Bojańczyk. A new algorithm for testing if a regular language is locally threshold testable. *Inf. Process. Lett.*, 104(3):91–94, 2007. Cited on pages 100 and 101.

[BP91] Danièle Beauquier and Jean-Éric Pin. Languages and scanners. *Theoret. Comput. Sci.*, 84(1):3–21, 1991. Cited on pages 84, 86, and 88.

[BS73] J.A. Brzozowski and I. Simon. Characterizations of locally testable events. *Discrete Mathematics*, 4(3):243–271, 1973. Cited on page 87.

[BT76] I. Borosh and L. B. Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Proc. Amer. Math. Soc.*, 55(2):299–304, 1976. Cited on page 112.

[CM14] Wojciech Czerwiński and Wim Martens. A note on decidable separability by piecewise testable languages. `http://arxiv.org/pdf/1410.1042.pdf`, 2014. Cited on pages 9, 48, 61, and 63.

[CMM13] Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proc. of ICALP'13*, volume 7966 of *Lect. Notes Comp. Sci.*, pages 150–161. Springer, 2013. Cited on pages 48 and 124.

[CMM$^+$14] Wojciech Czerwiński, Wim Martens, Tomáš Masopust, Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. On separation by piecewise testable languages. In preparation, 2014. Cited on pages 47 and 53.

[CN09] José Carlos Costa and Conceição Nogueira. Complete reducibility of the pseudovariety LSl. *Internat. J. Algebra Comput.*, 19(02):247–282, 2009. Cited on page 84.

[Col10] Thomas Colcombet. Factorization forests for infinite words and applications to countable scattered linear orderings. *Theor. Comput. Sci.*, 411(4-5):751–764, 2010. Cited on page 61.

[Cos01] José Carlos Costa. Free profinite locally idempotent and locally commutative semigroups. *J. Pure Appl. Algebra*, 163(1):19–47, 2001. Cited on page 84.

[DGK08] Volker Diekert, Paul Gastin, and Manfred Kufleitner. A survey on small fragments of first-order logic over finite words. *International Journal of Foundations of Computer Science*, 19(3):513–548, June 2008. Cited on page 68.

[Eil76] Samuel Eilenberg. *Automata, languages, and machines. Vol. B.* Academic Press [Harcourt Brace Jovanovich, Publishers], New York-London, 1976. With two chapters ("Depth decomposition theorem" and "Complexity of semigroups and morphisms") by Bret Tilson, Pure and Applied Mathematics, Vol. 59. Cited on pages x and 3.

[EVW97] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. In *LICS*, pages 228–235. IEEE Computer Society, 1997. Cited on page 68.

[EVW02] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Information and Computation*, 179(2):279 – 295, 2002. Cited on page 68.

[GGP08] Mai Gehrke, Serge Grigorieff, and Jean-Eric Pin. Duality and equational theory of regular languages. In *ICALP*, pages 246–257, 2008. Cited on page 123.

[Gil96] Robert H. Gilman. Formal languages and infinite groups. In *Geometric and computational perspectives on infinite groups (Minneapolis, MN and New Brunswick, NJ, 1994)*, volume 25 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 27–51. Amer. Math. Soc., Providence, RI, 1996. Cited on pages 40 and 44.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990. Cited on page 31.

[Gre68] Sheila Greibach. A note on undecidable properties of formal languages. *Mathematical systems theory*, 2(1):1–6, 1968. Cited on page 116.

[GS66] Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966. Cited on page 101.

[GS78] Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proc. Amer. Math. Soc.*, 72:155–158, 1978. Cited on page 112.

[Hal50] Marshall Hall, Jr. A topology for free groups and related groups. *Ann. of Math. (2)*, 52:127–139, 1950. Cited on pages 37, 39, and 123.

[Har67] Juris Hartmanis. Context-free languages and Turing machine computations. *Proc. Sympos. Appl. Math.*, 19:42–51, 1967. Cited on page 118.

[Hen88] Karsten Henckell. Pointlike sets: the finest aperiodic cover of a finite semigroup. *J. Pure Appl. Algebra*, 55(1-2):85–126, 1988. Cited on pages xi, 5, 6, and 21.

[HJM14] Stepan Holub, Galina Jirásková, and Tomáš Masopust. On upper and lower bounds on the length of alternating towers. In *MFCS*, pages 315–326, 2014. Cited on page 48.

[HMPR91] Karsten Henckell, Stuart W. Margolis, Jean-Éric Pin, and John Rhodes. Ash's type II theorem, profinite topology and Malcev products. I. *Internat. J. Algebra Comput.*, 1(4):411–436, 1991. Cited on pages 5, 37, and 39.

[HRS10] Karsten Henckell, John Rhodes, and Benjamin Steinberg. Aperiodic pointlikes and beyond. *IJAC*, 20(2):287–305, 2010. Cited on pages xi, 6, and 21.

[Hun82] Harry B. Hunt, III. On the decidability of grammar problems. *J. ACM*, 29(2):429–447, 1982. Cited on page 20.

[Imm82] Neil Immerman. Upper and lower bounds for first order expressibility. *Journal of Computer and System Sciences*, 25(1):76 – 98, 1982. Cited on page 70.

[Imm99] Neil Immerman. *Descriptive complexity.* Graduate texts in computer science. Springer, 1999. Cited on page 70.

[Kam68] Hans Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California, Los Angeles (California), 1968. Cited on page 2.

[Klí11] Ondřej Klíma. Piecewise testable languages via combinatorics on words. *Discrete Math.*, 311(20):2124–2127, 2011. Cited on page 50.

[KM02] Ilya Kapovich and Alexei Myasnikov. Stallings foldings and subgroups of free groups. *Journal of Algebra*, 248(2):608 – 668, 2002. Cited on page 40.

[KMM89] Sam Kim, Robert McNaughton, and Robert McCloskey. A polynomial time algorithm for the local testability problem of deterministic finite automata. In *Algorithms and Data Structures*, number 382 in Lect. Notes Comp. Sci., pages 420–436. Springer, 1989. Cited on page 87.

[Koz77] Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266. IEEE Computer Society, 1977. Cited on page 110.

[Kuf08] Manfred Kufleitner. The height of factorization forests. In *MFCS*, volume 5162 of *LNCS*, pages 443–454. Springer, 2008. Cited on page 61.

[McN74] Robert McNaughton. Algebraic decision procedures for local testability. *Math. Systems Theory*, 8(1):60–76, 1974. Cited on page 87.

[MP71] Robert McNaughton and Seymour Papert. *Counter-free automata*. The M.I.T. Press, 1971. Cited on pages x, 3, and 87.

[Mun00] James R. Munkres. *Topology*. Prentice Hall, Incorporated, 2000. Cited on page 24.

[Nog10] Conceição Veloso Nogueira. Propriedades algorítmicas envolvendo a pseudovariedade LSl. PhD thesis, available at http://hdl.handle.net/1822/12277, 2010. Cited on pages xi, 6, and 21.

[Par66] Rohit J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966. Cited on pages 100 and 101.

[Pin84] Jean-Éric Pin. *Variétés de langages formels*. Études et Recherches en Informatique. Masson, Paris, 1984. With a preface by M. P. Schützenberger. Cited on page 50.

[Pin91] Jean-Éric Pin. Topologies for the free monoid. *Journal of Algebra*, 137(2):297 – 337, 1991. Cited on page 40.

[Pin96] Jean-Éric Pin. The expressive power of existential first order sentences of Büchi's sequential calculus. In *Proc. of ICALP'96*, number 1099 in Lect. Notes Comp. Sci., pages 300–311. Springer, 1996. Cited on page 88.

[Pin97] Jean-Éric Pin. Syntactic semigroups. In *Handbook of language theory, Vol. I*, pages 679–746. Springer, 1997. Cited on page 11.

[Pin05] Jean-Éric Pin. Expressive power of existential first-order sentences of Büchi's sequential calculus. *Discrete Math.*, 291(1–3):155–174, 2005. Cited on page 88.

[Pin09] Jean-Éric Pin. Relational morphisms and pointlike sets. Not published, 2009. Cited on pages 20 and 23.

[Pin11] Jean-Éric Pin. Mathematical foundations of automata theory. Lecture notes, 2011. Cited on pages 11 and 24.

[PP04] Dominique Perrin and Jean-Éric Pin. *Infinite words: automata, semigroups, logic and games*. Pure and applied mathematics. Academic, London, San Diego (Calif.), 2004. Cited on page 32.

[PR91] Jean-Éric Pin and Christophe Reutenauer. A conjecture on the Hall topology for the free group. *Bull. London Math. Soc.*, 23(4):356–362, 1991. Cited on pages 26, 37, 39, and 40.

[Pre29] Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I congrès de Mathématiciens des Pays Slaves. Warszawa*, pages 92–101, 1929. Cited on pages 100, 102, and 116.

[PvRZ13a] Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating regular languages by locally testable and locally threshold testable languages. In *FSTTCS*, pages 363–375, 2013. Cited on page 83.

[PvRZ13b] Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *MFCS*, pages 729–740, 2013. Cited on pages 47, 53, 55, 67, and 72.

[PvRZ14] Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. On separation by locally testable and locally threshold testable languages. *Logical Methods in Computer Science*, 10(3), 2014. Cited on page 83.

[PW96] Jean-Éric Pin and Pascal Weil. Profinite semigroups, Mal'cev products and identities. *J. Algebra*, 182:604–626, 1996. Cited on page 5.

[PW97] Jean-Éric Pin and Pascal Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30(4):383–422, 1997. Cited on pages 2 and 68.

[PZ14a] T. Place and M. Zeitoun. Going higher in the first-order quantifier alternation hierarchy on words. In *ICALP'14*, 2014. Cited on page 123.

[PZ14b] Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. In *CSL-LICS'14*, 2014. Cited on page 122.

[PZ14c] Thomas Place and Marc Zeitoun. Separation and the successor relation. In preparation, 2014. Cited on pages 84 and 122.

[Rei82] Jan Reiterman. The Birkhoff theorem for finite algebras. *Algebra Universalis*, 14(1):1–10, 1982. Cited on pages x, 3, and 15.

[Reu79] Christophe Reutenauer. Une topologie du monoïde libre. *Semigroup Forum*, 18(1):33–49, 1979. Cited on page 44.

[Reu81] Christophe Reutenauer. Sur mon article: "Une topologie du monoïde libre" [Semigroup Forum **18** (1979), no. 1, 33–49; MR 80j:20075]. *Semigroup Forum*, 22(1):93–95, 1981. Cited on page 44.

[Rho87] John Rhodes. New techniques in global semigroup theory. In *Semigroups and their applications (Chico, Calif., 1986)*, pages 169–181. Reidel, Dordrecht, 1987. Cited on pages 5, 37, and 39.

[RL78] C. R. Reddy and D. W. Loveland. Presburger arithmetic with bounded quantifier alternation. In *STOC'78*, pages 320–325. ACM, 1978. Cited on page 112.

[RZ93] Luis Ribes and Pavel A. Zalesskiĭ. On the profinite topology on a free group. *Bull. London Math. Soc.*, 25:37–43, 1993. Cited on pages xi, 6, 21, 37, 39, and 42.

[RZ13] Lorijn van Rooijen and Marc Zeitoun. The separation problem for regular languages by piecewise testable languages. http://arxiv.org/abs/1303.2143, 2013. Cited on pages 30, 32, 47, 53, 55, 61, 62, and 63.

[Sch65] Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965. Cited on pages x and 3.

[Sch76] Marcel-Paul Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup Forum*, 13:47–75, 1976. Cited on pages 68 and 70.

[Sim72] Imre Simon. Hierarchies of events with dot-depth one. PhD thesis, University of Waterloo, 1972. Cited on pages 47 and 50.

[Sim75] Imre Simon. Piecewise testable events. In *Proc. of the 2nd GI Conf. on Automata Theory and Formal Languages*, pages 214–222. Springer, 1975. Cited on pages x, 3, and 50.

[Sim90] Imre Simon. Factorization forests of finite height. *Th. Comp. Sci.*, 72(1):65 – 94, 1990. Cited on pages 53, 55, and 61.

[Sko31] Thoralf Skolem. Über einige Satzfunktionen in der Arithmetik. Skr. Norske Vid.-Akad., Oslo, Math.-Naturv. Kl. No.7, 1-28 (1931)., 1931. Cited on pages 100, 102, and 116.

[SSMH04] Helmut Seidl, Thomas Schwentick, Anca Muscholl, and Peter Habermehl. Counting in trees for free. In *ICALP'04*, volume 3142 of *Lect. Notes Comp. Sci.*, pages 1136–1149. Springer, 2004. Cited on page 112.

[ST88] Howard Straubing and Denis Thérien. Partially ordered finite monoids and a theorem of I. Simon. *J. Algebra*, 119(2):393–399, 1988. Cited on page 50.

[Ste85] Jacques Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985. Cited on pages x, 3, and 51.

[Ste98] Benjamin Steinberg. On pointlike sets and joins of pseudovarieties. *Internat. J. Algebra Comput.*, 8(2):203–231, 1998. Cited on pages xi, 6, 21, and 84.

[Ste01] Benjamin Steinberg. A delay theorem for pointlikes. *Sem. Forum*, 63(3):281–304, 2001. Cited on pages xi, 6, 19, 21, 84, 91, and 121.

[Str85] Howard Straubing. Finite semigroup varieties of the form $\mathsf{V} * \mathsf{D}$. *J. Pure Appl. Algebra*, 36(1):53–94, 1985. Cited on pages 84 and 91.

[Str94] Howard Straubing. *Finite automata, formal logic, and circuit complexity*. Progress in Theoretical Computer Science. Birkhäuser Boston, Inc., Boston, MA, 1994. Cited on pages 11 and 20.

[SW76] T. G. Szymanski and J. H. Williams. Noncanonical extensions of bottom-up parsing techniques. *SIAM J. Comput.*, 5(2):231–250, 1976. Cited on pages 20, 117, and 119.

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. Cited on page 65.

[Tho82] Wolfgang Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25(3):360–376, 1982. Cited on page 88.

[Tra01a] A. N. Trahtman. An algorithm to verify local threshold testability of deterministic finite automata. In *Automata Implementation*, volume 2214 of *Lect. Notes Comp. Sci.*, pages 164–173. Springer, 2001. Cited on page 88.

[Tra01b] A. N. Trahtman. Piecewise and local threshold testability of DFA. In *Proc. FCT'01*, pages 347–358. Springer, 2001. Cited on page 51.

[TT02] Pascal Tesson and Denis Therien. Diamonds are forever: The variety DA. In *Semigroups, Algorithms, Automata and Languages*, pages 475–500. World Scientific, 2002. Cited on page 68.

[TW85] Denis Thérien and Alex Weiss. Graph congruences and wreath products. *J. Pure Appl. Algebra*, 36:205–215, 1985. Cited on page 88.

[TW98] Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proc. of STOC'98*, pages 234–240. ACM, 1998. Cited on pages 2 and 68.

[VSS05] Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational Horn clauses. In *Proc. of CADE'05*, volume 3632 of *Lect. Notes Comp. Sci.*, pages 337–352, 2005. Cited on page 112.

[WI09] Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FO^2 on words. *Logical Methods in Computer Science*, 5(3), 2009. Cited on page 68.

[Zal72] Yechezkel Zalcstein. Locally testable languages. *Journal of Computer and System Sciences*, 6(2):151–167, 1972. Cited on page 87.

Index

(α, s)-compatible, 90
A, 70
alphabet, 11
alphabet-testable languages, *see* SI
automaton, 12

(B, p)-pattern, 55, 75
$\mathcal{B}\Sigma_1(<)$, 16, 48
block, 89
  prefix block, 89
  suffix block, 89

common pattern, 54
commutative image, 101
complexity
  of $\mathsf{FO}^2(<)$-separation, 82
  of K-separation, 32–33
  of LT-separation, 107–115
  of LTT-separation, 107–115
  of PT-separation, 64–66
  of SI-separation, 30–32
context-free languages, 20, 115–120
counting threshold, 87

d-pattern, 89
DA, *see* $\mathsf{FO}^2(<)$
$\Delta_2(<)$, *see* $\mathsf{FO}^2(<)$
DFA, 12
discriminative power, 19

Ehrenfeucht-Fraïssé games, 69–70
expressive power, 19

factor, 11
factorization pair, 53
factorization tree, 61
first-order logic, *see* $\mathsf{FO}(<)$
$\mathsf{FO}(<)$, 15, 20
$\mathsf{FO}^1(<)$, 16, 29
$\mathsf{FO}^2(<)$, 16, 67–82
$\mathsf{FO}(=, +1)$, 16, 88
free group, 38
free pro-V semigroup, 14, 24

G, 37–45
group languages, *see* G

indistinguishable pairs, 26–29
infix, 11, 85

J, 51
J_1, *see* SI

K, 32–35
k-image, 101
k-loop, 95
k-profile, 85
k-unfolding, 97

ℓ-template, 55
left zero, 32
locally testable languages, 83–120
locally threshold testable languages, 83–120
LSI, 87
LT, *see* locally testable languages
LTT, *see* locally threshold testable languages

NFA, 12

(p, q)-compatible, 90
$\mathcal{P}$-decomposition, 89
$\Pi_2(<)$, 68
p-implementation, 56
Parikh image, *see* commutative image
permutation automaton, 38
piece, 11, 49
piecewise testable languages, 47–66, 68
pointlike sets, 22
  k-pointlike sets, 22

prefix, 11, 85
prefix-testable languages, *see* K
PT, *see* piecewise testable languages

rank, 15
relational morphism, 22

$\Sigma_1(<)$, 48
$\Sigma_2(<)$, 68
semilattice, 29
separable, 18
  L-separable, 19
  $\mathcal{S}$-separable, 18
  V-separable, 19
separation problem, 19
separator, 18
SI, 20, 29–32, 65, 112, 113
suffix, 11, 85
synchronizable, 109
  prefix synchronizable, 109
  suffix synchronizable, 109

$(\vec{u}, \vec{B})$-path, 53
unambiguous languages, *see* $\mathsf{FO}^2(<)$

variety, 14