Characterizing formality

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
M.Sc. Demen Güler
aus Ludwigshafen am Rhein

Tübingen
2018

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	13. März 2019
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Klaus-Jörn Lange
2. Berichterstatter:	Prof. Dr. Michael Kaufmann

Zusammenfassung

Komplexitätsklassen werden durch die quantitative Einschränkung der Ressourcen von Berechnungsmodellen, wie zum Beispiel der Turingmaschine, definiert. Für Familien formaler Sprachen hingegen gibt es keine solch uniforme Beschreibung. Sie werden stattdessen beispielhaft aufgezählt.

Diese Dissertation behandelt Charakterisierungsmöglichkeiten um entscheiden zu können, was eine Menge von Sprachen zu einer Menge *formaler Sprachen* macht. Familien formaler Sprachen, wie beispielsweise die regulären, die kontextfreien und ihre Unterklassen zeigen Eigenschaften auf, die im starken Kontrast zu der von Komplexitätsklassen stehen.

Zwei dieser Eigenschaften formaler Sprachfamilien sind der Abschluss unter dem Schnitt mit regulären Sprachen und das Vorhandensein von Pumping Lemmata, welche zu einer Entscheidbarkeit deren Leerheit führen. Weil Komplexitätsklassen generell kein entscheidbares Leerheitsproblem haben, war ein erster untersuchter Formalitätsbegriff die Entscheidbarkeit der Schnittleerheit mit regulären Sprachen (die Entscheidbarkeit von int_{Reg}). Wir widerlegen int_{Reg} als solches Kriterium indem wir für jede entscheidbare Sprache L eine Sprache L' konstruieren welche von gleicher Komplexität ist aber für die int_{Reg} entscheidbar ist – die Schwierigkeit wird *verborgen*. Da Komplexitätsklassen auf künstliche Weise ein entscheidbares Wortproblem haben, folgt, dass jede Komplexitätsklasse vollständige Probleme enthält, für welche die Schnittleerheit mit regulären Sprachen entscheidbar ist.

Ein Zwischenergebnis ist die Entscheidbarkeit von int_{Reg} für die Menge der wahren quantifizierten Booleschen Formeln. Alle bekannten Familien formaler Sprachen sind in NP enthalten. Daher ist dies eine *natürliche* Sprache, die (wahrscheinlich) außerhalb von NP liegt für die die Entscheidbarkeit von int_{Reg} gelten würde.

Wir führen den Begriff der *Protokollsprache* ein, welche das Verhalten einer Datenstruktur beschreibt, das einem Modell formaler Sprachen zugrunde liegt. Sie werden mit einem Fragment der Prädikatenlogik der zweiten Stufe definiert, in der jede binäre Variable eindeutig durch ein Wort einer Protokollsprache bestimmt wird und jeder Buchstabe eine definierte Unterstruktur des Wortes impliziert. Jedes Wort kann als Pfad gesehen werden, in welchem die Buchstaben die Knoten bilden, die von der Nachfolgerrelation verbunden werden. Durch die Betrachtung der binären Variablen als zusätzliche Kanten spannt jedes Wort einer Protokollsprache einen eindeutig definierten Graphen auf. Diese Familie von Graphen kann durch eine definierte Menge von *Tiles*, kanten- und notenmarkierter Graphen, gekachelt und

somit erkannt werden.

Eine zusätzliche numerische Einschränkung an Mindest- und Höchstzahlen von verwendeten Tiletypen liefert Kürzbarkeitsargumente (*shrinking*) für Protokollsprachen: Wenn ein Wort w einer Protokollsprache eine Mindestlänge überschreitet, so dass der entsprechende Graph zu dem Wort eine festgelegte Anzahl an Kacheln verwendet, kann man aus w konstruktiv ein kürzeres Wort w' aus der Protokollsprache erstellen.

Wir definieren logische Erweiterungen von Protokollsprachen, welche durch die *Verundung* von Prädikatenlogik der ersten Stufe oder Monadischer Logik der zweiten Stufe entstehen und untersuchen diese bezüglich der Trio-Operationen.

Wir stellen Protokollsprachen für die regulären-, kontextfreien- und die Indexsprachen vor. Für die Protokollsprachen der regulären und kontextfreien zeigen wir, dass sie als Generatoren der jeweiligen Sprachfamilie dienen. Abschließend zeigen wir, dass die Leerheit von Protokollsprachen entscheidbar ist.

Abstract

Complexity classes are defined by quantitative restrictions of resources available to a computational model, like for instance the Turing machine. Contrarily, there is no obvious commonality in the definition of families of formal languages – instead they are described by example.

This thesis is about the characterization of what makes a set of languages a family of *formal* languages. Families of formal languages, like for example the regular, context-free languages and their sub-families exhibit properties that are contrasted by the ones of complexity classes.

Two of the properties families of formal languages seem to have is closure of intersection with regular languages, another is the existence of pumping or iteration arguments which yield the decidability of the emptiness. Complexity classes do not generally have a decidable emptiness, which lead us to a first candidate for the notion of *formality* – the decidability of the emptiness of regular intersection (int_{Reg}). We refute the decidability of int_{Reg} as a criterion by *hiding* the difficulty of deciding the emptiness of regular intersection: We show that for every decidable language L there is a language L' of essentially the same complexity such that $int_{\mathsf{Reg}}(L')$ is decidable. This implies that every complexity class contains complete languages for which the emptiness of regular intersection is decidable.

An intermediate result we show is that the set of true quantified Boolean formulae has a decidable emptiness of regular intersection. As the known families of formal languages are all contained in NP, this yields a language (probably) outside of NP for which int_{Reg} is decidable, which additionally is a *natural* language in contrast to the *artificial* ones obtained by the *hiding* process.

We introduce the notion of *protocol languages* which capture in some sense the behavior of a data-structure underlying the model of a formal language. They are defined in a fragment of second order logic, where the second order variables are uniquely determined by each word in the language and each letter implies a determined sub-structure of a word. Viewing the letters of a word as vertices and the successor as edges between them, each word can be seen as a path. The binary second order variables can be viewed as additional edges between word positions. Therefore, each word in a protocol language defines some unique graph. These graphs can be recognized by covering them with a predefined set of *tiles* which are node and edge-labeld graphs. Additional numerical constraints on the amount of each tile-type yields *shrinking*-arguments for protocol languages. If a word w in a protocol language exceeds a

certain length such that the numerical constraints are *(over-)satisfied*, one can constuctively generate a shorter word w' from w that is also contained in the protocol language.

We define logical extensions of protocol languages by allowing the conjunction of additional first order or monadic second order definable formulae and analyze the extensions in regard to trio operations.

Protocol languages for the regular, context-free and indexed languages are exhibited – for the first two we give protocol languages which act as *generators* for the respective family of formal languages. Finally, we show that the emptiness of protocol languages is decidable.

Contents

Acknowledgements

Now for the third time my foremost thanks go Klaus-Jörn Lange for his guidance, his patience with me and the freedom I had while working with him. I thank Michael Kaufmann for supervising me and offering to review this thesis. I am deeply indebted to my friend and colleague Silke Czarnetzki for carefully proofreading this dissertation and aiding me over the years with my mathematical shortcomings. My thanks go to Andreas Krebs for the fruitful discussions and advice he gave me. Michaël Cadilhac and Charles Paperman were role-models in many aspects – it was an enriching pleasure having them around. I also want to thank Petra Wolf for working with me on the article which became part of this thesis. I am deeply grateful to Mariechristin Hähnel and my family for their emotional and moral support throughout. Finally, I thank my friends and housemates (many in double staffing) for supporting me and taking over many of my responsibilities over the last year.

CHAPTER 1

Introduction

Formal language theory and complexity theory are two of the fundamental research areas of theoretical computer science and are densely interwoven subjects. The natural connection between both of these topics are *decision problems*. In formal language theory this question turns up when we want to know whether some *word* is contained in a *language*. Problems in complexity theory are often of combinatorial nature which are turned into decision problems and encoded such that they can be viewed as languages.

Complexity classes are sets of languages which are defined by restricting resources a computational model might use during its computation. This could be for instance time or space for a Turing machine, or the size or depth of a circuit family where the limits are most often set as a function dependent on the input-size.

For families of formal languages, as for instance the regular, context-free or indexed languages and their sub-classes, there is no such uniform definition. Instead, they are addressed by example or by common properties like decidabilities and closures. Many decision problems such as the emptiness are decidable for families of formal languages. This decidability often is due to the existence of pumping/shrinking and iteration lemmata, which give bounds on the word-lengths that have to be considered when testing for emptiness.

The resource bounding artificially gives the languages in a complexity class the a decidable word problem. On the other hand, deciding any other none-trivial property for a complexity class quickly yield undecidability results with typical *Rice* arguments.

The following table shows the difference of complexity classes and families of formal languages in regard to their closure properties.

Closure under	families of formal languages	complexity classes
intersection	×	✓
union	✓	✓
many-one reductions	×	✓
homomorphisms	✓	×
Kleene star	✓	×

While there is a *gap* between the closure regarding union and intersection for (non-deterministic) families of formal languages, complexity classes do not discern between these operations. Both are closed under inverse morphisms, though complexity classes are in contrast to families of formal languages not closed under forward morphisms – in particular not regarding erasing morphisms.

There are examples of language families – the regular, context-free and indexed languages – that share many common properties with each other which are in contrast to the ones of complexity classes. We call these *families of formal languages* and propose a more precise notion of what makes a set of languages formal, beside listing examples.

A first candidate of such a notion was formed with properties families of formal languages typically possess and complexity classes generally lack – the closure under intersection with regular languages together with a decidability of the emptiness. We show that the decidability of the emptiness of regular intersection is not an adequate characterization for the *formality* of a language class, by showing that every complexity class contains complete languages where this property is decidable.

In a more constructive approach we aim to describe families of formal languages with a logical framework. Thomas [Tho91] considers MSO-definable structures (e.g. words, graphs and infinite words) and shows that they are the homomorphic images of the respective first order definable structures. We focus on words, but consider also non-regular languages. As regular languages are the ones definable with $\mathsf{MSO}[S]$, that is monadic second order logic with the successor relation we need a stronger logic fragment to describe non-regular languages. Lautemann et al. [LST94] give a logical characterization of the context-free languages. They use first order formulae which might adopt a binary second order variable like a relational symbol, which is semantically constrained to describe solely *nestings* on word-positions.

We define a logic fragment which allows first order formulae to use (but not to quantify) multiple binary second order variables, which are semantically constrained to behave like total functions on the word positions and are uniquely defined for each word. Inside this fragment we define *auto-generative* (word-)languages. In these languages each letter defines a certain sub-structure in a word and vice versa each sub-structure implies a definite letter. The uniquely determined second order variables define a graph for each word of an auto-generative language. These graphs can be recognized via *tiling*, which can be used to define shrinking arguments for auto-generative languages. We consider logical extensions of such languages and study them regarding trio operations and show that there are auto-generative *deviation languages* for regular, context-free and indexed languages.

Examples for families of formal languages form an inductively defined canonical hierarchy – the OI-hierarchy [Dam82]. The regular languages are the base level: $\mathsf{OI}^0 = \mathsf{Reg}$. The level $i+1$ is the *yield* of trees where paths are words of languages (called δ- or delta-operation) in the deterministic version of OI^i, named DOI^i, i.e. $\delta(\mathsf{DOI}^i) = \mathsf{OI}^{i+1}$ [Eng02]. For instance $\mathsf{OI}^1 = \delta(\mathsf{DReg}) = \delta(\mathsf{Reg}) = \mathsf{CFL}$ and $\mathsf{OI}^2 = \delta(\mathsf{DOI}^1) = \delta(\mathsf{DCFL}) = \mathsf{Index}$ which correspond to Fischer's [Fis68] OI macro languages. Note, that the OI-hierarchy is a true hierarchy.

The word problem is a natural connection between families of formal languages and complexity

theory [Lan96]. In the following we give a few examples of families of formal languages being complete in complexity classes.

Families of formal languages	complete in complexity class
Reg	NC^1
VPL	NC^1
Lin	NL
NOCA	NL
CFL	SAC^1
Index	NP
OI-hierarchy	NP

For non-deterministic families there seems to be a *special* relation to complexity classes: Krebs and Lange [KL12a] introduced the notion of *dense completeness*. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two sets of languages. $\mathcal{C}_1$ is called *densely complete* in $\mathcal{C}_2$ if $\mathcal{C}_1 \subseteq \mathcal{C}_2$ and for all languages L in $\mathcal{C}_2$ there is a language $L' \in \mathcal{C}_1$ such that L and L' are (DLOGTIME-uniform) AC^0-many-one equivalent, i.e. L and L' are reducible to each other. They showed that the context-free languages are densely complete in SAC^1, that the one-counter languages are densely complete in NL and that the indexed languages are densely complete in NP. It was also shown, that the regular languages and VPL (where non-determinism equals determinism) are not densely complete in NC^1.

The decidability of the emptiness of regular intersection

All known families of formal languages are closed under the intersection with regular languages. Due to shrinking and iteration arguments the emptiness of these language families is typically decidable. This implies that the following property is decidable for families of formal languages:

Definition 1.0.1 (int_{Reg})**.** Let $int_{\mathsf{Reg}}(L)$ be the problem of deciding, given a finite automaton, whether its language has a non-empty intersection with L. For a family of languages $\mathcal{L}$ we say $int_{\mathsf{Reg}}(\mathcal{L})$ is decidable if $int_{\mathsf{Reg}}(L)$ is decidable for every $L \in \mathcal{L}$.

In the first part of this thesis we show that the decidability of int_{Reg} is not an adequate notion of *formality*.

Van Leeuwen [VL75] showed that SAT (in *variable-free encoding*) is contained in ET0L. Since the emptiness is decidable in ET0L and it is closed under intersection with regular sets we have that $int_{\mathsf{Reg}}(\text{SAT})$ is decidable. We proved that $int_{\mathsf{Reg}}(\text{SAT})$ is also decidable when Boolean formulae are encoded such that variable names are denoted as unary strings – an encoding previously used by Stockmeyer [Sto76].

All known families of formal languages are contained in NP. We therefore, considered the complementary problem to SAT – The set of all propositional formulae (in 3-CNF) that are tautologies TAUT, which is $\Pi_1^P = \mathsf{CoNP}$-complete. To strengthen our notion of *formality*, we expected to show that $int_{\mathsf{Reg}}(\text{TAUT})$ was undecidable.

These attempts were unsuccessful as we proved the converse. In fact we were able show that the for the set of (appropriately encoded) true quantified Boolean formulae the emptiness of regular intersection is decidable. This result was published in [GKLW18].

This result means that if given a regular set of encoded quantified Boolean formulae it is decidable whether one of them evaluates to true. In our proof we make use of pumping

arguments which assure that there are repetitions of certain sub-words in words accepted by the automaton recognizing the regular set. We *reduce* an infinite regular set to a finite number of formulae from which we search for true quantified Boolean formulae.

This result implies that there are complete languages for all levels of the polynomial hierarchy and PSPACE for which int_{Reg} is decidable.

We use a technique called *hiding* which was inspired by [KL12a] to show the dense completeness results. With this we were able to show that for every decidable language L there is a language L' such that L and L' are AC^0-many-one equivalent and $int_{\mathsf{Reg}}(L')$ is decidable. Since complexity class are closed under AC^0-reductions each of them contains complete problems for which int_{Reg} is decidable. These results contradict the decidability of the emptiness of regular intersection as formality criterion.

We further show that if $int_{\mathsf{Reg}}(L)$ is decidable for some language L then the int_{Reg} is decidable for the full AFL generated by L. This means every complexity class contains complete problems for which their full AFLs have a decidable emptiness of regular intersection.

With this we conclude that the decidability of the emptiness of regular intersection seems a necessary property of families of formal languages, but is not sufficient to characterize them precisely.

Protocol languages

Engelfriet [Eng14] portrayed the derivations of context-free grammars as "string-generating [...] non-deterministic recursive programs": Each non-terminal is a subroutine that is built according to the rules of the grammar. Depending on the rules they will recursively call other routines or write letters on the output. For example if a context-free grammar contains the rules $B \to bCD$ and $B \to a$ for the non-terminal B then the sub-routine of B would be the schematic procedure denoted in Algorithm 1.

Algorithm 1: Schematic procedure for the non-terminal B

begin
  `write`(b) `call`(C) `call`(D);
end
or
begin
  `write`(a);
end

This view-point of procedural generation can be adapted for some other families of formal languages. The languages in the OI-hierarchy could be procedurally described by *passing* parameters when calling subroutines. In case of indexed languages [Aho68] this could be a sequence of indices, for higher classes in the OI-hierarchy one could pass stacks of stacks ... of stacks. Fischer introduced the OI and IO macro languages [Fis68] which apply this idea.

In case of the previously mentioned context-free languages the *data-structure* or *access pattern* could be viewed as a stack of subroutines that have yet to be processed. In the beginning of the derivation process this stack would only contain the start symbol of the grammar. In each derivation step the top-most stack symbol would be popped and the corresponding procedure would be executed. Each call in a procedure would push the respective symbols onto the stack. The derivation is finished once the stack is emptied.

To analyze this structural behavior of a model we have to look at the operations or instructions it might be equipped with.

If one would for example consider push-down automata, the data-structure used by this model is the stack. During the computation, symbols are pushed onto and popped from the stack which is emptied when the PDA accepts the word. Thus, when *recording* the data-structure during the computation of an accepted word the resulting stack *protocol* would be a well-matched sequence of push and pop-operations – and hence a Dyck word.

Exemplary, we consider *derivation languages* of (context-free) grammars which are the sequences of all correct left-derivations of words generated by the grammar. Derivation languages have been considered before and are also known as Szilard languages [DPS79, Mäk84].

We begin the analysis with derivation languages of the family of the context-free languages and consider grammars in Chomsky normal form and Greibach normal form. One of the key languages we consider in this thesis is the Łukasievicz language Ł which is generated by the grammar with the two rules $S \to aSS$ and $S \to b$. We show that Ł is the derivation language of the grammar in Chomsky normal form with a single non-terminal and one terminal symbol.

We give a logic framework which is applicable to represent the structural behavior of models, as for instance derivation languages. The derivation languages of non-regular grammars are non-regular themself.

Büchi [Büc60], Elgot [Elg61] and Trakhtenbrot [Tra61] showed that the regular languages are precisely the sets of words definable with monadic second order logic with the successor-relation ($\mathsf{MSO}[S]$). Therefore, $\mathsf{MSO}[S]$ will not be suffice to define protocol languages. For non-regular languages $\mathsf{MSO}[S]$ is therefore too weak. Existential second order logic captures the languages definable in NP [Fag73, Fag74], though certain fragments of existential second order logic only capture the same languages as $\mathsf{MSO}[S]$ [EGG00]. Lautemann et al. [LST94] gave a logical characterization of the context-free languages: They consider a fragment of existential second order logic where they only allow *binary* second order variables. Furthermore, these binary second order variables are semantically constrained such that they only describe *matchings* or *nestings*. They define the class of first-order formulae $\exists\mathcal{M}\mathsf{FO}[(Q_a)_{a\in A}, <]$ where $\mathcal{M}$ is the set of second order variables describing matchings which are used by the first order formulae as atomic formulae. The fragment $\exists\mathcal{M}\mathsf{FO}[(Q_a)_{a\in A}, <]$ is not well suited to describe the deterministic nature of for instance derivation languages. We build upon the ideas of the logic fragment introduced by Lautemann et al. by allowing more than one binary second order variable and not restricting them to define solely nestings. Instead, we require the binary second order variables to be unambiguously determined by every word that models the formula. This means that each word implicitly defines a distinct structure on itself. We call the fragment of logic formulae $\mathsf{unEx}k\mathsf{FO}[S]$ where k is the number of uniquely quantified and constrained binary second order variables and S is the successor relation. $\mathsf{unEx}k\mathsf{FO}[S]$ contains first order formulae with the signature over letter predicates, the successor relation and the k binary second order relations.

We define *auto-generative languages* which are word languages defined by formulae inside $\mathsf{unEx}k\mathsf{FO}[S]$. Each letter γ in an auto-generative language determines a definite sub-structure in word and vice versa the structure of the binary second order relations uniquely implies which letter needs to occur. This sub-structure is defined by a sub-formula ψ_γ which only uses *positional information* (i.e. binary second order variables and the successor relation) and can be used as a *substitute* for the letter predicate.

The Łukasievicz is an auto-generative language in $\mathsf{unEx1FO}[S]$, i.e. definable with one additional binary second order variable and the successor relation.

To decide the emptiness of (word-)automata with auxiliary storage, Madhusudan and Parlato [MP11] propose a method to transform the behavior of the *word recognizer* into a *graph recognition* problem which is then solved via *tiling*: A graph is accepted if a set of patterns can be *coherently* mapped onto the graph.

Each word of an auto-generative language defines a graph where the nodes are the positions of the word and the edges are given by the successor relation and the uniquely determined second order relations. To decide the emptiness of auto-generative (word-) languages we define tiling on the graphs families generated by them.

Madhusudan and Parlato [MP11] use a simplified version of the tiling more generally defined by Thomas [Tho91], who additionally *counts* (up to some constant) the number of instances of tiles used when defining the acceptance criterion. We make use of this counting to define *shrinking* arguments on auto-generative languages. For instance, one might imagine that one tile covers a *push* and *pop* pair on a sequence of stack operations and that removing this matching pair will yield a valid sequence of operations on a stack. Therefore, an arbitrary amount of these tiles may be used. On the other hand, such a sequence of operations must contain precisely one transition which pops the *bottom of the stack* (at the end of the computation).

We call languages which are auto-generative which can be tiled with shrinking arguments *protocol languages*.

We consider logical extensions of protocol languages. For a protocol language P, first-order formulae, monadic second order formulae and existential monadic second order formulae which are in conjunction with a formula that ensures that the structure of a word is *in accord* with P yield the classes of languages $\mathsf{FO}[P]$, $\mathsf{MSO}[P]$ and $\mathsf{EMSO}[P]$. Additionally, we define the logic classes $\mathsf{strongFO}[P]$, $\mathsf{strongMSO}[P]$ and $\mathsf{strongEMSO}[P]$ as the classes where the additional formulae might use the binary second order variables defined by the formula of the protocol language as atomic sub-formulae.

We show that $\mathsf{EMSO}[P]$, $\mathsf{MSO}[P]$, $\mathsf{strongMSO}[P]$ and $\mathsf{strongEMSO}[P]$ are closed under length-preserving morphisms. We compare these extensions with each other and show that the languages in $\mathsf{EMSO}[P]$ coincide with the length-preserving morphic images of languages in $\mathsf{FO}[P]$ (and the analogous relation for $\mathsf{strongEMSO}[P]$ and $\mathsf{strongFO}[P]$).

Protocol languages only capture *deterministic* and *visible* languages. The determinism is preserved for languages in $\mathsf{FO}[P]$, while homomorphisms applied to languages in $\mathsf{FO}[P]$ may introduce non-determinism. Therefore, $\mathsf{EMSO}[P]$ and $\mathsf{MSO}[P]$ contain non-deterministic languages. The *strong* extensions of a protocol language P, which have access to the binary second order variables, are outside of the trio of P.

We give several examples for protocol languages in various families of formal languages. The regular languages are shown to coincide with $\mathsf{MSO}[a^*]$ and the context-free languages with $\mathsf{strongMSO}[\mathcal{G}_2]$ where $\mathcal{G}_2$ is a protocol language which is the derivation language of some special grammar in Greibach normal form. We show that a *tagged* version of the Dyck language with k pairs of parenthesis is in $\mathsf{strongFO}[\hat{\mathbb{D}}_1]$ where $\hat{\mathbb{D}}_1$ is the *tagged* version of the Dyck language with one pair of parenthesis. This result implies that the *strong* extensions of a protocol language P are not contained in the trio of P.

Finally, we show how to decide the emptiness and int_{Reg} of protocol languages by the application of tiling and shrinking arguments.

Protocol languages seem a promising candidate to define data-structural *skeletons* of families of formal languages. Nevertheless, the conceptualization of protocol languages have not yet completely matured and definitions might still be subject to change.

Also, future work must be put into the analysis of the *robustness* of protocol languages, in the sense that they might be used as a tool to discriminate families of formal languages from complexity classes. Results of descriptive complexity give us upper bounds (from a complexity perspective) for languages which might be protocol languages. Fagin's theorem [Fag73, Fag74] states that the languages definable with existential second order logic precisely capture NP. This was extended by Stockmeyer [Sto76] who showed that the languages in the Polynomial Hierarchy (PH) are the ones definable with (arbitrarily quantified) second order logic. Since protocol languages are defined with a fragment of second order, finding protocol languages which are considered *outside* of PH would imply the collapse of complexity classes and therefore seems rather unlikely.

Structure of this thesis

This thesis aims to find a characterization of families of formal languages beyond exemplary enumeration.

We begin by fixing our notation and defining the language generators/descriptors we use formally in our proofs, such as grammars, finite automata and logic on words.

In the third chapter we work on the decidability of the emptiness of regular intersection. We show that even though it might be a property of families of formal languages, every complexity class contains complete languages for which the emptiness of the regular intersection is decidable.

Subsequently, in Chapter 4 we give our notion of protocol languages. We analyze logical extensions, their relation to each other and regarding to trio operations. The Łukasievicz language showcases the analysis and the development of protocol languages. We propose logical extensions of protocol languages and give protocol languages for some important families of formal languages, such as the regular, context-free and indexed languages. Using tiling arguments we show how to decide the emptiness of protocol languages.

Finally, in Chapter 5 we discuss our results and state open questions for further research on this subject.

CHAPTER 2

Preliminaries

In this chapter we fix our notation and give a brief overview of the definitions of formal language theory. We point the reader to Hopcroft and Ullman [HU79], where most of the language theoretic concepts we use are rigidly defined. For the definitions for *logic on words* we stick to the notation of Straubing [Str94]. For notions on complexity theoretic basics we direct the reader to the standard textbooks as [Pap94, AB09, Koz12]

2.1 Essentials

By $\mathbb{N}$ we denote the *natural numbers* excluding 0, by $\mathbb{N}_0$ for $\mathbb{N} \cup \{0\}$ and by $\mathbb{Z}$ the set of *integers*. For $n \in \mathbb{N}$ we write $[n]$ for the set $\{1, \ldots, n\}$. If S is some set, we denote the *power set* $\{I \mid I \subseteq S\}$ *of* S by 2^S or $\mathcal{P}(S)$. For a binary relation R by R^* we denote the *reflexive, transitive closure* of R. We denote the Boolean values *true* and and *false* by 1 and 0, respectively.

2.2 Words and languages

An *alphabet* is a finite set A of symbols, a *word* is a finite sequence of elements of A. A *language* is a set of words. We denote the *empty word* by λ. Let $w = w_1 \ldots w_n$ be a word. Then let $|w| := n$ denote the *length of a word*. By $w[i]$ we denote the i-th letter of w. For $1 \leq i, j \leq n$ we call $w_i w_{i+1} \ldots w_j$ a *factor* or *sub-string* of w and denote it by $w[i, j]$. If $i > j$ then let $w[i, j]$ denote λ. Let A^* denote the set of all languages over the alphabet A. If L_1, L_2 are languages of strings in A^* let $L_1 L_2$ denote $\{uv \mid u \in L_1 \wedge v \in L_2\}$. For a language L we define $L^0 := \{\lambda\}$ and $L^i = LL^{i-1}$ for $i \in \mathbb{N}$. We define the *Kleene star* or *Kleene closure* of L as $L^* := \bigcup_{i=0}^{\infty} L^i$ and by $L^+ := \bigcup_{i=1}^{\infty} L^i$.

2.3 Finitely describing infinite languages

Describing finite languages is straight forward – simply enumerate all words in the language. Of course this approach is not possible for languages with infinitely many words. From the plethora of ways to finitely define languages we focus on grammars, finite automata, Turing machines and logic on words.

By Reg, DCFL, CFL, Index, Loc, LTT we denote the the regular , (deterministic) context-free, indexed, local and locally threshold testable languages respectively. By P, NP we denote the classes of languages recognized by polynomial time bounded deterministic/non-deterministic Turing machines. Let PH denote the Polynomial Hierarchy and PSPACE class of languages recognized by polynomial space bounded deterministic Turing machines.

In the following we give precise definitions of the language descriptors we use in a formal manner.

2.3.1 Finite automata

A *non-deterministic finite automaton* (NFA) is a 5-tuple $M = (Q, A, \delta, Q_0, F)$ where Q is a finite set of *states*, A is an alphabet, $\delta \subseteq Q \times A \times Q$ is a transition relation, $Q_0 \subseteq Q$ is a set of *initial states* and $F \subseteq Q$ is a set of final states. Let $\hat{\delta}\colon Q \times A^* \to Q$ be defined recursively:

1. For $q \in Q$ let $\hat{\delta}(q, \lambda) = q$
2. For $w \in A$ and $a \in A$ let $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$

We define the language of M as $L(M) = \{w \in A^* \mid \exists q_0 \in Q_0, q_f \in F : \hat{\delta}(w, q_0) = q_f\}$. We call M a *deterministic finite automaton* (DFA) if $\delta\colon Q \times A$ is a total function and there is a single initial state q_0.

2.3.2 Regular expressions

Let A be an alphabet. Regular expressions are defined recursively. Let $L(\alpha)$ denote the language defined by a regular expression α. $\emptyset$ is a regular expression with $L(\emptyset) = \emptyset$. λ is a regular expression $L(\lambda) = \{\lambda\}$. For all $a \in A$, a is a regular expression with $L(a) = \{a\}$. If α and β are regular expression, then so is

1. $\alpha\beta$ with $L(\alpha\beta) := L(\alpha)L(\beta)$,
2. $(\alpha \mid \beta)$ with $L((\alpha \mid \beta)) := L(\alpha) \cup L(\beta)$,
3. $(\alpha)^*$ with $L((\alpha)^*) = L(\alpha)^*$.

We sometimes do not differentiate between a regular expression and its language and use notations like $w \in \alpha$ instead of $w \in L(\alpha)$.

2.3.3 Grammars

We give a very brief introduction into grammars. For a more detailed view we point the reader to Hopcroft and Ullman [HU79].

A *Chomsky grammar* or simply *grammar* is a tuple $G = (N, A, P, S)$ where N is a finite set of *non-terminals*, A is an alphabet, $P \subseteq (A \cup N)^* N (A \cup N)^* \to (A \cup N)^*$ is a finite set of *rules* or *productions* and $S \in N$ is the *start symbol*. For $x, y \in (A \cup N)^*$ we define the binary relation $\Rightarrow_G$ for words over $(A \cup N)$ as

$$x \Rightarrow_G y \iff \exists (u \to v) \in P : \exists s, t \in (A \cup V)^* : x = sut \wedge y = svt$$

Let $L(G) = \{w \in A^* \mid S \Rightarrow_G^* w\}$ be the language defined by G. We call a grammar $G = (N, A, P, S)$ *context-sensitive* if for all $(u \to v) \in P$ we have $|u| \leq |v|$. A context-sensitive grammar is called *context-free* if for all $(u \to v) \in P$ we have $|u| = 1$. A context-free grammar is *regular* if $P \subseteq N \to (V \cup VA)$.

We call the set of languages recognized by context-sensitive grammars the *context-sensitive languages* or CS, the languages recognized by context-free grammars *context free languages* or CFL and the languages recognized by regular grammars *regular languages* or Reg.

Normal forms for context-free languages

There are several *normal forms* for context-free grammars, i.e. constraints to how the production rules might be formed. We make use of the *Chomsky normal form (CNF)* and the *Greibach normal form (GNF)*. For every context-free language there are both grammars in CNF and GNF.

Chomsky normal form A context-free grammar $G = (V, A, P, S)$ is said to be in Chomsky normal form if all rules are of the form $B \to CD$ or $B \to a$ for $B, C, D \in V$ and $a \in A$.

Greibach normal form A context-free grammar $G = (V, A, P, S)$ is said to be in Greibach normal form if all rules are of the form $B \to aB_1, \ldots, B_k$ for $k \geq 0$ with $B_1 \ldots, B_k$ and $a \in A$.

Indexed grammars

In addition we consider *indexed grammars* introduced by Aho [Aho68] which are natural extensions of context-free grammars where non-terminals are equipped with a stack of indices. We give the formal definition following [HU79].

An *indexed grammar* is a tuple $G = (V, A, F, P, S)$ where V is a finite set of non-terminals, A is an alphabet, F is a finite set of *indices*, S is the start symbol and P is a finite set of rules of the form

1. $B \to \alpha$
2. $B \to Cf$ (*index producing rule*)
3. $Bf \to \alpha$ (*index consuming rule*)

where $B, C \in V$, $f \in F$ and $\alpha \in (V \cup A)^*$. Non-terminals are followed by (possibly empty) sequence of indices. The derivation relation $\Rightarrow_G$ on strings in $(VF^* \cup A)^*$ is defined as follows. Let $s, t \in (VF^* \cup T)^*$, $\delta \in F^*$ and $X_i \in (V \cup A)$

1. If $B \to X_1 X_2 \cdots X_k$ is a production of form 1.) then

$$sB\delta t \Rightarrow_G sX_1\delta_1 X_2\delta_2 \cdots X_k\delta_k t \ ,$$

where

$$\delta_i = \begin{cases} \delta & \text{if } X_i \in V \ , \\ \lambda & \text{if } X_i \in A \ . \end{cases}$$

This means that the index of a non-terminal is copied over to all the non-terminals it is derived to.

2. If $B \Rightarrow Bf$ is an index producing rule then

$$sB\delta t \Rightarrow_G sCf\delta t \ ,$$

that means f is set to be the top-most symbol on C's stack.

3. If $B \to X_1 X_2 \cdots X_k$ is an index consuming rule than

$$sBf\delta t \Rightarrow_G sX_1\delta_1 X_2\delta_2 \cdots X_k\delta_k t \ ,$$

where

$$\delta_i = \begin{cases} \delta & \text{if } X_i \in V \ , \\ \lambda & \text{if } X_i \in A \ . \end{cases}$$

This means that all non-terminals A is derived into inherit A's stack without its top-most symbol f.

We define $L(G) = \{w \in A^* \mid S \Rightarrow_G^* w\}$ as the language defined by the indexed grammar G.

Normal form Let Index be the family of formal languages which are defined by indexed grammars. For each $L \in$ Index there is an indexed grammar $G = (V, A, F, P, S)$ where the rules in P are of the form

1. $B \to CD$
2. $B \to a$
3. $B \to Cf$
4. $Bf \to C$

for $B, C, D \in V$, $a \in A$ and $f \in F$.

2.3.4 Logic on words

We use formal logic to define languages. To this end, we construct formulae that argue over the realm of positions in a word and use *predicates* that state that a position has a certain letter. For example the predicate $Q_a(x)$ will be *true* if the position x will be the letter a. The formula $\phi = \exists x \forall y : x \leq y \wedge Q_a(x)$ will evaluate to true for all words for which there is a position x that is smaller than all other positions and there is an a on position x. This means the formula finitely defines the language of all words that begin with the letter a. For a comprehensive elaboration we direct the reader to [Str94]. We will first define the syntax of logical formulae and afterwards present semantics.

Syntax

Formulae are built from *variables*, *relational symbols*, *letter predicates*, *Boolean connectives* and *quantifiers*.

First order variables A first order variable will reference a position in a word and will typically be denoted by a symbol $x, y, x_1, \ldots, x_n$.

We call a variable *bounded* if it is referenced by a quantifier. Otherwise, it is called *free*. A logical formula without free variables is called *sentence*.

relational symbol A *relational symbol* is a symbol R^l representing an l-ary relation on word-positions. If $x_1, \ldots, x_l$ are first order variables

$$R^l(x_1, \ldots, x_l)$$

is an *atomic formula*. The formula is true iff the positions $(x_1, \ldots, x_l) \in R^l$, i.e. we do not discern relational symbols and the relation they represent. Throughout, we only make use of binary relational symbols.

Letter predicates For a letter $a \in A$ and a first order variable x

$$Q_a(x)$$

is an atomic formula and true iff the word on position x has the letter a.

Monadic second order variables A *monadic second order variable* is a set of first order variables and we will typically represent by symbols X, Y, A_y. If x is a first order variable and X is a monadic second order variable then

$$x \in X$$

is an *atomic formula*. The formula is true iff x is contained in the set represented by X.

Binary second order variables A *binary second order variable* is a binary relation on positions we will typically represent them by symbols $N, N_1, \ldots, N_k$. If x and y are first order variables and N is a binary second order variable then

$$(x, y) \in N$$

is an *atomic formula*. The formula is true iff the tuple (x, y) is in binary relation N.

Boolean operations Formulae are defined inductively. Every atomic formula is a formula. If ϕ and ψ are formulae, then so are

$$\begin{aligned} \phi \wedge \psi \ , \\ \phi \vee \psi \ , \\ \neg\phi \ , \end{aligned}$$

where $\wedge, \vee$ and $\neg$ represent the Boolean operation *and*, *or* and *negation*, respectively.

Quantification If ϕ is a formula and x is a first order variable then *existentially* or *universally quantifying* x in ϕ, denoted as

$$\exists x\phi \ ,$$
$$\forall x\phi \ ,$$

yields a formula. Thereby, every occurrence of x in ϕ is *bound* by the quantifier. Analogously, if X is a monadic second order variable (or N is a binary second order variable) then *existentially* or *universally quantifying* X (and N, respectively) in ϕ, denoted as

$$\exists X\phi \ ,$$
$$\forall Y\phi \ ,$$
$$\exists N\phi \ ,$$
$$\forall N\phi \ ,$$

yields a formula. Even though we defined how to universally quantify a second order variable we only make use of them to define *unique existential quantification of a sub-formula*:

By $!\exists N\phi$ we obviate the formula $\exists N\phi(N) \wedge \forall N' : (N' \neq N) \Rightarrow \neg\phi(N')$.

Semantics

The previously defined logical formulae include ones with free variables, which by itself will not precisely describe sets of words. Therefore, we need to consider some form of an extension of an alphabet called $\mathcal{V}$-structure. We use an inductive way to define $\mathcal{V}$-structures as in [Sil19]. Let A be an alphabet and $n \in \mathbb{N}$ and $a_i \in A$ for $i = 1, \ldots, n$.

- Then, $w = (a_1, \emptyset)(a_2, \emptyset)\cdots(a_n, \emptyset)$ is a $\mathcal{V}$-structure.
- Let $\mathcal{V}$ be a finite set of first order variables and $(a_1, S_1)(a_2, S_2)\cdots(a_n, S_n)$ be a $\mathcal{V}$-structure with $S_1, \ldots, S_n \subseteq \mathcal{V}$ and let $x \in \mathcal{V}$ be a first order variable not contained in $\mathcal{V}$. Then for each $i = 1, \ldots, n$ the word

  $$w_{i=x} = (a_1, S_1)(a_2, S_2)\cdots(a_i, S_i \cup \{x\})\cdots(a_n, S_n)$$

  is a $\mathcal{V}$-structure.

Let $\mathcal{V}_1$ be a set of fist order variables, let $\mathcal{V}_2$ be a set of monadic second order variables and let $\mathcal{V}_3$ be a set of binary second order variables. Then a $(\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3)$-structure over the alphabet A is a word in $(A \times 2^{\mathcal{V}_1} \times 2^{\mathcal{V}_2} \times 2^{\mathcal{V}_3})$:

$$w = (a_1, S_1, T_1, U_1)(a_2, S_2, T_2, U_2)\cdots(a_n, S_n, T_n, U_n) \ ,$$

such that $(a_1, S_1)(a_2, S_2)\cdots(a_n, S_n)$ is a $\mathcal{V}$-structure.

Let ϕ be a formula with the set of free first order variables $\mathcal{V}_1$, free monadic second order variables $\mathcal{V}_2$, free binary second order variables $\mathcal{V}_3$ and let w be a $(\mathcal{V}_1, \mathcal{V}_1, \mathcal{V}_3)$-structure.

We write $w \models \phi$ (read "w models ϕ") if

$\phi = Q_a(x)$ for $a \in A$ if and only if $w = v_{x=i}$ for $v \in A^*$ and $v[i] = a$.

$\phi = R(x_1 \ldots, x_l)$ for an l-ary relational symbol R^l if and only if

$$w = v_{x_1=i_1, x_2=i_2, \ldots, x_l=i_l}$$

for $v \in A^*$ and $(i_1, \ldots, i_l) \in R^l$.

$\phi = x \in X$ if and only if w contains the letter (a_i, S_i, T_i, U_i) such that $x \in S_i$ and $X \in T_i$.

$\phi = (x, y) \in N$ if and only if w contains the letters

$$(a_i, S_i, T_i, U_i) \text{ and } (a_j, S_j, T_j, U_j)$$

such that $x \in S_i$, $y \in S_j$ and $N \in U_i \cap U_j$.

$\phi = \psi \wedge \theta$ if and only if $w \models \psi$ and $w \models \theta$.

$\phi = \psi \vee \theta$ if and only if $w \models \psi$ or $w \models \theta$.

$\phi = \neg\psi$ if and only if not $w \models \psi$, which we also denote as $w \not\models \psi$.

$\phi = \exists x\psi$ for a first order variable x if and only if

$$(a_1, S_1, T_1, U_1) \cdots (a_i, S_i \cup \{x\}, T_i, U_i) \cdots (a_n, S_n, T_n, U_n) \models \psi$$

for some $i \in [n]$.

$\phi = \forall x\psi$ for a first order variable x if and only if $w \models \neg\exists x\neg\psi$

$\phi = \exists X\psi$ for a monadic second order variable X if and only if there is a (possibly empty) set $I \subseteq [n]$ and for the $(\mathcal{V}_1, \mathcal{V}_1, \mathcal{V}_3)$-structure w' constructed by substituting each (a_i, S_i, T_i, U_i) by $(a_i, S_i, T_i \cup \{X\}, U_i)$ for $i \in I$ we have that $w' \models \psi$.

$\phi = \forall X\psi$ for a monadic second order variable X if and only if $w \models \neg\exists X\neg\psi$

$\phi = \exists N\psi$ for a binary second order variable X if and only if there is a (possibly empty) set $I \subseteq [n]$ and for the $(\mathcal{V}_1, \mathcal{V}_1, \mathcal{V}_3)$-structure w' constructed by substituting each (a_i, S_i, T_i, U_i) by $(a_i, S_i, T_i, U_i \cup \{N\})$ for $i \in I$ we have that $w' \models \psi$.

$\phi = \forall N\psi$ for a binary second order variable N if and only if $w \models \neg\exists N\neg\psi$

If a formula ϕ has no free variables we call ϕ a *sentence*. In this case ϕ can be modeled by $w \in A^*$ which can be viewed as $(\emptyset, \emptyset, \emptyset)$-structure and we define

$$L(\phi) := \{a_1 \cdots a_n \in A^* \mid (a_1, \emptyset, \emptyset, \emptyset) \cdots (a_n, \emptyset, \emptyset, \emptyset) \models \phi\}$$

as the language defined by ϕ. For the sake of readability we write $w \models \phi$ instead of $(a_1, \emptyset, \emptyset, \emptyset) \cdots (a_n, \emptyset, \emptyset, \emptyset) \models \phi$ for a word $w = a_1 \cdots a_n$ and a formula ϕ.

The *signature* of a formula, written as $\langle (Q_a)_{a\in A}, R^{l_1}, \ldots, R^{l_k} \rangle$ is the set of relational symbols and letter-predicates it might use. A formula is called a *first order formula* if the only variables it might use are first order variables. Let $S \subseteq \mathbb{N}^2$ be the successor relation, i.e. $S = \{(i, i+1) \mid i \in \mathbb{N}\}$ A language $L \subseteq A^*$ is in $\mathsf{FO}[(Q_a)_{a\in A}, S]$ (or simply $\mathsf{FO}[S]$ if the alphabet is understood in context) if there is a first order formula ϕ with the signature $\langle S, (Q_a)_{a\in A} \rangle$ such that $L = L(\phi)$.

If we extend a first order formulae by monadic second order variables we call it a *monadic second order formula*. The class of all languages defined by monadic second order formulae is called MSO. If we further allow the use of binary second order variables and their quantification we have an *second order formula* and call the defined class of languages SO.

2.3.5 Regular languages and their sub-classes

The regular languages are a family of formal languages definable in various ways, among which we consider (deterministic) finite automata, regular expressions, regular grammars and $\mathsf{MSO}[S]$.

The *locally threshold testable* languages are recognized by logical formulae in $\mathsf{FO}[S]$ and are a proper subset of the regular languages.

A true subset of locally threshold testable languages are the so called *local languages*. A language $L \subseteq A^*$ is called *local* if $(L = PA^+ \cup A^*S) \setminus A^*FA^*$, for $P, S \subseteq S$, $F \subseteq A^2$. This means P (S) is a set that specifies the first (last) letter of words in L and F is the set of *forbidden successors*, i.e. specifies which letter must not be followed by some other. We usually denote local languages in form of triples (P, S, F).

2.4 Operations on languages and families of languages

Let A and B be alphabets. A *homomorphism* (or *morphism*) is an function $h\colon A \to B^*$ that maps a letter of A to a word in B. We extend homomorphisms to words over A recursively: $h(\lambda) = \lambda$ and for $w \in A^*$ and $a \in A$ let $h(wx) = h(w)h(a)$. A homomorphism is called *length-preserving* if $|h(a)| = 1$ for all $a \in A$. If $h(a) \neq \lambda$ for all $a \in A$ we call h *non-erasing* (or λ-*free*). For a language L we denote the *(homo)morphic image* of L as $h(L) := \{h(w) \mid w \in L\}$ the application of h to each word in L. For $v \in B^*$ we call $h^{-1}(v) = \{w \in A^* \mid h(w) = v\}$ the *inverse (homo)morphic image* of v and analogously for a language $M \subseteq B^*$ the *inverse homomorphic image* of M is $h^{-1}(M) := \{w \in A^* \mid h(w) \in M\}$. For a language $L \subseteq A^*$ and a homomorphism $h\colon A^* \to B^*$ we call h *injective over* L if for all $u, v \in L$ with $h(u) = h(v)$ it holds that $u = v$.

A *family of languages* is a set of languages. We call a family of languages a *trio* if it closed under non-erasing inverse homomorphisms, non-erasing homomorphisms and the intersection with regular languages. This means if $\mathcal{L}$ is a trio and $L \in \mathcal{L} \subseteq A^*$ then for all regular languages R and non-erasing homomorphisms $h\colon : A^* \to B^*$ and $g\colon B^* \to A^*$ we have that $L \cap R, h(L), h^{-1}(L) \in \mathcal{L}$. If a trio is closed under arbitrary homomorphisms and inverse homomorphisms we call it *full trio*.

If a trio is additionally closed under union, concatenation and Kleene star we call it an *abstract family of languages* (AFL). This means if $\mathcal{L}$ is an AFL and $L_1, L_2 \in \mathcal{L} \subseteq A^*$ then also $L_1L_2, L_1 \cup L_2, L_1^* \in \mathcal{L}$. An AFL that is closed under arbitrary homomorphisms and inverse homomorphisms is called a *full AFL*.

2.5 Graphs

A *graph* G is a tuple $G = (V, E)$ where V is a set and $E \subseteq V^2$. We call V *nodes* or *vertices* and E *edges*. G is called *undirected* if for all $u, v \in V$ we have $(u, v) \in E \iff (v, u) \in E$. In that case we write edges as sets $\{u, v\}$. Otherwise, we call G *directed*. A graph is *edge-labeled* (or simply *labeled*) with a set $\mathcal{L}$ if there is a total function $E \to \mathcal{L}$. Sometimes we write $E = E_{l_1} \cup E_{l_2} \cup \cdots E_{l_n}$ to denote the labeling of the edges explicitly and without giving a function. Nodes u and v are *adjacent* if $(u, v) \in E$ or $(v, u) \in E$. Let $e \in E$ and $v \in V$. We call e *incident* in v if $e = (v, u)$ or $e = (u, v)$ for $u \in V$. The *degree* of a node $v \in V$ is the number of incident edges in v. By *in-degree* and *out-degree* of $v \in V$ we denote the number of incoming and outgoing edges, respectively. A *path* p in a graph is a sequence of

vertices $p = (v_1, \ldots, v_n)$ such that $v_1, \ldots, v_n \in V$ and $(v_i, v_{i+1}) \in E$ for $i = 1 \ldots, n-1$. We call a path $(v_1, \ldots, v_n)$ a *cycle* if $v_1 = v_n$. A path that does not contain a sub-path (with length > 1) that is a cycle is called *simple path*. We call a graph *connected* if for all $u \in V$ there is a path to every $v \in V$. For a directed graph $G = (V, E)$ let $\text{undir}(G)$ denote the graph $(V, \{\{u, v\} \mid (u, v) \in E\})$. A directed graph G is called *weakly connected* if $\text{undir}(G)$ is connected. A graph without cycles is called a *forest*. We call a forest that is connected a *tree*.

CHAPTER 3

Regular intersection emptiness

A main property of families of formal languages seems to be the closure of intersection with regular languages. This characteristic is (by definition) given as soon as e.g. trios are considered. Among the decidabilities of families of formal languages we have the decidability of the emptiness of a language, which is a question that is usually undecidable for complexity classes.

By combining these two properties into one we are left with the following decision problem:

Definition 3.0.1 (int_{Reg})**.** Let $int_{\mathsf{Reg}}(L)$ be the problem of deciding, given a finite automaton, whether its language has a non-empty intersection with L. For a family of languages $\mathcal{L}$ we say $int_{\mathsf{Reg}}(\mathcal{L})$ is decidable if $int_{\mathsf{Reg}}(L)$ is decidable for every $L \in \mathcal{L}$.

For families of formal languages, e.g. Reg, CFL and Index int_{Reg} the property is decidable. On the other hand, for complexity classes this is generally undecidable: Let $\mathcal{C}$ be complexity class which is defined by restricting resources such as time or space a Turing machine is allowed to use, as for instance NP, L, PSPACE. Every such complexity class contains a *trivial* complete language of the following form:

$$L_{T,r,f} := \{(\langle M\rangle \# x \# 0^n) \mid M \text{ is } T\text{-TM and recognizes } x \text{ with } f(n) \text{ of resource } r\} \quad ,$$

where T denotes either "*deterministic*" or "*non-deterministic*" and specifies the computation of the Turing machine, r is the restricted resource *space* or *time* and $f\colon \mathbb{N} \to \mathbb{N}$ is a function.

For all such $L_{T,r,f}$ we can show that $int_{\mathsf{Reg}}(L_{T,r,f})$ is undecidable. Let M_0 be an arbitrary, but fixed Turing-machine. Consider the regular language

$$R_0 := L(\langle M_0\rangle \# \Sigma^* \# 0^*) \quad .$$

We have, $L_{T,r,f} \cap R_0 = \emptyset$ if and only if $L(M_0) = \emptyset$. Since $L(M_0) = \emptyset$ is undecidable, so is $int_{\mathsf{Reg}}(L_{T,r,f})$.

These observations led us to the conjecture that the decidability of the int_{Reg}-property distinctly characterizes families of formal languages and might yield a tool to differentiate them from complexity classes.

In this chapter we disprove aforementioned conjecture.

The families of languages considered as *formal* all seem to be inside NP. We analyze (suitably encoded) sets of true quantified Boolean formulae and show that the int_{Reg} property is decidable. Therefore, we have a language which is PSPACE-complete and therefore likely to be outside of NP, but still holds the *formality* criterion. This result was published in [GKLW18]. Here, we present some of the proofs in a clearer way and give concise definitions of the used encoding.

We then strengthen this result with a technique called *Hiding* which was previously applied in [KL12a]. For any decidable language L we give an AC^0-many-one equivalent language L' such that $int_{\mathsf{Reg}}(L')$ is decidable. This means that all classes of languages which are closed under AC^0-reductions have complete problems for which int_{Reg} is decidable. Furthermore, we show that for a language L the property of having a decidable emptiness of intersection with a regular set is inherited by all members of the full AFL generated by L.

This chapter is structured as follows: We begin with introducing quantified Boolean formulae and give a suitable encoding. In Section 3.3 we show that int_{Reg} for quantified Boolean formulae with bounded quantifier alternation is decidable. We extend this result to arbitrary quantified Boolean formulae in Section 3.4. The *hiding*-technique is presented in Section 3.5. We finish this chapter with the analysis of the AFL generated by languages for which int_{Reg} is decidable in Section 3.6.

3.1 Languages of quantified Boolean formulae

In the following we will consider sets of true quantified Boolean formulae, which are dependent on a few restrictions and complete languages for various complexity classes.

Quantified Boolean formulae (QBF) can be defined inductively.

$$0 \text{ is a QBF}$$
$$1 \text{ is a QBF}$$
$$\text{A Boolean variable } x \text{ is a QBF}$$

Let ϕ and ψ be QBFs

$$\neg\phi \text{ is a QBF}$$
$$\phi \wedge \psi \text{ is a QBF}$$
$$\phi \vee \psi \text{ is a QBF}$$

Let x be a Boolean variable and ϕ be a QBF

$$\exists x\phi \text{ is a QBF}$$
$$\forall x\phi \text{ is a QBF}$$

In complexity theory, questions about the satisfiability/universality of quantified Boolean formulas are usually considered *indirectly*. Let ψ be a quantified Boolean formula with free variables $x_1, \ldots, x_n$. The QBF ϕ is satisfiable iff $\phi' := \exists x_1 \ldots \exists x_n \phi$ evaluates to true. Analogously, ϕ is a tautology iff $\phi'' := \forall x_1 \ldots \forall x_n \phi$ evaluates to true. Thus, all occurring Boolean variables will be bound by quantifiers, turning a QBF into a *sentence* which evaluates to either 0, or 1.

A quantified Boolean formula ψ is in *prenex-normal-form* if

$$\psi = \mathcal{Q}_1 x_1 \mathcal{Q}_2 x_2 \mathcal{Q}_n x_n \phi(x_1, x_2, \ldots, x_n) \ ,$$

where $\mathcal{Q}_i \in \{\exists, \forall\}$ are either existential or universal quantifiers, x_i are Boolean variables and ϕ is a quantifier-free, propositional formula. A propositional formula ϕ is said to be in *k-conjunctive normal form* (k-CNF) if

$$\phi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{k} l_{ij} \ ,$$

where l_{ij} are (possibly negated) Boolean variables, called *literals* and disjunctions of k literals are *clauses*.

From a complexity theory perspective quantifier alternations will have an impact on the difficulty of determining the validity of quantified Boolean formulae. Therefore, we will introduce the terminology for quantifying vectors of variables. Let $X := \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean variables and let ϕ be a propositional formula with n free variables. By $\exists X \phi$ we denote $\exists x_1 \exists x_2 \ldots \exists x_n \phi$ and analogously $\forall X \phi$ we denote $\forall x_1 \forall x_2 \ldots \forall x_n \phi$. Note, that the order of quantification is minuscule if one solely considers the validity of formulas with a single type of quantification.

Definition 3.1.1. Let $X_1, \ldots, X_n$ be sets of variables and let ϕ be a quantifier-free propositional formula with $\sum_{i=1}^{n} |X_i|$ free variables and $\mathcal{Q}_1, \ldots, \mathcal{Q}_n$ be a sequence of quantifiers with $\mathcal{Q}_i \neq \mathcal{Q}_{i+1}$ for $1 \leq i \leq n-1$. If $\mathcal{Q}_1 = \exists$ then $\mathcal{Q}_1 X_1 \ldots \mathcal{Q}_n X_n \phi(X_1, \ldots, X_n)$ is called a Σ_n-formula. If $\mathcal{Q}_1 = \forall$ then $\mathcal{Q}_1 X_1 \ldots \mathcal{Q}_n X_n \phi(X_1, \ldots, X_n)$ is called a Π_n-formula.

Deciding validity of quantified Boolean formulae is an expectedly high-complexity problem, since the NP-complete language SAT is equivalent to evaluating a Σ_1 formula.

Stockmeyer [Sto76] showed that for each $n \geq 1$, the set of true Σ_n and Π_n formulae form complete languages for a series of nested complexity classes – the *Polynomial Hierarchy* (PH.

Definition 3.1.2. Let $n \geq 1$. The class Σ_n^P is the set of all language $L \subseteq \{0,1\}^*$ which are recognized by a polynomial-time Turing machine M and a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ such that for all $x \in \{0,1\}^*$

$$x \in L \iff \exists c_1 \in \{0,1\}^{p(|x|)} \forall c_2 \in \{0,1\}^{p(|x|)} \ldots \mathcal{Q}_n c_n \in \{0,1\}^{p(|x|)} M(x, c_1, c_2, \ldots c_n) = 1 \ .$$

Here, $\mathcal{Q}_n$ is $\exists$ if n is odd and $\forall$ if n is even.

Another characterization of the Polynomial Hierarchy can be made by using *oracle Turing machines* or *alternating Turing machines*.

Stockmeyer also showed that the set of arbitrarily deep quantified true quantified Boolean formulae (TQBF) is PSPACE-complete.

3.2 Encoding quantified Boolean formulae

In this section we will provide a coding for quantified Boolean formulae where the explicit quantifiers are withdrawn. Thus, we will deal with propositional formulae where each variable is coded to *carry* its quantification depth and type.

Let $\psi = \mathcal{Q}_1 X_1 \mathcal{Q}_2 X_2 \cdots \mathcal{Q}_n X_n \phi(X_1, X_2, \ldots, X_n)$ be a quantified Boolean formula in prenex normal form where the propositional part ϕ is in 3-CNF with $\sum_{i=1}^{n} |X_i|$ free variables

and $\mathcal{Q}_i$ are alternating quantifiers. Let Ψ be the set of such QBFs ψ. Let the function $\text{quant}(\psi) = \mathcal{Q}_1 X_1 \mathcal{Q}_2 X_2 \cdots \mathcal{Q}_n X_n$ return the sequence of quantifiers of a given QBF and let $\text{prop}(\psi) = \phi(X_1, X_2, \ldots, X_n)$ the analogous operator, return the propositional part of a quantified Boolean formula.

Definition 3.2.1. Let $\Gamma = \{a, b, \oplus, \ominus, \langle, \rangle, \vee, \wedge\}$. Let $\text{dec}\colon \Gamma^* \to \Psi$ be defined as follows: Each *literal* is a sequence of the form $(\oplus|\ominus)b^+a^+$, where $\oplus, \ominus$ denote if the literal is positive or negated, the bs are a unary number encoding of the quantification depth and the as are a unary number encoding the name of the variable.

$$\begin{aligned}\text{dec}(\oplus b^i a^j) &= \mathcal{Q}_i x_{ij} x_{ij} \\ \text{dec}(\ominus b^i a^j) &= \mathcal{Q}_i x_{ij} \neg x_{ij} \ ,\end{aligned}$$

where

$$\mathcal{Q}_i := \begin{cases} \exists \text{if } i \text{ is even} \\ \forall \text{if } i \text{ is odd} \ . \end{cases}$$

Let $l_1, l_2, l_3 \in (\oplus|\ominus)b^+a^+$ be literals. A *clause* is encoded by encoding each of its literals, where the quantifiers of each literal are merged with the method denoted in Algorithm 2:

$$\begin{aligned}\text{dec}(\langle l_1 \vee l_2 \vee l_3 \rangle) = &\,\text{merge}(\text{quant}(\text{dec}(l_1)), \text{quant}(\text{dec}(l_2)), \text{quant}(\text{dec}(l_3)))) \\ &\text{prop}(\text{dec}(l_1)) \vee \text{prop}(\text{dec}(l_2)) \vee \text{prop}(\text{dec}(l_3))\end{aligned}$$

Let $c_1, \ldots, c_n \in \langle (\oplus|\ominus)b^+a^+ \vee (\oplus|\ominus)b^+a^+ \vee (\oplus|\ominus)b^+a^+ \rangle$ be clauses. A formula is encoded by encoding each of its clauses:

$$\begin{aligned}\text{dec}(c_1 \wedge \cdots \wedge c_n) = &\,\text{merge}(\text{quant}(\text{dec}(c_1), \ldots, \text{dec}(c_n))) \\ &\text{prop}(\text{dec}(c_1) \wedge \cdots \wedge \text{dec}(c_n))\end{aligned}$$

Without loss of generality assume $i_1 \leq i_2 \leq \cdots \leq i_n$ (otherwise sort the sequence of quantifiers accordingly) and consider a sequence of quantified variables $\mathcal{Q}_{i_1} x_{i_1 j_1}, \mathcal{Q}_{i_2} x_{i_2 j_2}, \ldots, \mathcal{Q}_{i_n} x_{i_n j_n}$. Algorithm 2 formally shows how to merge quantified variables into sets.

Algorithm 2: Algorithm to compute merge

Data: $\mathcal{Q}_{i_1} x_{i_1 j_1}, \mathcal{Q}_{i_2} x_{i_2 j_2}, \ldots, \mathcal{Q}_{i_n} x_{i_n j_n}$, sorted by i_l.
Result: $\text{merge}(\mathcal{Q}_{i_1} x_{i_1 j_1}, \mathcal{Q}_{i_2} x_{i_2 j_2}, \ldots, \mathcal{Q}_{i_n} x_{i_n j_n})$

```
quantifiers ← λ;
for l ← 1, ..., n do
    X_l ← {x_{i_k j_k} | i_k = i_l};
    if ∀l' < l : X_l ≠ X_{l'} then
        quantifiers ← quantifiers · Q_l X_l;
    end
end
return quantifiers
```

Example 3.2.2. Consider the encoded words w_1, w_2 and w_3

$$\begin{aligned}w_1 &= \langle \oplus bbbbaa \vee \oplus bbaa \vee \ominus baaa \rangle \ , \\ w_2 &= \langle \ominus baa \vee \oplus baa \vee \ominus bbaaa \rangle \wedge \langle \oplus baa \vee \ominus bbbbbaa \vee \oplus baaaa \rangle \ , \\ w_3 &= \langle \oplus bba \vee \ominus bbbaa \vee \ominus bbbba \rangle \wedge \langle \ominus bbaa \vee \oplus bbbbaaa \vee \ominus bbaaaaaa \rangle \ .\end{aligned}$$

Then the decoding of w_1, w_2 and w_3 respectively are

$$\begin{aligned}
\mathrm{dec}(w_1) &= \exists\{x_{1,3}\}\forall\{x_{2,2}\}\forall\{x_{4,2}\}(x_{4,2} \vee x_{2,2} \vee \neg x_{1,3}) \ , \\
\mathrm{dec}(w_2) &= \exists\{x_{1,2}, x_{1,4}\}\forall\{x_{2,3}\}\exists\{x_{5,2}\}(\neg x_{1,2} \vee x_{1,2} \vee x_{2,3}) \wedge (x_{1,2} \vee \neg x_{5,2} \vee x_{1,4}) \ , \\
\mathrm{dec}(w_3) &= \forall\{x_{2,1}, x_{2,2}, x_{2,6},\}\exists\{x_{2,2}, x_{2,6}\}(x_{2,1} \vee \neg x_{3,2}\neg x_{3,1}) \wedge (\neg x_{2,2} \vee x_{3,3}\neg x_{2,6}) \ .
\end{aligned}$$

Note that there are two consecutive universal quantifiers in w_1. This is due to the fact that w_1 does not contain a literal with b^3 and thus there is no existential quantifier between levels 2 and 4. w_2 contains literals with the same amount of bs. Thus there are non-singleton sets of variables that are quantified by the same existential quantifies. Finally, w_3 does not contain any literal with a single b, making X_1 empty. This means the literal(s) with the minimal number of bs decide whether the encoded formula is a Σ- or a Π-formula.

Definition 3.2.3. Let α be a regular expression and $k \in \mathbb{N}$ be fixed. By $\alpha^{\leq k}$ we denote $(\alpha \mid \alpha^2 \mid \alpha^3 \mid \cdots \mid \alpha^k)$ the repetition of α up to k times.

Definition 3.2.4. Let $k \in \mathbb{N}$ be fixed and let $\pm = (\oplus|\ominus)$ be a regular expression. We denote the set of encoded quantified Boolean formulae with k sets of quantified variables (and thus $k-1$ quantifier alternations) as

$$L_{k\text{-Q}_{\mathrm{BF}}} := L\left(\begin{array}{l}\left\langle \pm b^{\leq k}a^+ \vee \pm b^{\leq k}a^+ \vee \pm b^{\leq k}a^+ \right\rangle \\ \left(\wedge \left\langle \pm b^{\leq k}a^+ \vee \pm b^{\leq k}a^+ \vee \pm b^{\leq k}a^+ \right\rangle\right)^*\end{array}\right) \ .$$

Analogously, define

$$L_{\mathrm{Q_{BF}}} := \bigcup_{k\geq 1}\left(\begin{array}{l}\left\langle \pm b^+a^+ \vee \pm b^+a^+ \vee \pm b^+a^+ \right\rangle \\ \left(\wedge \left\langle \pm b^+a^+ \vee \pm b^+a^+ \vee \pm b^+a^+ \right\rangle\right)^*\end{array}\right)$$

as the set of encoded quantified Boolean formulae of arbitrary quantification depth.

Lemma 3.2.5. *Let $L_{\Sigma_k} \subseteq L_{k\text{-Q}_{\mathrm{BF}}}$ be defined as*

$$L_{\Sigma_k} = \left\{ w \in L_{k\text{-Q}_{\mathrm{BF}}} \,\middle|\, \begin{array}{l}\textit{the first quantifier of } \mathrm{dec}(w) \textit{ is existential and} \\ \mathrm{dec}(w) \textit{ evaluates to true}\end{array} \right\}$$

and let $L_{\Pi_k} \subseteq L_{k\text{-Q}_{\mathrm{BF}}}$ be defined as

$$L_{\Pi_k} = \left\{ w \in L_{k\text{-Q}_{\mathrm{BF}}} \,\middle|\, \begin{array}{l}\textit{the first quantifier of } \mathrm{dec}(w) \textit{ is universal and} \\ \mathrm{dec}(w) \textit{ evaluates to true}\end{array} \right\} \ .$$

For even k the set L_{Σ_k} is Σ_k^P complete and for odd k the language L_{Π_k} is Π_k^P complete.

Proof. Following Wrathall's [Wra76] result the set of true Σ_k is Σ_k^P-complete for odd k and the set of true Π_k formulae is Π_k^P complete for even k. The decoding function dec works as a linear time reduction and yields the completeness results for L_{Σ_k} and L_{Π_k} respectively. □

Definition 3.2.6. Let $L_{k\text{-T}_{\mathrm{QBF}}} := L_{\Sigma_k} \cap L_{\Pi_k}$ be the set of encoded true quantified Boolean formulae in 3-CNF with up to $k-1$ quantifier alternations. Let $L_{\mathrm{TQBF}} := \{w \in L_{\mathrm{Q_{BF}}} \mid \mathrm{dec}(w) \text{ evaluates to true}\}$ be the set of encoded true quantified Boolean formulas in 3-CNF of arbitrary quantifier alternation.

3.3 Quantified Boolean formulae of bounded quantification depth

In this section we will show that for each $k \in \mathbb{N}$ the languages L_{Σ_k} and L_{Π_k} have a decidable emptiness of regular intersection, i.e. $int_{\mathsf{Reg}}(L_{\Sigma_k})$ and $int_{\mathsf{Reg}}(L_{\Pi_k})$ is decidable. The int_{Reg}-property of these two language sets can be viewed from another perspective, when neglecting the requirement of the first quantifier being existential/universal:

> For a fixed k, given a regular language R, is there a $w \in R$ that is a true quantified Boolean formula with up to k alternating quantifiers in R?

Consider a regular language R. If $R \cap L_{k\text{-QBF}} = \emptyset$ is decidable then $R \cap L_{\Sigma_k} = \emptyset$ and $R \cap L_{\Pi_k} = \emptyset$ are decidable, too. If R_f is finite, then test for every $w \in R_f$ if w is valid. The intersection with L_{Σ_k} is non-empty if there is a valid $w \in R_f$ such that the first quantifier of $\mathrm{dec}(w)$ is existential. Analogously, if in a valid w the first quantifier is universal we have $R_f \cap L_{\Pi_k} \neq \emptyset$.

Without loss of generality, in order to decide if a regular language contains true quantified Boolean formulae we only consider (infinite) regular languages $R \subseteq L_{k\text{-QBF}}$. Decoding every single word and testing its validity is obviously not possible in this scenario. We will describe a method in which for each infinite regular language R a finite subset $\{w_1, \ldots, w_n\} \subseteq R$ is extracted such that R contains a true formula iff there is a valid w_i in $\{w_1, \ldots, w_n\}$.

Definition 3.3.1. Let $k \geq 1$ be a natural number, $\Gamma = \{a, b, \langle, \rangle, \oplus, \ominus, \vee, \wedge\}$ and let $M = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(M) \subseteq L_{k\text{-QBF}}$. For each pair of states $q, q' \in Q$, $d \in [1, k]$ and $s \in \{\oplus, \ominus\}$ we define the *literal transition set* from q to q' with quantification depth d and sign s as

$$\Lambda_{q,q'}^{d,s} := \left\{ w \in (\langle, \lambda)sb^d a^+(\vee|\rangle) \mid \delta^*(q, w) = q' \right\} \ .$$

Moreover, we define the union of literal transition sets over all signs and depths for two particular states q, q'

$$\Lambda_{q,q'} := \bigcup_{s \in \{\oplus,\ominus\}} \bigcup_{d=1}^{k} \Lambda_{q,q'}^{d,s}$$

A literal transition set from q to q' with quantification depth d and sign s of an automaton M recognizing a subset of $L_{k\text{-QBF}}$ contains all words w that are encoded literals and *lead* M from state q to q'.

Example 3.3.2. Consider the following component of a deterministic finite automaton depicted in Figure 3.1. For $i, j \in [8]$ the literal transition sets Λ_{q_i,q_j} are empty, since every literal is considered to carry its delimiters. Only $\Lambda_{q,q'}$ is non-empty and it partitioned into the following non-empty literal transitioning sets.

$$\begin{aligned}
\Lambda_{q,q'}^{1,\oplus} &= \{\langle \oplus ba \vee\} \\
\Lambda_{q,q'}^{1,\ominus} &= \left\{\langle \ominus b(aa)^i a \vee \mid i \geq 0\right\} \\
\Lambda_{q,q'}^{2,\oplus} &= \left\{\langle \oplus bb(aa)^i a \vee \mid i \geq 0\right\} \\
\Lambda_{q,q'}^{4,\oplus} &= \left\{\langle \oplus bbbb(aa)^i a \vee \mid i \geq 0\right\}
\end{aligned}$$

Lemma 3.3.3. *Let $M = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(M) \subseteq L_{k\text{-QBF}}$. For each $q, q' \in Q$, sign $s \in \{\oplus, \ominus\}$ and $d \in [1, k]$ the literal transition set $\Lambda_{q,q'}^{d,s}$ is regular.*

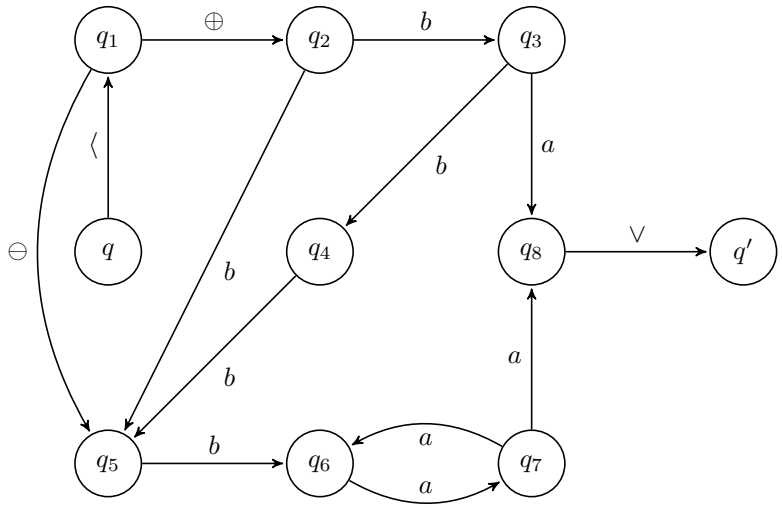


Figure 3.1: Illustration of a (partial) transition function of a DFA. In Example 3.3.2 the respective literal transition sets are denoted.

Proof. Let $M_{q,q'} := (Q, \Gamma, \delta, q, \{q'\})$ be the deterministic finite automaton that is constructed from M by taking q as the new initial state and q' as the single final state. Let $l := (\langle|\lambda)sb^k a^+(\vee|\rangle)$ be the regular expression of the literal structure. For each $q, q' \in Q$, sign $s \in \{\oplus, \ominus\}$ and $d \in [1, k]$ we have

$$\Lambda_{q,q'}^{d,s} = L(M_{q,q'}) \cap L(l) \ .$$

Since $L(M_{q,q'})$ and $L(l)$ are regular, $\Lambda_{q,q'}^{d,s}$ is regular as well. □

Intuitively speaking, a quantified Boolean formula is *more likely* to be true, if it contains fewer universally quantified variables. We try to *unite* the variables used in the literals of a formula. To this end we have to test every combination of possible transition sets intersections.

Definition 3.3.4. Let $k \geq 1$ and let $M = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(M) \subseteq L_{k\text{-QBF}}$ and let $d \leq k$ be an even number. We define

$$\Upsilon^d := \{\Lambda_{q,q'}^{d,s} \mid q, q' \in Q, s \in \{\oplus, \ominus\}\}$$

as the set of all literal transition sets with quantification depth d. Note, that only sets of universally quantified literals are contained in the Υ sets.

Definition 3.3.5. Define the homomorphism $\text{trunc} : \{a, b, \langle, \rangle, \oplus, \ominus\} \to \{a, b\}$ with

$$\text{trunc}\,(\sigma) := \begin{cases} \sigma & \text{if } \sigma \in \{a, b\}, \\ \lambda & \text{otherwise.} \end{cases}$$

Let $\Lambda \subseteq \{a, b, \langle, \rangle, \oplus, \ominus\}^*$ and $w \in \text{trunc}\,(\Lambda)$. Define

$$\text{extend}\,(w, \Lambda) := \text{trunc}^{-1}(w) \cap \Lambda$$

as the operation that reverses trunc for a language Λ and a word in Λ.

The trunc function is used to extract the variable out of a literal, i.e. remove positional information and the sign. The function extend does *reverse* of trunc – given some variable and a literal transition set, it returns all literals in the transition set that use this variable.

Example 3.3.6. Consider the following literal transition set $\Lambda_1 = L((\oplus|\ominus)bbb(aa)^*a\vee)$ and $w = \langle\oplus bbaaaaa\vee$. Then we have $\text{trunc}\,(\langle\oplus bbaaaaa\vee) = bbaaaaa$ and $\text{extend}\,(\Lambda_1, bbaaaaa) = \{\langle\oplus bbaaaaa\vee, \langle\ominus bbaaaaa\vee\}$.

Definition 3.3.7. Define for every even d with $1 \leq d \leq k$

$$P^d := \left\{ p \in \Upsilon^d \mid \bigcap_{\Lambda\in p} \text{trunc}\,(\Lambda) \neq \emptyset \right\} \ .$$

P^d contains sets of literal transition sets which share common variables.

If there are no shared variables between any $p \in P^d$ then there are only singleton sets in P^d.

In the following we will construct a *finite* representative set for each (possibly infinite) literal transition set.

Definition 3.3.8. Let $\text{rep}\colon \{a, b, \langle, \rangle, \oplus, \ominus\}^* \to \{a, b, \langle, \rangle, \oplus, \ominus\}^*$ be defined by considering both, odd and even quantification depths.

1. If d is odd, then $\Lambda^{d,s}_{q,q'}$ contains only literals that are existentially quantified.

$$\text{rep}(\Lambda^{d,s}_{q,q'}) := \begin{cases} \text{extend}(b^d a^l, \Lambda^{d,s}_{q,q'}) & \text{if } \left|\Lambda^{d,s}_{q,q'}\right| = \infty \ , \\ \Lambda^{d,s}_{q,q'} & \text{otherwise} \ , \end{cases}$$

   where for infinite $\Lambda^{d,s}_{q,q'}$ the length l of the a-factor is set to a *unique* value such that we have for infinite and odd quantification depth literal transition sets Λ and Λ' so that

$$\#_b(\text{rep}(\Lambda)) = \#_b(\text{rep}(\Lambda')) \Rightarrow \Lambda = \Lambda' \ .$$

2. For even d we deal with literal sets of universally quantified variables. We compute representatives for all literal transition sets of quantification depth d, i.e. for Υ^d simultaneously with Algorithm 3. For a language L the operation $\min_{\text{lex}}(L)$ gives the lexicographic minimal element of L. Any other deterministic operation extracting a definite element from a set. This means for every universal literal set we extract the variables they contain. We then look at the power set of these variable sets and pick a representative for each non-empty element. This way we test every combination of a literal transition set with each other.

Lemma 3.3.9. *For every d with $1 \leq d \leq k$, $s \in \{\oplus, \ominus\}$ and $q, q' \in Q$ the representative set $\text{rep}(\Lambda^{d,s}_{q,q'})$ is finite.*

Proof. For odd d and finite $\Lambda^{d,s}_{q,q'}$ the representative is the set itself and therefore finite. If it is infinite, $\text{rep}(\Lambda^{d,s}_{q,q'})$ is a singleton. For even d Algorithm 3 is used to compute representatives. There, for every $p \in P^d$ we compute a single representative and possibly add it to the representative set for $\Lambda^{d,s}_{q,q'}$. Since $|P^d| \leq |\mathcal{P}(\Upsilon^d)| \leq 2^{|Q|^2}$, we have that $\text{rep}(\Lambda^{d,s}_{q,q'})$ is finite. □

Algorithm 3: Algorithm to compute merge

Data: Υ^d, literal transition sets Λ of quantification depth d.
Result: Finitely many representative for each Λ of quantification depth d.
forall $\Lambda \in \Upsilon^d$ **do**
  $\text{rep}(\Lambda) \leftarrow \emptyset$;
end
forall $p \in P^d$ **do**
  $\text{rep}(p) \leftarrow \min_{\text{lex}} \left(\bigcap_{\Lambda \in p} \text{trunc}(\Lambda) \right)$;
end
forall $\Lambda \in p$ **do**
  $\text{rep}(\Lambda) \leftarrow \text{rep}(\Lambda) \cup \text{extend}(\text{rep}(p), \Lambda)$
end

The notion of representatives is now lifted to more general literal sets and to clauses respectively.

Definition 3.3.10. For each pair of states $q, q' \in Q$ define $\text{rep}(\Lambda_{q,q'})$ as the set of representatives from q to q':

$$\text{rep}(\Lambda_{q,q'}) := \bigcup_{s \in \{+,-\}, d \leq k} \text{rep}\left(\Lambda_{q,q'}^{d,s}\right)$$

The literals may now have arbitrary quantification depth and sign.

Since we only consider regular languages which are subsets of $L_{k\text{-Q}_{\text{BF}}}$, we know that each literal transition set contains only mutually exclusive first, second or third literals in a clause. Thus, we can partition the set of literal sets into three disjunctive sets.

Definition 3.3.11. Let the set of *first*, *second* and respectively *third* literals be defined as

$$\begin{aligned}
L_1 &:= \{\Lambda_{q,q'} \mid q, q' \in Q, \Lambda_{q,q'} \neq \emptyset \text{ and all } w \in \Lambda_{q,q'} \text{ start with } \langle\} \\
L_2 &:= \{\Lambda_{q,q'} \mid q, q' \in Q, \Lambda_{q,q'} \neq \emptyset \text{ and all } w \in \Lambda_{q,q'} \text{ contain neither } \rangle, \text{ nor } \langle\} \\
L_3 &:= \{\Lambda_{q,q'} \mid q, q' \in Q, \Lambda_{q,q'} \neq \emptyset \text{ and all } w \in \Lambda_{q,q'} \text{ end with } \rangle\}
\end{aligned}$$

Definition 3.3.12. For every $q, q' \in Q$ define the *clause transition sets* $C_{q,q'}$ as

$$C_{q,q'} := \bigcup_{q_1,q_2 \in Q, \Lambda_{q,q_1} \in L_1, \Lambda_{q_1,q_2} \in L_2, \Lambda_{q_2,q'} \in L_3} \text{rep}(\Lambda_{q,q_1}) \cdot \text{rep}(\Lambda_{q_1,q_2}) \cdot \text{rep}(\Lambda_{q_2 q'}) \ .$$

Lemma 3.3.13. *For every $q, q' \in Q$ clause transition sets are $C_{q,q'}$ finite.*

Proof. Every clause transition set is a finite concatenation of elements of finite sets and thus is finite itself. □

Definition 3.3.14 (Condensed automaton)**.** Let $M = (Q, \Gamma, \delta, q_0, F)$ be an DFA with $L(M) \subseteq L_{k\text{-Q}_{\text{BF}}}$. Define the *condensed automaton* of M as $\text{condense}(M) = (Q, \Gamma', \delta', q_0, F)$, where the alphabet Γ' consists of whole clauses and junctors

$$\Gamma' = \bigcup_{q,q' \in Q} C_{q,q'} \cup \{\wedge\} \ ,$$

and

$$\delta' = \begin{array}{l} \{(q, w, q') \mid q, q' \in Q, w \in C_{q,q'}\} \cup \\ \{(q, \wedge, q') \mid q, q' \text{ and } (q, \wedge, q') \in \delta\} \end{array} \ .$$

The condensed automaton recognizes formulae where each literal is a *representative*. Therefore, for each finite automaton M we have that $L(\text{condense}(M)) \subseteq L(M)$. In the following we will show that the *condensation* preserves containment of valid formulae.

Lemma 3.3.15. *Let M be a DFA with $L(M) \subseteq L_{k\text{-QBF}}$. If $L(\text{condense}(M)) \cap L_{k\text{-TQBF}} \neq \emptyset$ then also $L(M) \cap L_{k\text{-TQBF}} \neq \emptyset$.*

Proof. The condensed automaton condense(M) recognizes a subset of M. Thus, if

$$L(\text{condense}(M)) \cap L_{k\text{-TQBF}} \neq \emptyset$$

then also $L(M) \cap L_{k\text{-TQBF}} \neq \emptyset$. □

Lemma 3.3.16. *Let $M = (Q, \Gamma, \delta, q_0, F)$ be an DFA with $L(M) \subseteq L_{k\text{-QBF}}$. If $L(M) \cap L_{k\text{-TQBF}} \neq \emptyset$ then also $L(\text{condense}(M)) \cap L_{k\text{-TQBF}} \neq \emptyset$.*

Proof. Assume that $L(M) \cap L_{k\text{-TQBF}} \neq \emptyset$. Then, there is some $w \in L(M)$ that is a true quantified Boolean formula. Since $L(M) \subseteq L_{k\text{-QBF}}$ the formula w can be factorized in the following way

$$w = w_0 w_1 w_2 \wedge w_3 w_4 w_5 \wedge \cdots \wedge w_{n-2} w_{n-1} w_n \ ,$$

such that for $q_j \in Q$ we have

- $w_i \in \Lambda_{q_i, q_{i+1}}$ for $i \not\equiv 0 \mod 3$,
- $w_i \in \Lambda_{q_i, q_i'}$ for $i \equiv 0 \mod 3$,
- $\delta(q_i, \wedge) = q_i'$ for $i \equiv 0 \mod 3$,
- $w_0 \in \Lambda_{q_0, q_1}$ and
- $w_n \in \Lambda_{q_n, q_f}$ for $q_f \in F$.

There is a $w' \in L(\text{condense}(M))$ which can be factorized the same way as w into

$$w' = w_0' w_1' w_2' \wedge w_3' w_4' w_5' \wedge \cdots \wedge w_{n-2}' w_{n-1}' w_n' \ ,$$

such that if $w_i \in \Lambda_{q_i q_{i+1}}$ then $w_i \in \text{rep}(\Lambda_{q_i q_{i+1}})$. We make a case distinction over the factors w_i to show that there is a w' that is a true quantified Boolean formula:

1. If $\Lambda_{q_i, q_{i+1}}$ is finite then $\Lambda_{q_i, q_{i+1}} = \text{rep}(\Lambda_{q_i, q_{i+1}})$ and therefore we can pick $w_i' = w_i \in \text{rep}(\Lambda_{q_i, q_{i+1}})$.
2. If $\Lambda_{q_i, q_{i+1}}$ contains infinitely many existentially quantified variables then pick w_i' such that it is uniquely occurring in w'. Every existentially quantified variable that occurs uniquely in a quantified Boolean formula equals to the Boolean constant 1.
3. If $\Lambda_{q_i, q_{i+1}}$ contains universally quantified variables we have to consider every possible intersection of $\Lambda_{q_i, q_{i+1}}$ with other literal transition sets that contain universally quantified variables. For w_i let the index set $J = \{j \mid j \in 1, \ldots, n \text{ and } \text{trunc}(w_j) = \text{trunc}(w_i)\}$ contain the indices of all factors in w in which the universally quantified variable in w_i also occurs. Let $d := \#_b(w_i)$ be the number of bs in w_i (and therefore its quantification depth). Then by definition we have that, $\{\Lambda_{q_j, q_{j+1}} | j \in J\} \in P^d = \{p \in \Upsilon^d \mid \bigcap_{\Lambda \in p} \text{trunc}\,(\Lambda) \neq \emptyset\}$. Then, with the construction of Algorithm 3 we assign to each w_j with $j \in J$ a label p_j which yields a consistent renaming of the variable names and preserves validity of the quantified Boolean formula.

The substitutions preformed on each factor w_i of w yield the same valuation. Therefore, $w' \in L(\text{condense}(M))$ is also a true quantified Boolean formula □

The *condensation* process of an automaton recognizing encoded quantified Boolean formula now restricts the number of variables in a formula. Formulae recognized by a condensed automaton are limited to a finite amount of variables. Therefore, deciding whether such an automaton recognizes a true quantified Boolean formula is decidable.

Lemma 3.3.17. *Let $M = (Q, \Gamma', \delta', q_0, F)$ be the condensed automaton of some deterministic finite automaton and $w = w_1 \dots w_n \in L(M)$ such that $\hat{\delta}(q_0, w_1 \dots w_i) = q = \hat{\delta}(q_0, w_1 \dots w_j)$ for $1 \leq i < j \leq n$. Let $w' = w_1 \dots w_i w_{j+1} \dots w_n$ i.e. w without the factor read in the loop from i to j. If w evaluates to true, then so does w'.*

Proof. Since M is a condensed automaton the true formula $w \in L(M)$ is of the form $c_1 \wedge \dots \wedge c_m$ for disjunctions of literals c_l for $1 \leq l \leq m$. Each factor $w_{i+1} \dots w_j$ induces for the sub-formula $c_s \wedge \dots \wedge c_t$ with $1 \leq s \leq t \leq m$. The conjunction of clauses implies that each c_l evaluates to true such that w does. Therefore, we have that

$$c_1 \wedge \dots \wedge c_m \equiv 1 \Rightarrow c_1 \wedge c_{s-1} \wedge c_{t+1} \wedge \dots \wedge c_m \equiv 1 \ ,$$

i.e. we can remove clauses from w while preserving validity. □

Corollary 3.3.18. *Let $M = (Q, \Gamma', \delta', q_0, F)$ be the condensed automaton of some deterministic finite automaton and $w = w_1 \dots w_n \in L(M)$ be a true quantified Boolean formula. Then there is a true quantified Boolean formula $w_s \in L(M)$ such that the accepting path in M with the label w_s is simple.*

Proof. A suited w_s can be obtained by iterative application of Lemma 3.3.17 on w. □

Therefore, if there are words in a condensed automaton that represent true quantified Boolean formulae, there are also true ones which are accepted via simple path.

Lemma 3.3.19. *Let R be a regular language and M be a deterministic finite automaton with $L(M) = R$. It is decidable whether* condense(M) *recognizes a true quantified Boolean formula. In particular we can decide*

1. $L(\text{condense}(M)) \cap L_{\Sigma_k} = \emptyset$ *and*
2. $L(\text{condense}(M)) \cap L_{\Pi_k} = \emptyset$.

Proof. Assume $R \subseteq L_{k\text{-Q}_{\text{BF}}}$, otherwise consider the regular set $R \cap L_{k\text{-Q}_{\text{BF}}}$. Let $w_1, \dots, w_n$ be finitely many words that are labels of simple accepting paths in condensed(M). Evaluate each quantified Boolean formula w_i. If for all $1 \leq i \leq n$ we have that w_i evaluates to false, then both $L(\text{condense}(M)) \cap L_{\Sigma_k} = \emptyset$ and $L(\text{condense}(M)) \cap L_{\Pi_k} = \emptyset$. The intersection condense$(M) \cap L_{\Sigma_k}$ is non-empty, if there is an i such that w_i evaluates to true and the first quantifier of w_i is existential. If there is a true w_i where the first quantifier is universal the intersection of R and L_{Π_k} is non-empty. □

Putting together the previous lemmata we obtain the desired decidability result.

Theorem 3.3.20. *Let R be a regular language and k be a natural number. It is decidable if $R \cap L_{\Sigma_k} = \emptyset$ and $R \cap L_{\Pi_k} = \emptyset$.*

Proof. Assume $R \subseteq L_{k\text{-}\mathrm{QBF}}$, otherwise consider the regular set $R \cap L_{k\text{-}\mathrm{QBF}}$. Let M be the minimal deterministic automaton recognizing R and let $M' := \mathrm{condense}(M)$. Following Lemma 3.3.19 it is decidable if $L(M') \cap L_{\Sigma_k} = \emptyset$ and $L(M') \cap L_{\Pi_k} = \emptyset$. Lemma 3.3.15 and Lemma 3.3.16 together state that

$$L(M) \cap L_{k\text{-}\mathrm{TQBF}} \iff L(M') \cap L_{k\text{-}\mathrm{TQBF}} \ .$$

Therefore, it is decidable if $R \cap L_{\Sigma_k} = \emptyset$ and $R \cap L_{\Pi_k} = \emptyset$. □

Theorem 3.3.20 yields for every level k of the polynomial hierarchy languages that are complete for Σ_k^P and Π_k^P for which we can decide the emptiness of intersection with regular sets.

3.4 The decidability of $int_{\mathsf{Reg}}(L_{\mathbf{TQBF}})$

We can extend the result of the previous section to decide whether regular sets of quantified Boolean formulae unbounded in their quantification depth contain true formulae. This means we show that $int_{\mathsf{Reg}}(L_{\mathrm{TQBF}})$ is decidable.

Lemma 3.4.1. *Let $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_m)$ be vectors of Boolean variables, let ϕ be a propositional Boolean formula with $|X|+|Y|$ free variables and let ψ be a propositional Boolean formula with $|X|$ free variables. Then, we have*

1. $\exists X \forall Y \phi(X, Y) \Rightarrow \forall Y \exists X \phi(X, Y)$
2. $\forall X \psi(X) \Rightarrow \exists X \psi(X)$
3. $\exists X \exists Y \phi(X, Y) \Rightarrow \exists Y \exists X \phi(X, Y)$
4. $\forall X \forall Y \phi(X, Y) \Rightarrow \forall Y \forall X \phi(X, Y)$

Proof.

1. Let $\exists X \forall Y \phi(X, Y) \equiv 1$. Then there is a Boolean vector (i.e. an assignment to the variables in X) $C = (c_1, \ldots, c_n) \in \{0,1\}^n$ such that $\forall Y \phi(C, Y) \equiv 1$. Then, for $\forall Y \exists X \phi(X, Y)$ we can assign c_i for each x_i for $1 \leq i \leq n$ and obtain that $\forall Y \exists X \phi(X, Y) \equiv 1$.
2. Let $\forall X \psi(X) \equiv 1$. Then for all Boolean vectors $C = (c_1, \ldots, c_n) \in \{0,1\}^n$ the formula $\psi(C) \equiv 1$. Therefore, we can pick any C to find a satisfying assignment for ψ and thus $\exists X \psi(X) \equiv 1$.
3. If $\exists X \exists Y \phi(X, Y) \equiv 1$ then there are Boolean vectors $C = (c_1, \ldots, c_n) \in \{0,1\}^n$ and $D = (d_1, \ldots, d_m) \in \{0,1\}^m$ such that $\psi(C, D) \equiv 1$. We can assign c_i for each x_i for $1 \leq i \leq n$ and therefore have $\exists X \phi(X, D) \equiv 1$. By also assigning d_j for each y_j for $1 \leq j \leq m$ we are left with $\exists Y \exists X \phi(X, Y) \equiv 1$.
4. Analogous to 3.

□

This means, that we do not falsify quantified Boolean formulae, by *pulling out* universally quantified variables. It is also more likely that a QBF is true if existential quantifiers are used instead of universal ones. Furthermore, the order of consecutive variables with the same kind of quantification is interchangeable.

3.4.1 Bounding the quantification depth

In the following we will reduce deciding $int_{\mathsf{Reg}}(L_{\text{TQBF}})$ to deciding $int_{\mathsf{Reg}}(L_{k\text{-TQBF}})$ for some fixed k.

Definition 3.4.2. Let $M = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(M) \subseteq L_{\text{QBF}}$, let $q, q' \in Q$ be states and let $s \in \{\oplus, \ominus\} \subseteq \Gamma$ be a sign symbol. We define $G^s_{q,q'}$ as the set of complete quantification depths that can be between q and q', when a literal with sign s is read M:

$$G^s_{q,q'} := \{n \in \mathbb{N} \mid \exists q_1, q_2 \in Q : \delta(q_1, s) = q \wedge \delta(q, b^n) = q' \wedge q' \wedge \delta(q', a) = q_2\}$$

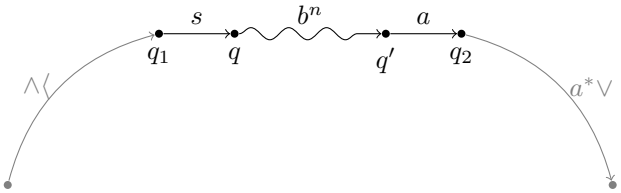


Figure 3.2: Visualization of $n \in G^s_{q,q'}$. The states q_1 and q_2 ensure that only *complete* b^n-sequences are read between q and q'.

For pairs of states and a sign the set $G^s_{q,q'}$ is empty if there is some $q'' \in Q$ such that either $\delta(q'', b) = q$ or $\delta(q', b) = q''$. See Figure 3.2 for an illustration of $G^s_{q,q'}$.

Lemma 3.4.3. *Let $M = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(M) \subseteq L_{\text{QBF}}$, let $q, q' \in Q$ be states and let $s \in \{\oplus, \ominus\} \subseteq \Gamma$ be a sign symbol. If there is an $n_1 \in G^s_{q,q'}$ such that $n_1 \geq |Q|$ and n_1 is odd, then there are infinitely odd n in $G^s_{q,q'}$.*

Proof. Assume there is an odd $n_1 \geq |Q|$ in $G^s_{q,q'}$. Since $n_1 \in G^s_{q,q'}$ we have that $\delta(q, b^{n_1}) = q'$. The path from q to q' with the label b^{n_1} must contain some loop, say of length $m < n_0$, because $n_1 \geq |Q|$. Therefore, for all $i \in \mathbb{N}$ we obtain that $n + mi \in G^s_{q,q'}$. This particularly holds for $i' = 2i$, hence $n + mi' \mod 2 \equiv n + m2i \mod 2 \equiv n \mod 2 \equiv 1$, which consequently proofs the claim. □

Lemma 3.4.4. *Let $M = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(M) \subseteq L_{\text{QBF}}$. Let $Z \subseteq Q^2$ be the set of all pairs of states between which complete quantification depth are read:*

$$Z := \left\{(q, q') \in Q^2 \mid \exists s \in \{\oplus, \ominus\} : G^s_{q,q'} \neq \emptyset\right\}$$

Let $\mathbb{N}_E := \{n \in \mathbb{N} \mid n \equiv 0 \mod 2\}$ be the set of even natural numbers. For all $s \in \{\oplus, \ominus\} \subseteq \Gamma$ and for all $S \subseteq Z$ if

$$\mathbb{N}_E \cap \bigcap_{(q,q') \in S} G^s_{q,q'} \neq \emptyset \ ,$$

then there is an even $n_0 \in \bigcap_{(q,q') \in S} G^s_{q,q'}$ such that $n_0 \leq 2^{|Q|^{|Q|^2}}$.

Proof. For $G^s_{q,q'}$ let $M^s_{q,q'} = (Q, \Gamma, \delta, q, \{q'\})$ be a deterministic finite automaton constructed from M by setting q to the initial state and q' to the single final state. Then for all $n \in \mathbb{N}$ we have

$$n \in G^s_{q,q'} \iff b^n \in L(M^s_{q,q'}) \ ,$$

and therefore

$$\mathbb{N}_E \cap \bigcap_{(q,q')\in S} G^s_{q,q'} \neq \emptyset \iff \{b^n \mid n \in \mathbb{N}_E\} \cap \bigcap_{(q,q')\in S} L(M^s_{q,q'}) \neq \emptyset \ .$$

Since all $M^s_{q,q'}$ are finite automata and $\{b^n \mid n \in \mathbb{N}_E\}$ is regular (and can be recognized by a DFA with two states) we can build for each $S \subseteq Z$ a deterministic finite automaton M_S recognizing $\bigcap_{(q,q')\in S} L(M^s_{q,q'})$ with the standard intersection construction. Intersections of finite automata are built with *Cartesian products* of the states, wherefore M_S operates as the state set $2 \times \underbrace{Q \times \cdots \times Q}_{|S|} = 2 \cdot Q^{|S|}$.

Since $|S| \leq |Z| \leq |Q|^2$ the number of states in M_S is at most $2|Q|^{|Q|^2}$. Now, if $\mathbb{N}_E \cap \bigcap_{(q,q')\in S} G^s_{q,q'} \neq$ then $\{b^n \mid n \in \mathbb{N}_E\} \cap \bigcap_{(q,q')\in S} L(M^s_{q,q'}) \neq \emptyset$ and therefore $L(M_S) \neq \emptyset$. If $L(M^s_{q,q'}) \neq \emptyset$, there is an n_0 such that b^{n_0} is accepted by $M^s_{q,q'}$ along a simple path, and therefore $n_0 \leq 2|Q|^{|Q|^2}$ and n_0 is an even number in $\bigcap_{(q,q')\in S} G^s_{q,q'}$. □

Lemma 3.4.4 states that there is an upper bound of $2|Q|^{|Q|^2}$ for the quantification depth when testing if variables of the same quantification depth can be read at multiple positions.

Definition 3.4.5. Let $M = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(M) \subseteq L_{\mathrm{QBF}}$ and $Q = \{q_0, q_1, \ldots, q_{n-1}\}$. Let $G^{\Delta,s}_{q,q'} := G^s_{q,q'} \cap \{i \in \mathbb{N} \mid i \geq |Q|\}$ be the complete quantification depths in $G^s_{q,q'}$ that must read along a loop in M. We define the *level* of $G^{\Delta,s}_{q,q'}$ as the set of all finite subset of $G^{\Delta,s}_{q,q'}$ that needs to be regarded when searching for true quantified Boolean formula in accepted by M.

If there are existential quantification depths in $G^{\Delta,s}_{q,q'}$ then we can establish a unique quantification depth. Otherwise we will use the upper bound of Lemma 3.4.4 to cap the quantification depth at $2|Q|^{|Q|^2}$ for universal variables. We use the levels computed with Algorithm 4 to

Algorithm 4: Algorithm to compute level($G^{\Delta,s}_{q,q'}$). Each level is a finite set. If $G^{\Delta,s}_{q,q'}$ contains an existential level it is uniquely chosen. Otherwise, all depths up to the upper bound are assigned.

Data: $M = (Q, \Gamma, \delta, q_0, F)$, a deterministic finite automaton with $L(M) \subseteq L_{\mathrm{QBF}}$ and $Q = \{q_0, q_1, \ldots, q_{n-1}\}$.
Result: level($G^{\Delta,s}_{q,q'}$) for all pairs of states q, q' and signs s.

```
uniqueExistentialLevel ← 2|Q|^{|Q|^2} + 1
for i ← 0, ..., n − 1 do
    for j ← 0, ..., n − 1 do
        for s ← ⊕, ⊖ do
            if G^{Δ,s}_{q_i,q_j} ∩ {i ∈ ℕ | i ≡ 1 mod 2} ≠ ∅ then
                level(G^{Δ,s}_{q_i,q_j}) ← min(G^{Δ,s}_{q_i,q_j} ∩ {odd i | i ≥ uniqueExistentialLevel})
                uniqueExistentialLevel ← level(G^{Δ,s}_{q_i,q_j}) + 2
            end
            level(G^{Δ,s}_{q_i,q_j}) ← G^{Δ,s}_{q_i,q_j} ∩ {even i | 2 ≤ i ≤ 2|Q|^{|Q|^2}}
        end
    end
end
```

construct for a given automaton M a modified deterministic finite automaton that recognizes a true quantified Boolean formula if and only if M does.

Definition 3.4.6. Let $M = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(M) \subseteq L_{\text{QBF}}$. We define the modified deterministic finite automaton $\text{restrict}(M) = (Q', \Gamma, \delta', q_0, F)$ as follows: The set of transitions δ' is constructed by first removing any transitions of the form $\delta(q, b) = q'$ for any $q, q' \in Q$. In particular any b-loops in M are removed in process. Now, for each $m \in \text{level}(G^{\Delta,s})_{q,q'}$ and $m \in G^s_{q,q'} \cap \{1, \ldots, |Q| - 1\}$ we add $m - 1$ states $q_1, \ldots, q_{m-1}$ into the state set Q' and introduce transitions $\delta(q, b) = q_1$, $\delta(q_i, b) = q_{i+1}$ for $i = 1, \ldots, m - 2$ and $\delta(q_{m-1}, b) = q'$.

Lemma 3.4.7. *Let $M = (Q, \Gamma, \delta, q_0, F)$ be deterministic finite automaton. Then $L(M) \cap L_{\text{TQBF}} \neq \emptyset \iff L(\text{restrict}(M)) \cap L_{\text{TQBF}} \neq \emptyset$.*

Proof. **"⇒"** If $L(M) \cap L_{\text{TQBF}} \neq \emptyset$ then there is a $w \in L(M)$ such that $\text{dec}(w)$ evaluates to true, i.e. $w \in L_{\text{TQBF}}$. There is an $n \in \mathbb{N}$ such that w can be separated into a list of $3n$ literals, each in $\{\oplus, \ominus\}b^+a^+$:

$$s_1 b^{\beta_1} a^{\alpha_1}, s_2 b^{\beta_2} a^{\alpha_2}, \ldots, s_{3n} b^{\beta_{3n}} a^{\alpha_{3n}} \quad ,$$

where $s_i \in \{\oplus, \ominus\}$, and $\beta_i, \alpha_i \in \mathbb{N}$ for $i = 1, \ldots, 3n$. We denote $s_i b^{\beta_i} a^{\alpha_i}$ as lit_i. Now, consider the literals lit_i of w where $\beta_i \geq |Q|$, i.e. the factor b^{β_i} of lit_i takes loop while being read in M. Let $q_i, q_i' \in Q$ be the two states in M between which b^{β_i} is read. We give substitution for such literals lit_i in a fixed order (say successively from $i = 1$ to $3m$), distinguishing the following four cases:

1. If β_i is odd (and therefore $\text{dec}(lit_i)$ is an existentially quantified literal), then following Lemma 3.4.3 there are infinitely many odd $n \in \mathbb{N}$ in $G^{s_i}_{q_i,q_i'}$. For β_i' defined as

$$\beta_i' := \min \left\{ n \in G^{s_i}_{q_i,q_i'} \;\middle|\; \begin{array}{lll} n \text{ is odd}, & & \\ n > \beta_j & \text{for } j = 1, \ldots, 3n & \text{and} \\ n > \beta_l' & \text{for } l = 1, \ldots, i-1 & \end{array} \right\} \quad ,$$

   we construct the substituted literal $lit_i' = s_i b^{\beta_i'} a^{\alpha_i}$. Since the quantification depth β_i' is unique to the literal lit_i' and the variable in it is existentially quantified, it can be evaluated to true without interfering with the validity of other literals.

2. If β_i is even (and hence $\text{dec}(lit_i)$ is a universally quantified literal) and there is some odd $o \in G^{s_0}_{q_i,q_i'} \cap \{|Q|, |Q| + 1, \ldots, 2|Q| - 1\}$ then lit_i can be substituted by a literal containing an existentially quantified variable in the same way as in case 1, yielding $lit_i' = s_i b^{\beta_i'} a^{\alpha_i}$.

3. If β_i is even and $\beta_i \leq 2|Q|^{|Q|^2}$ and $G^{s_0}_{q_i,q_i'} \cap \{|Q|, |Q| + 1, \ldots, 2|Q|\} = \emptyset$ we leave lit_i unchanged.

4. If β_i is even and $\beta_i > 2|Q|^{|Q|^2}$ and $G^{s_0}_{q_i,q_i'} \cap \{|Q|, |Q| + 1, \ldots, 2|Q|\} = \emptyset$ we identify all j for $j = 1, \ldots, 3m$ with $\beta_j = \beta_i$ and pairs of states (q_j, q_j') such that the b-factor in β_j is read between q_j and q_j' (i.e. $\beta_j \in G^{s_j}_{q_j,q_j'}$). Let J be the set of all such indices j. Since $\beta_i \in \bigcap_{j \in J} G^{s_j}_{q_j,q_j'}$ the intersection is non-empty and following Lemma 3.4.4 there is some even $n_0 \leq 2|Q|^{|Q|^2}$ in $\bigcap_{j \in J} G^{s_j}_{q_j,q_j'}$. For each $j \in J$ substitute lit_j by $lit_j' = s_j b^{n_0} a^{\alpha_j}$.

Let w' be the result of the application of the above described substitutions to w. Clearly, we still have $w' \in L(M)$ and following Lemma 3.4.1 all substitutions made preserve the validity of w. Furthermore, every level of universal quantification is lower than $2|Q|^{|Q|^2}$ and (ignoring ordering of the states of M) existential levels above $2|Q|^{|Q|^2}$ are assigned uniquely, which yields that $w' \in L(\text{restrict}(M))$.

"$\Leftarrow$" By definition restrict(M) recognizes a (proper) subset of M. In particular every $w \in L(\text{restrict}(M))$ such that dec(w) is a true quantified Boolean formula is also also in $L(M)$.

□

Theorem 3.4.8. *For any regular languages R it is decidable if $R \cap L_{k\text{-TQBF}} = \emptyset$.*

Proof. Let M be the minimal deterministic finite automaton with $L(M) = R$. In restrict(M), let k be the length of the longest (simple) path of b-edges. Following Theorem 3.3.20 we can decide if $L(\text{restrict}(M)) \cap L_{k\text{-TQBF}} = \emptyset$. Lemma 3.4.7 states that $L(\text{restrict}(M)) \cap L_{\text{TQBF}} = \emptyset \iff L(M) \cap L_{\text{TQBF}} = \emptyset$. Therefore, we can decide whether $R \cap L_{k\text{-TQBF}} = \emptyset$. □

3.5 Hiding the difficulty of int_{Reg}

The previous section showed the existence of PSPACE-complete problems for which the intersection of regular intersection is decidable. Since all (known) families of formal languages lie within NP we have languages (most probably) outside of NP for which int_{Reg} is decidable.

In this section we show that for every decidable language L there is an AC^0-many-one equivalent language L' for which the regular intersection emptiness is decidable. Since (almost) all complexity classes (disregarding non-uniform circuit classes) are decidable and all complexity classes *above* AC^0 are closed under AC^0-reductions, this implies that all of these classes contain complete problems for which int_{Reg} is decidable. This, of course, means that int_{Reg} is not an adequate characterization of families of formal languages.

The technique used to obtain this result is similar to the *dense completeness* results from Krebs and Lange [KL12a] which we call *hiding*.

It preserves the complexity of a language but makes it very easy to decide the emptiness of regular intersection. Hiding the difficulty of int_{Reg} for a language L is done in two steps. First, we expand the words in L in a highly redundant manner. We then enrich the resulting language with all misshaped words. Each infinite regular set has then (independent of L) a non-empty intersection with the set of misshaped words. Therefore, we only have to consider intersections with finite sets, which equals testing finitely many membership queries for L. If L is decidable so are these finitely many tests, which together results in decidability of the emptiness of regular intersection.

Definition 3.5.1. For $\Gamma = \{0, 1\}$ let $T\colon \Gamma^* \to \Gamma^*$ be the mapping defined as

$$T(w) = w10^{|w|} \ ,$$

and for some language $L \subseteq \Gamma^*$ let

$$T(L) = \{T(w) \mid w \in L\} \ ,$$

be the application of T to each word in L. Let

$$L_{\text{bad}} := \Gamma^* \setminus T(\Gamma^*)$$

be the set of words not in the image of T. We define the *int_{Reg}-hiding* function $\phi\colon 2^{\Gamma^*} \to 2^{\Gamma^*}$ as

$$\phi(L) = T(L) \cup L_{\text{bad}} \ .$$

Remark 3.5.2. L_{bad} is recognizable by a non-deterministic 1-counter automaton, but is not contained in DOI^n for any fixed n.

Lemma 3.5.3. *If L is decidable and $L \neq \emptyset$ then $\phi(L) \leq_m^{\mathsf{AC}^0} L \leq_m^{\mathsf{AC}^0} \phi(L)$.*

Proof. $L \leq_m^{\mathsf{AC}^0} \phi(L)$ We first construct a circuit family C_n^T with n inputs and $2n+1$ outputs for the mapping $T\colon \{0,1\}^n \to \{0,1\}^{2n+1}$. For an input word $w = w_1 \dots w_n$ the first n outputs are wired to the input gates in a *linear* fashion (i.e. w_i to the i-th output). The last $n+1$ outputs are only dependent on the input length but not the w_i. All output bits are only dependent on a constant number of inputs, which means that T is an NC^0 function and thus an AC^0 function. For $w \in \{0,1\}^*$ the output of the application of $C_{|w|}^T(w)$ on w is $w01^{|w|}$. If $w \in L$ then $w01^{|w|} \in T(L)$ and therefore $C_{|w|}^T(w) \in \phi(L)$. If otherwise $w \notin L$ then $w01^{|w|} \notin T(L)$. But since $w01^{|w|} \in T(\{0,1\}^*)$ we have $w01^{|w|} \notin L_{\text{bad}}$ and therefore $w01^{|w|} \notin \phi(L)$.

$\phi(L) \leq_m^{\mathsf{AC}^0} L$ Since $L \neq \emptyset$ there is some $\hat{w} \in \Gamma^*$ with $\hat{w} \in L$. Define the (sub-)circuit $C_n^{T^{-1}}$ that behaves as follows: For $n = 2m+1$ for some $m \in \mathbb{N}$ test for an input word $w = w_1 \dots w_{2m+1}$ if $w_{m+1} = 0$ and $w_{m+2} = w_{m+3} = \dots = w_{2m+1} = 1$ and returns 1 on the single output gate. If the input has no 01^m-suffix or there is no such $m \in \mathbb{N}$, the circuit $C_n^{T^{-1}}$ returns 0. Let the circuit $C_n^{\phi^{-1}}$ now test for an input of length n if AC^0 circuit $C_n^{T^{-1}}$ outputs 1, and if so returns the first m bits of the input for $n = 2m+1$. Otherwise, return the hard-wired (and independent from the input) word $\hat{w}$. Therefore, $C_n^{\phi^{-1}}$ is the following mapping:

$$w_1 \dots w_m \dots w_n \mapsto \begin{cases} w_1 \dots w_m & \text{if } w_1 \dots w_m = T(w_1 \dots w_n) \ , \\ \hat{w} & \text{otherwise} \ . \end{cases}$$

Clearly $C_n^{\phi^{-1}}$ can be constructed with constant depth and hence an AC^0-circuit. For $w \in \phi(L)$ we have either $w = w_1 \dots w_m 01^m$ or $w \in L_{\text{bad}}$. If $w = w_1 \dots w_m 01^m$ then $C_n^{T^{-1}}(w) = 1$ which implies $C_n^{\phi^{-1}}(w) = w_1 \dots w_m$ with $w_1 \dots w_m \in L$. For $w \in L_{\text{bad}}$ we have $C_n^{T^{-1}}(w) = 0$ and thus $C_n^{\phi^{-1}}(w) = \hat{w}$, which is by assumption in L. If $w \notin \phi(L)$, then $w \in T(\{0,1\}^*) \setminus T(L)$, and hence $w = w_1 \dots w_m 01^m$. Therefore we have $C_n^{T^{-1}}(w) = 1$ and $C_n^{\phi^{-1}}(w) = w_1 \dots w_m$ with $w_1 \dots w_m \notin L$.

□

Lemma 3.5.4. *If L is decidable then $int_{\mathsf{Reg}}(\phi(L))$ is decidable.*

Proof. Let $R \subseteq \{0,1\}^*$ be a given regular language for which we have to decide whether $\phi(L) \cap R = \emptyset$. We make the following case distinction to decide the emptiness of intersection:

R finite Since L is decidable and L and $\phi(L)$ are AC^0-many-one equivalent, $\phi(L)$ is decidable, too. Thus, to decide whether $L \cap R = \emptyset$, test for each $w \in R$ if $w \in \phi(L)$. The intersection is non-empty, if we have at least one $w \in R$ that $w \in \phi(L)$.

R infinite We show that each infinite regular language R has a non-empty intersection with $L_{\text{bad}} \subseteq \phi(L)$. Assume that $R \cap L_{\text{bad}} = \emptyset$. Then $R \subseteq \{0,1\}^* \setminus L_{\text{bad}} = T(\{0,1\}^*)$. This

implies for all $w \in R$ that $w = u01^{|u|}$ for some $u \in \{0,1\}^*$. Since R is infinite, for every $n \in \mathbb{N}$ there is some $w \in R$ with $|w| \geq n$. We can now use a standard pumping argument to pump up either the u-part or the 1-block of a large enough w to obtain a word that does not have the form required by $T(\{0,1\}^*)$. Thus, $R \not\subseteq T(\{0,1\}^*)$ and therefore $R \cap L_{\text{bad}} \neq \emptyset$ which yields independently of L that $R \cap \phi(L) \neq \emptyset$ if R is infinite.

□

Theorem 3.5.5. *Let $L \subseteq \{0,1\}^*$ be a non-empty decidable language. Then, there is a decidable language L' with $L \equiv_{\mathsf{AC}^0} L'$ and $int_{\mathsf{Reg}}(L')$ is decidable.*

Proof. Let $L' = \phi(L)$. If L is decidable, following Lemma 3.5.3 $L \equiv_{\mathsf{AC}^0} L'$ and L' is also decidable. Lemma 3.5.4 yields the decidability of $int_{\mathsf{Reg}}(L')$, which together proves the claim. □

Since all complexity classes that include AC^0 are closed under (DLOGTIME-uniform) AC^0 reductions, we can construct complete problems for any of these classes for which int_{Reg} is decidable by applying Theorem 3.5.5 to arbitrary complete languages. Thus, in order to describe a family of formal languages the decidability of the intersection emptiness with regular sets might be a property that is necessary, but not sufficient.

3.6 AFL-stability of the int_{Reg} property

In this section we show that the set of languages for which the intersection emptiness with regular sets is closed under the full AFL-operations. In context of the formality question and the result of Theorem 3.5.5 this means that every complexity class (that includes AC^0) contains complete languages for which the emptiness of regular language intersection is decidable and the AFLs of these languages *inherit* this property too.

Theorem 3.6.1. *Let L be a language for which $int_{\mathsf{Reg}}(L)$ is decidable. Then, the emptiness of regular intersection is decidable for the full AFL of L, i.e. $int_{\mathsf{Reg}}(\mathsf{fullAFL}(L))$ is decidable.*

Proof. To show that the int_{Reg}-property is decidable for $\mathsf{fullAFL}(L)$ we need to show that int_{Reg} is decidable for every AFL-operation, namely union, intersection with regular languages, homomorphisms, inverse homomorphisms, and the Kleene star. Assume that $int_{\mathsf{Reg}}(L)$, $int_{\mathsf{Reg}}(L_1)$ and $int_{\mathsf{Reg}}(L_2)$ is decidable.

union For a given regular language R the intersection $R \cap (L_1 \cup L_2)$ is empty if and only if the intersection with both languages individually is empty. Since both, $int_{\mathsf{Reg}}(L_1)$ and $int_{\mathsf{Reg}}(L_2)$ is decidable, so is $int_{\mathsf{Reg}}(L_1 \cup L_2)$.

intersection with regular languages Consider the intersection of $L \cap R$ for a regular language R. Since $int_{\mathsf{Reg}}(L)$ is decidable, intersection of sets is associative and commutative and regular languages are closed under intersection $int_{\mathsf{Reg}}(L \cap R)$ is decidable.

Kleene star We show that it is decidable to test if $L^* \cap R = \emptyset$ for a given regular set R. Let $M = (Q, \Gamma, \delta, q_0, F)$ be the minimal deterministic finite automaton recognizing R. We call a sequence of states $p = (q_{t_0}, \ldots, q_{t_n})$ in M *simply accepting* if

- $q_{t_0} = q_0$, it starts in the initial state,

- $q_{t_n} \in F$, ends in a final state,
- $0 \leq i < j \leq n \Rightarrow q_{t_i} \neq q_{t_j}$, that is p is loop-free.

For a simply accepting path $p = (q_{t_0}, \ldots, q_{t_n})$ we construct n new deterministic finite automata in the following way:

For $1 \leq i \leq n$ define $M_i = (Q, \Gamma, \delta, q_{t_i}, q_{t_{i+1}})$ as the automaton recognizing precisely the words $w \in \Gamma^*$ such that M transitions from q_{t_i} to $q_{t_{i+1}}$ when reading w. In case $\lambda \in L(M)$, we consider the additional automaton $M_\lambda = (Q, \Gamma, \delta, q_0, \{q_0\})$. Note, that the number of simply accepting paths is finite for every DFA. Let P be the finite set of all simply accepting paths in M. Now, we have $L^* \cap R \neq \emptyset$ if and only if

$$\bigvee_{(q_{t_0},\ldots,q_{t_n})\in P} \left(\bigvee_{i=0}^{n-1} L \cap L(M_i) \neq \emptyset \right) \ ,$$

which is decidable, since $int_{\mathsf{Reg}}(L)$ is decidable and we only make finitely many intersection emptiness tests with regular languages.

homomorphism and inverse homomorphism Let R be a given regular language and h be a homomorphism. Let M, N be arbitrary sets and $f\colon M \to N$ be any mapping between them. Note that for all $A \subseteq M$ and $B \subseteq N$ we have the following equivalence:

$$f(A) \cap B \neq \emptyset \iff A \cap f^{-1}(B) \neq \emptyset \ .$$

We can use the previous equivalence to decide $int_{\mathsf{Reg}}(h(L))$ and $int_{\mathsf{Reg}}(h^{-1}(L))$ in the following way:

$$\begin{array}{lcll} h(L) \cap R \neq \emptyset & \iff & L \cap h^{-1}(R) \neq \emptyset & \text{and} \\ h^{-1}(L) \cap R \neq \emptyset & \iff & L \cap h(R) \neq \emptyset \ . & \end{array}$$

Regular languages are closed under both, homomorphisms and inverse homomorphisms. Therefore, $int_{\mathsf{Reg}}(h(L))$ and $int_{\mathsf{Reg}}(h^{-1}(L))$ are decidable.

concatenation The closure under concatenation is a consequence of the closures under union, intersection with regular languages, (inverse) morphisms and the Kleene star, as noted in [HU79, Theorem 11.6]

□

The closure under union uses the distributivity of intersection and union. Note, that there is no similar technique to show that the set of languages which int_{Reg} is decidable is not closed under intersection.

Lemma 3.6.2. *There are L_1 and L_2 for which $int_{\mathsf{Reg}}(L_1)$ and $int_{\mathsf{Reg}}(L_2)$ is decidable, but $int_{\mathsf{Reg}}(L_1 \cap L_2)$ is undecidable.*

Proof. Assume that $L_1, L_2 \subseteq \Gamma^*$ are context-free languages. CFL are closed under the intersection with regular languages and the emptiness is decidable. For the regular language Γ^* we have

$$\Gamma^* \cap (L_1 \cap L_2) = \emptyset \iff (L_1 \cap L_2) = \emptyset \ .$$

which is undecidable for context-free languages and therefore yields the undecidability of $int_{\mathsf{Reg}}(L_1 \cap L_2)$. □

3.7 Complexity classes and int_{Reg}

Definition 3.7.1. Let $\mathcal{C}$ be a complexity class which is closed under AC^0-reductions. We denote the set of all languages in $\mathcal{C}$ for which int_{Reg} is decidable as

$$\mathcal{C}_{int_{\mathsf{Reg}}} := \{L \in \mathcal{C} \mid int_{\mathsf{Reg}}(L) \text{ is decidable}\} \ .$$

Proposition 3.7.2. *If $\mathcal{C}$ and $\mathcal{D}$ are complexity classes, closed under AC^0-many-one and $\mathcal{C} \subseteq \mathcal{D}$ then*

$$\mathcal{C} = \mathcal{D} \iff \mathcal{C}_{int_{\mathsf{Reg}}} = \mathcal{D}_{int_{\mathsf{Reg}}} \ .$$

Proof. "$\Rightarrow$" If $\mathcal{C} = \mathcal{D}$ then $L \in \mathcal{C}$ if and only $L \in \mathcal{D}$. In particular this holds for all $L \in \mathcal{C}$ for which $int_{\mathsf{Reg}}(L)$ is decidable which implies that $\mathcal{C}_{int_{\mathsf{Reg}}} = \mathcal{D}_{int_{\mathsf{Reg}}}$.

"$\Leftarrow$" Assume that $\mathcal{C}_{int_{\mathsf{Reg}}} = \mathcal{D}_{int_{\mathsf{Reg}}}$. Let $D \in \mathcal{D}$ be complete for $\mathcal{D}$ under AC^0-reductions. If $D \neq \emptyset$ then according to Lemma 3.5.3 there is a $\phi(D) \in \mathcal{D}_{int_{\mathsf{Reg}}}$ with $\phi(D) \equiv_{\mathsf{AC}^0} D$. Now, $\mathcal{C}_{int_{\mathsf{Reg}}} = \mathcal{D}_{int_{\mathsf{Reg}}}$ implies that $\phi(D) \in \mathcal{C}_{int_{\mathsf{Reg}}} \subseteq \mathcal{C}$, i.e. $\mathcal{C}$ contains a $\mathcal{D}$-complete language from which we can follow that $\mathcal{C} = \mathcal{D}$.

□

A direct application of Proposition 3.7.2 yields for instance $\mathsf{P} = \mathsf{NP}$ if and only if $\mathsf{P}_{int_{\mathsf{Reg}}} = \mathsf{NP}_{int_{\mathsf{Reg}}}$ and $\mathsf{NP} = \mathsf{PSAPCE}$ if and only if $\mathsf{NP}_{int_{\mathsf{Reg}}} = \mathsf{PSPACE}_{int_{\mathsf{Reg}}}$.

3.8 Summary

In this chapter we disproved our conjecture that families for formal languages can be distinctly characterized with the int_{Reg}-property.

We showed that there are natural PSPACE-complete languages, as well as languages that are complete for each level of the Polynomial Hierarchy for which int_{Reg} is decidable. These languages were encoding of true quantified Boolean formulae. By restricting the alternation depth of the formulae we obtained the complete languages for the polynomial hierarchy.

We applied a technique called *Hiding* to show that for every decidable language L there is a language L' with *same* complexity such that $int_{\mathsf{Reg}}(L')$ is decidable. Therefore, every complexity class contains complete problems for which int_{Reg} is decidable.

Furthermore, we showed that the full AFL of languages for which int_{Reg} is decidable also has a decidable regular intersection emptiness.

This means, even though the decidability of int_{Reg} seems to be a *necessary* property a family of formal languages should have, it is not a *sufficient* property for a class of languages to be formal.

CHAPTER 4

Protocol languages

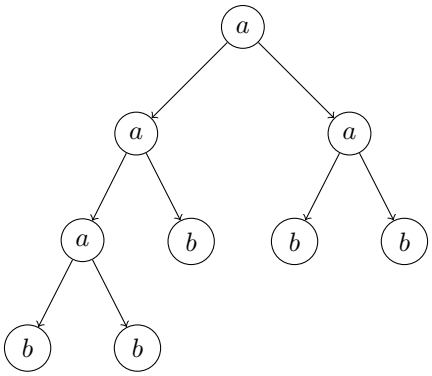
In this chapter we introduce the notion of *protocol languages*. These protocol languages will express in some sense the *data-structure* underlying a typical family of formal languages. This will be expressed via logical formulae.

Grammars seem to be one of the foundations for families of formal languages. The data-structure of a grammar is recognizable in the derivation process: In each derivation step there is a list of non-terminals which have yet to be processed. Once this list is worked through and emptied the derivation process is finished and we are left with the word. Since the data-structure of grammars seem to be found in the derivation process, we consider the *derivation language* of a grammar.

When analyzing regular grammars the data-structure is unsurprisingly simple. In each step of the process there can only be a single non-terminal which has to be handled. This observation concurs with the fact that the *machine model* of regular languages are finite automata, which are equipped with only a finite amount of memory in form of their state set.

For context-free grammars the data-structure becomes a bit more involved and cannot be managed with *finite memory* anymore. Since the rules in context-free grammars allow arbitrary (but finite) lengths on the right hand side the list of non-terminals which are added to the data-structure might increase unboundedly. We give an *order* to the data-structure by considering *left-derivations*, e.g. if we use some rule $B \to CD$ and hence add C and D to the front list of *active* non-terminals, the next derived non-terminal would be C.

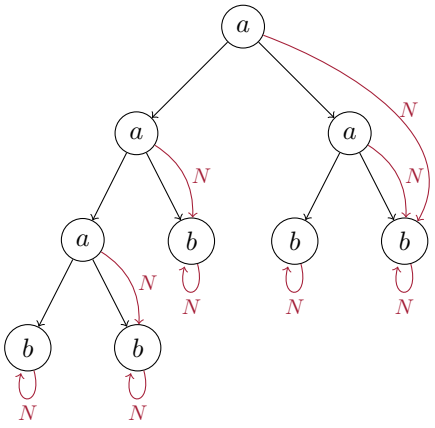
To simplify a first analysis we consider a very stripped-down context free grammar in Chomsky normal form $G = (\{S\}, \{t\}, \{S \to SS \mid t\}, S)$ with a single terminal and non-terminal symbol. The derivation tree of each $w \in L(G)$ is a full binary tree. Consider the exemplary derivation tree for $ttttt \in L(G)$ where the rule $S \to SS$ is abbreviated by a and $S \to t$ is abbreviated by b. Dealing with left derivations means that we recursively first handle the left sub-tree before the right sub-tree, i.e. we consider *pre-order-traversals* of derivation trees.



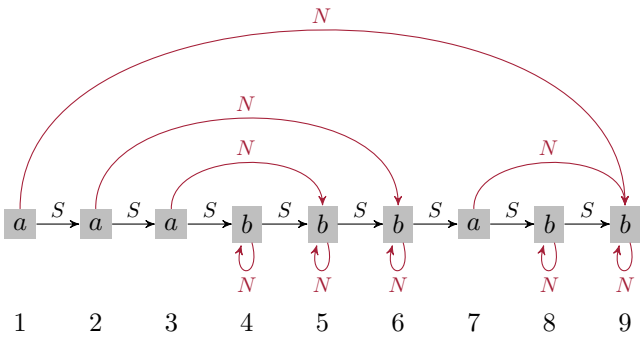
The pre-order traversal of the derivation tree of $w = ttttt$ yields the *derivation word* $d(w) = aaabbbabb$. Applying this to all words in $L(G)$ yields the language accepted by the grammar $G_Ł = (\{S\}, \{a, b\}\{S \to aSS \mid b\})$ which is known as the Łukasievicz language [Ber79]. We refer to the Łukasievicz language as Ł.

The usage of monadic second order logic will not be enough to describe Ł, since the languages definable in MSO are precisely the regular languages [Jul61, Elg61, Tra61], and Ł is a non-regular context-free language. Lautemann, Schwentick and Thérien [LST94] gave a logical characterization of context-free languages in form of a fragment of existential second order logic, with binary second order variables which define *nestings*, i.e. semantically constrained numerical relations M such that for all $(i, j) \in M$ we have $i < j$ and if $(i, j), (k, l) \in M$ and $i < k < j$ then $l < j$. As Ł is a context-free language is expressible with aforementioned logical fragment. But it would not reflect the highly deterministic structure of grammar derivations very well that the Łukasievicz language represents. We utilize a logical fragment which is built upon the idea of the formulae used by Lautemann et al. [LST94] to express the deterministic character of for instance derivation languages or other *protocol languages*. Our formulae use a fixed number of binary second order relations, which we require to be *uniquely induced* by each word that models the formula. We constrain these relations to behave like *pointers* which define the data-structure underlying the word, i.e. to be total mappings.

Assume, we had additional edges N in the derivation tree that would point to the *end* of its derivation. For our example $w = ttttt$ we would have the following:



This would yield the following the derivation word with additional structure on it.



This additional *pointer relation* N is unique for every $w \in Ł$. We give a formula for Ł, which uses the successor relation as well as the N relation. Since N is constrained to be a total mapping we use the notation $N(x) = y$ for $(x, y) \in N$.

$$w \in Ł \iff (w, N) \models \forall x : (Q_a(x) \iff N(x) = N(S(N(S(x))))) \wedge (Q_b(x) \iff N(x) = x)$$

We can identify all positions in a word that have the letter b (i.e. the terminating rule of the grammar) by the N-self-loop. There is an a on a position (i.e. a *bifurcation* rule which implies two sub-trees in the derivation tree) iff the N-pointer points to the same position as its right sub-tree. The right sub-tree itself begins after the left sub-tree ends.

As seen in this example, each letter in a word *implies* a definite sub-structure of the word. Vice versa, each sub-structure *implies* which letter has to be on a position. We call languages which have this *equivalence* between letters and uniquely determined structures *auto-generative* languages.

Madhusudan and Parlato [MP11] analyzed various automata on words with various auxiliary storage models in regard to the decidability of the emptiness problem. To this end, they encode the mechanisms of the (word-)automata Our use of additional N-pointers on words is therefore very similar to this approach. Madhusudan and Parlato then define graph acceptors, based on a concept called *tiling*, to decide the emptiness of the original word-language. Tiling was previously introduced more generally by Thomas [Tho91].

We adopt Thomas' idea of tiling for auto-generative languages. Each word in an auto-generative language is linked to a uniquely determined graph and therefore each auto-generative language defines a family of graphs which can be recognized with tiling arguments.

This chapter is structured as follows. We begin with the formal definition of the logical fragment used by us and introduce auto-generative languages. In Section 4.2 we fix our concept of tiling auto-generative languages and shrinking arguments on them, followed by our notion of *protocol languages*. We consider several logical extensions of auto-generative languages in Section 4.5. Section 4.6 addresses trio operations which are applied to auto-generative languages and their logical extensions. We give protocol languages for some well-known families of formal languages in Section 4.7 and finally, analyze the emptiness problem of various extensions of protocol languages in Section 4.11.

4.1 Auto-generative languages

In this section we define a logical fragment of binary second order logic we going to use through this chapter and will define auto-generative languages.

Definition 4.1.1 ($\mathsf{unEx}k\mathsf{FO}[S]$)**.** Let Γ be an alphabet. For $k \in \mathbb{N}$ we say that a formula Φ is in $\mathsf{unEx}k\mathsf{FO}[S]$ if there are k binary second order variables $N_1, \ldots, N_k$ and Φ is built from quantified first order variables, the successor relation S, letter predicates $(Q_\gamma)_{\gamma\in\Gamma}$, Boolean operations and the binary second order variables $N_1, \ldots, N_k$, but does not quantify any second order variable. We say a language $L \subseteq \Gamma^*$ is in $\mathsf{unEx}k\mathsf{FO}[S]$ if there is a $\Phi \in \mathsf{unEx}k\mathsf{FO}[S]$ such that for all words $w \in \Gamma^*$ we have

$$\begin{aligned} w \in L \iff w \models \\ !\exists N_1 \ldots !\exists N_k : & \bigwedge_{i=1}^{k} (\forall x \exists y : (x,y) \in N_i) \wedge \\ & \bigwedge_{i=1}^{k} (\forall x \forall y \forall z : ((x,y) \in N_i \wedge ((x,z) \in N_i) \Rightarrow y = z)) \\ & \wedge \Phi(N_1, \ldots, N_k) \end{aligned}$$

For readability reasons we omit the explicit quantification and restriction of the binary second order variables N_i and write $w \models \Phi$ for $w \in L$, or write $(w, N_1, \ldots, N_k) \models \Phi$.

This means if formula Φ is in $\mathsf{unEx}k\mathsf{FO}[S]$ it might use the binary second order variables like additional relational symbols. Hence the signature of Φ defining a language over Γ is $\langle (Q_\gamma)_{\gamma\in\Gamma}, S, N_1, \ldots, N_k\rangle$. Note, the binary second order variables N_i are uniquely quantified and hence are uniquely defined for each word in L. Each N_i is semantically constrained such that it maps each position x in a word to some other position y, i.e. the second order relations *behave* like a total functions on word positions. We often refer to N_i as *pointer relations*.

When defining formulae in $\mathsf{unEx}k\mathsf{FO}[S]$ we make a few notational adjustments for readability reasons: Let $x, x_0, x_1, \ldots, x_n, y, z$ be first order variables let N be a pointer relation.

1. We often write $N(x) = y$ instead of $(x, y) \in N$.
2. By $N(N(x)) = z$ we denote the formula $\exists y : N(x) = y \wedge N(y) = z$.
3. For some fixed $n \in \mathbb{N}$ we write $N^n(x_0) = x_n$ for the formula
$$\exists x_1 \cdots \exists x_n : \bigwedge_{i=1}^{n} N(x_{i-1}) = x_i \ .$$
4. We apply the same *functional* notation for the successor relation S on word positions. Instead of $(x, y) \in S$ we write $S(x) = y$.
5. We write $x = \max$ for the formula $\neg\exists y : S(x) = y$ and $x = \min$ for $\neg\exists y : S(y) = x$.
6. Let $\mathcal{R} = \{S, N_1, \ldots, N_k\}$ the set of relations in the signature for some formula Φ. Let $W = W_1 \cdots W_n \in \mathcal{R}^*$ and $V = V_1 \cdots V_m \in \mathcal{R}^*$. We write $W(x) = V(x)$ for the formula
$$\begin{aligned} \exists x_1 \cdots \exists x_n \exists y_1 \cdots \exists y_m : & W_1(x) = x_1 \wedge V_1(x) = y_1 \wedge \\ & \bigwedge_{i=2}^{n} W_i(x_{i-1}) = x_i \wedge \\ & \bigwedge_{j=2}^{m} V_j(y_{j-1}) = y_j \wedge \\ & x_n = y_m \end{aligned}$$

Intuitively speaking W and V define a path through the graph induced by $N_1, \ldots, N_k$ starting at position x. We write $W(x) = V(x)$ if the paths end at the same position.

Definition 4.1.2 (Auto-generative language)**.** Let Γ denote a finite alphabet. We call $P \subseteq \Gamma^*$ over Γ an *auto-generative language* if there is a formula $\Phi_P \in \mathsf{unEx}k\mathsf{FO}[(Q_\gamma)_{\gamma\in\Gamma}, S]$ with

$$\Phi_P = \forall x : \bigwedge_{\gamma\in\Gamma} (Q_\gamma(x) \Leftrightarrow \psi_\gamma(x)) \quad ,$$

where for each $\gamma \in \Gamma$ there is a sub-formula $\psi_\gamma(x)$. Each $\psi_\gamma(x)$ is a conjunction of terms $W(x_1) = V(x_2)$ or $W(x_2) \neq V(x_1)$ for $x_1, x_2 \in \{x, \min, \max\}$ and $W, V \in \{N_1, \ldots, N_k, S\}^*$ where $N_1, \ldots, N_k$ are the k binary second order variables *quantified* outside of Φ_P such that $P = \{w \in \Gamma^* \mid w \models \Phi_P\}$.

In an auto-generative language each sub-formula ψ_γ just uses information about positions in a word and can be used as a *substitute* for the letter predicates Q_γ. Further, each ψ_γ formula defines a certain substructure on a word. When defining some auto-generative language $P \subseteq \Gamma^*$ we can do this by giving a ψ_γ for each $\gamma \in \Gamma$. Note that for $\gamma \neq \gamma'$ we have that $\psi_\gamma \wedge \psi_{\gamma'}$ is *false*, since otherwise this would imply that two letter predicates would be true at the same position, i.e. there must not be two letters which *behave* the same way in an auto-generative language.

Example 4.1.3. We can now give the formal definition of the formula recognizing the Łukasievicz language by giving ψ_a and ψ_b formulae:

$$\psi_a(x) : N(x) = N(S(N(S(x)))) \ ,$$
$$\psi_b(x) : N(x) = x \wedge N(\min) = \max$$

Therefore, Ł is an auto-generative language in $\mathsf{unEx1FO}[S]$.

In Section 4.5 we will give logical extensions of a auto-generative languages which will be defined as conjunctions with the *shell* of an auto-generative language.

Definition 4.1.4 (Shell)**.** Let $P \subseteq \Gamma^*$ be an auto-generative language recognized by a formula

$$\Phi_P = \forall x \bigwedge_{\gamma\in\Gamma} \psi_\gamma(x) \iff Q_\gamma(x)$$

in $\mathsf{unEx}k\mathsf{FO}[(Q_\gamma)_{\gamma\in\Gamma}, S]$. We call the *shell* of a protocol language P the formula

$$\begin{aligned}\Phi_P^{shell} = \quad \exists N_1 \ldots \exists N_k : & \bigwedge_{i=1}^{k} (\forall x \exists y : (x,y) \in N_i) \wedge \\ & \bigwedge_{i=1}^{k} (\forall x \forall y \forall z : ((x,y) \in N_i \wedge ((x,z) \in N_i) \Rightarrow y = z)) \wedge \\ & \forall x \bigvee_{\gamma\in\Gamma}^{\bullet} \psi_\gamma(x)\end{aligned}$$

The shell formula of a protocol language does not test for any letters but constructs letter predicates implicitly with the N-structure.

Lemma 4.1.5. *Let P be an auto-generative language recognized by a formula $\Phi_P \in \mathsf{unEx}k\mathsf{FO}[S]$ and Φ_P^{shell} its shell. If $w \in P$ then for any letter a we have that $a^{|w|} \models \Phi_P^{shell}$.*

Proof. If $w \in P$ then there are $N_1, \ldots, N_k$ such that $(w, N_1, \ldots, N_k) \models \Phi_P$ which is of the form $\forall x \bigwedge_{\gamma\in\Gamma} \psi_\gamma(x) \iff Q_\gamma(x)$. Since every position of w is quantified and at each position precisely one letter predicate can evaluate to true, at each position precisely one ψ_γ formula is true. Therefore, these particular binary second order variables $N_1, \ldots, N_k$ can be picked such that $a^{|w|} \models \Phi_P^{shell}$. Note that if there are several $w \in L$ of the same length the binary second order variables are not unique. □

The extension of auto-generative languages by ψ_γ formulae with *disjunction* yields letters with some *ambiguity* in their behavior. Intuitively speaking, Proposition 4.1.6 states that for every such language P there is an auto-generative language I where each letter is unambiguous and P is the length-preserving morphic image of I.

Proposition 4.1.6. *Let $P \subseteq \Gamma^*$ be a language recognized by a formula $\Phi_P \in \mathsf{unEx}k\mathsf{FO}[S]$ with*

$$\Phi_P = \forall x : \bigwedge_{\gamma\in\Gamma} (Q_\gamma(x) \Leftrightarrow \psi_\gamma(x)) \quad ,$$

where each $\psi_\gamma(x)$ is Boolean combination of statements of the form $W(x) = V(y)$ or $W(x) \neq V(y)$ for $W, V \in \{N_1, \ldots, N_k, S\}^*$. Then there is an alphabet Y and an auto-generative language $I \subseteq Y^*$ recognized by a formula Φ_I with and a length preserving morphism $h\colon Y \to \Gamma$ that is injective over I such that $h(I) = P$.

Proof. Assume without loss of generality that each $\psi_\gamma(x)$ is in exclusive disjunctive normal form:

$$\psi_\gamma(x) = \dot{\bigvee_{i=1}^{l_\gamma}} C_i^\gamma(x)$$

where each of the l_γ many clauses C_i^γ is a conjunctions of statements $W(x) = V(y)$ or $W(x) \neq V(y)$ for $W, V \in \{N_1, \ldots, N_k, S\}^*$. We define an alphabet Y with $|Y| = \sum_{\gamma\in\Gamma} l_\gamma$ and the length preserving morphism $h\colon Y \to \Gamma$ such that $h(y_i^\gamma) = \gamma$ for $i = 1, \ldots, l_\gamma$ for all $\gamma \in \Gamma$. Then I is recognized by the formula

$$\Phi_I = \forall x : \bigwedge_{y_i^\gamma \in Y} \left(Q_{y_i^\gamma}(x) \Leftrightarrow C_i^\gamma(x) \right) \quad .$$

and $h(I) = P$. □

4.2 Tiling auto-generative languages

Each word of an auto-generative language models a formula in $\mathsf{unEx}k\mathsf{FO}[S]$ and therefore uniquely implies k binary second order relations. Thus, we can view each such word as a graph, where the binary second order relations introduce additional edges to the successor. Therefore, each auto-generative (word) language implicitly defines a family of graphs.

In this section we introduce the concept of *tiling* which is a way to finitely describe (possibly infinite) sets of graphs according to Thomas [Tho91]. We consider a finite set of *tiles* which are directed graph with edge- and node-labels.

Intuitively speaking a graph G can be recognized by a set of tiles if we can place instances of tiles onto G such that G is covered and the tiles overlap *coherently*. We extend the notion of tiling by introducing *constraints* which require or forbid a certain number of instances of a tile. This is then used to define a *shrinking property* for auto-generative languages: The idea is that if the graph of some word is long enough we can remove a part of the graph and therefore *shrink* the reflected word to obtain a shorter one still in the language.

Definition 4.2.1 (Tile)**.** Let Γ be a finite alphabet and let $\mathcal{L} = \{S, N_1, \ldots, N_k\}$ be a set of labels. We call the directed graph $t = (V, E, \tau_{\text{core}}, \tau_{\text{port}})$ a *tile* where

1. $V = V_{\text{core}} \dot\cup V_{\text{port}}$ is a set of nodes divided into disjoint sets V_{core} and V_{port}
2. $E = E_S \cup E_{N_1} \cup \cdots \cup E_{N_k}$ is a set of directed edges labeled with $S, N_1, \ldots, N_k$ and each $E_i \subseteq V \times V$ such that every node has precisely one outgoing edge with each label $N_1, \ldots, N_k$.
3. $\tau_{\text{core}} \colon V_{\text{core}} \to \Gamma$ is a total mapping assigning each core-node a letter from Γ
4. $\tau_{\text{port}} \colon V_{\text{port}} \to 2^\Gamma$ is a total mapping assigning each port-node a set of letters from Γ

and t is weakly connected and every node in V_{port} is adjacent to some vertex in V_{core}.

Note that tiles might contain loops $(v, v) \in E_{N_i}$ for $v \in V$, i.e. tiles might not be *simple graphs* which prohibit such edges.

Definition 4.2.2 (Tiling sets of graphs)**.** Let T be a (finite) set of tiles with the label set $\{S, N_1, \ldots, N_k\}$ and let $G = ([n], E_S \cup E_{N_1} \cdots \cup E_{N_k})$ be the graph with node labels over Γ, i.e. there is a function $w \colon [n] \to \Gamma$. We say that G can be *tiled* with T if there is a total mapping $\xi \colon [n] \to T \times \bigcup_{t \in T} V_{\text{core}}(t)$ with

$$\xi(i) = \left(\underbrace{\left(V_{\text{core}}^{t_i} \cup V_{\text{port}}^{t_i}, E^{t_i}, \tau_{\text{core}}^{t_i}, \tau_{\text{port}}^{t_i}\right)}_{=t_i}, v_i \right)$$

where $t_i \in T$ and $v_i \in V_{\text{core}}^{t_i}$ such that

1. For all $i \in [n]$ we have
$$\tau_{\text{core}}^{t_i}(v_i) = w(i) \ .$$
2. For all $i \in [n]$ and for all $u \in V_{\text{core}}^{t_i}$ if there is a path with label $\alpha \in \{S, N_1, \ldots, N_k\}^*$ in t_i from v_i to u (respectively u to v_i) there is a $j \in [n]$ such that there is a path from i to j (or from j to i) in G with $t_j = t_i$ and $v_j = u$.
3. For all $i, j \in [n]$ and paths from j to i (or from i to j) in G with label $\alpha \in \{S, N_1, \ldots, N_k\}^*$ and a node $u \in V_{\text{port}}^{t_j}$ such that there is a path with label α in t_j from v_j to u (respectively u to v_j) we have
$$\tau_{\text{core}}^{t_i}(v_i) \in \tau_{\text{port}}^{t_j}(u) \ .$$
4. For all $i \in [n]$ if $v \in V_{\text{core}}^{t_i}$ is reachable from v_i with label $\alpha \in \{S, N_1, \ldots, N_k\}^*$ in t_i (respectively v_i reachable from v with α) and there is a path from j to i (or i to j respectively) in G with label α then $t_j = t_i$ and $v_j = v$ where $\xi(j) = (t_j, v_j)$.

We say a set of graphs $\mathcal{G}$ can be with T if every $G \in \mathcal{G}$ can be tiled with T.

Definition 4.2.3 (Graph of w)**.** Let $P \subseteq \Gamma^*$ be a auto-generative language recognized by $\Phi_P \in \mathsf{unEx}k\mathsf{FO}[S]$ and let $w \in P$ be a word with $|w| = n$. Then, there are k unique binary second order relations $N_1, \ldots, N_k \subseteq [n]^2$ such that $w, N_1, \ldots, N_k \models \Phi_P$. We call the directed graph with edge labels $S, N_1, \ldots, N_k$

$$G_w = \begin{pmatrix} V = [w], \\ E = (E_S \cup E_{N_1} \cup \cdots \cup E_{N_k}) \end{pmatrix} ,$$

where

$$E_S = \{(i, i+1) \mid i \in [n-1]\}$$

and

$$E_{N_l} = \{(i, j) \mid 1 \leq i, j \leq n \text{ and } (i, j) \in N_l\}$$

for $1 \leq l \leq k$ the *graph of* w. Let $U \in \{S, N_1, \ldots, N_k\}^*$ be a sequence of edge labels and $v, v' \in V$ nodes of G_w. We say *there is a path from* v *to* v' *with label* U if there is a path $p = (v, v_1, \ldots, v_n, v')$ in G_w such that $(v, v_1) \in E_{U[1]}$, for $i = 1, \ldots, n$ we have $(v_i, v_{i+1}) \in E_{U[i]}$ and $(v_n, v') \in E_{U[|U|]}$.

Definition 4.2.4 (Tiling auto-generative languages)**.** Let $P \subseteq \Gamma^*$ be an auto-generative language in $\mathsf{unEx}k\mathsf{FO}[S]$ and let T be a (finite) set of tiles with the label set $\{S, N_1, \ldots, N_k\}$. We say that P can be tiled with T if for all $w \in P$ the graph of G_w can be tiled with T.

When tiling a graph of a word with a set of tiles each position of the word is assigned a tile and is *anchored* a core-vertex with a matching label. All word positions are covered by core-vertices of tiles, which means that port-vertices of one tile overlap with the core of a *neighbor*-tile. Condition 2.) verifies that all core vertices of are tile are used. With condition 3.) make sure that the labels of overlapping *agree* in regard to their label assignments.

Note, that the *neighborhood* of a core is definite. This means if the core of a tile does not have an S-predecessor it may only be placed on the first position of a word. Similarly, if it has no S-successor then the only place it might fit is the last position of a word.

Example 4.2.5. For $w = aaabbbabb$ with $w \in Ł$ the graph G_w is depicted in Figure 4.1.

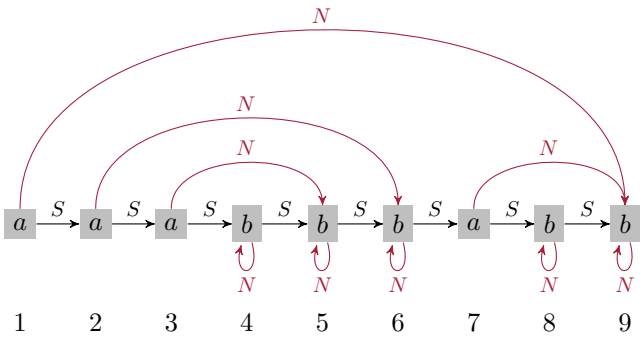
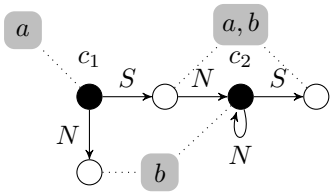


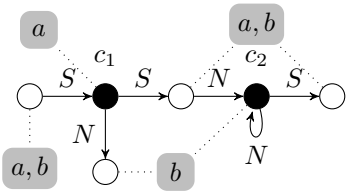
Figure 4.1: The graph of the word *aaabbbabb*.

The Łukasievicz language can be tiled with the set $T_Ł = \{t_1, t_2, t_3, t_4\}$ depicted in Figure 4.2 and following mapping ξ for the word *aaabbbabb* (with the graph in Figure 4.1 and the tiling in Figure 4.3a:

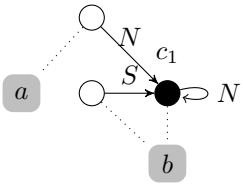
$$\begin{aligned}
\xi(1) &= (t_1, c_1)\\
\xi(2) &= (t_2, c_1)\\
\xi(3) &= (t_2, c_1)\\
\xi(4) &= (t_2, c_2)\\
\xi(5) &= (t_2, c_2)\\
\xi(6) &= (t_1, c_2)\\
\xi(7) &= (t_2, c_1)\\
\xi(8) &= (t_2, c_2)\\
\xi(9) &= (t_3, c_1)
\end{aligned}$$



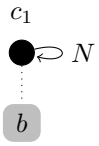
(a) Tile t_1. Note that c_1 has no S-predecessor and therefore only fits on the first position of a word.



(b) Tile t_2 matches as and bs which can be shrunk.



(c) Tile t_3. The core c_1 has S-successor and can therefore only be placed on the last position of a word



(d) Tile t_4 – can only be used to tile the word $b \in Ł$.

Figure 4.2: Tiles for Ł. Black nodes denote core vertices and white nodes port vertices. Each node is connected to a gray rectangle that shows what label τ_{core} and τ_{port} assign to it. t_1 and t_2 both *match* an a and b together. Tile t_3 covers the final position of each word. Finally, t_4 is a tile that is only used when tiling the word $b \in Ł$.

With this definition of tiling we can not make any statements about words that are not contained in an auto-generative language. To cope with this we constrain tilings numerically. This allows so saturate the set of tiles (with *bad* tiles) such that every graph could be tiled when disregarding the constraints.

Muller and Schupp used in their seminal article [MS85] the idea of tiling to show that the MSO-theory of configuration graphs of pushdown automata is decidable. When defining languages they take the inverse approach of ours: Instead of tiling graphs that represent words in the language, they give a set of *forbidden* patterns that, if placed on a graph G_w imply that that w is not in the considered language. In [Tho91] Thomas defines *graph acceptors* which are defined via tiling. He extends the definition of Muller and Schupp by giving both, *allowed* and *forbidden* tiles and allows finite counting of tiles, i.e. Boolean

combinations of statements "there are $\geq n$ instances of tiles of type t in graph G". We adapt Thomas' definition and constrain tilings with such statements. It is noteworthy to mention that our tiles differ from [Tho91] in the sense that Thomas works on graphs with bounded degrees. Our graphs have too a bounded out-degree, but there might be vertices that have an unbounded degree of incoming N_i-edges.

Using the statements "there are $\geq n$ instances of tiles of type t in graph G" of Thomas [Tho91] we separate our set of tiles into two subsets: *Good* tiles which might be used and *bad* tiles which are forbidden.

Definition 4.2.6 (Trivial tiles)**.** We call a tile

$$t = (V_{\text{core}} \cup V_{\text{port}}, E = (E_S \cup E_{N_1} \cup \cdots \cup E_{N_k}), \tau_{\text{core}}, \tau_{\text{port}})$$

trivial if there is precisely one core node c in t. For $k \in \mathbb{N}_0$ let $\mathcal{T}_k$ be the *set of all trivial tiles* with edge labels $S, N_1, \ldots, N_k$.

Remark 4.2.7. Every $N_1, \ldots, N_k$-structure can be tiled with $\mathcal{T}_k$.

In Definition 4.2.3 we introduced the graph G_w for some w in an auto-generative language P. In the following we extend the notion of such graphs by explicitly giving edges E_{N_i} for words $w' \notin P$ and denote them as $G_{w'}^{N_1,\ldots,N_k}$.

Definition 4.2.8 (Tiling with constraints)**.** Let T be a set tiles. A *(tiling-) constraint* is a Boolean combination of statements

"*there are* $\geq n_t$ *instances of tile* $t \in T$" ,

denoted as $\#(t) \geq n_t$. An auto-generative language $P \subseteq \Gamma^*$ in $\mathsf{unExk FO}[S]$ *can be tiled with* T *satisfying* C if

1. for all $w \in P$ the graph G_w can be tiled with T satisfying C and
2. for all $w' \notin P$ and $N_1, \ldots, N_k \subseteq [|w|]^2$ the graph $G_{w'}^{N_1,\ldots,N_k}$ can *not* be tiled with T satisfying C but $G_{w'}^{N_1,\ldots,N_k}$ can be tiled with $T \cup \mathcal{T}_k$.

We use some notational shortcuts when defining constraints. For tiles $t, t_1, \ldots, t_m$ and $n \in \mathbb{N}_0$ some fixed number we use statements $\#(t) \leq n$, $\#(t) = n$, $\#(t) > n$, $\#(t) < n$ and $n = \sum_{i=1}^{m} t_i$ with self-explanatory meanings, which can be constructed by Boolean combinations of *basic* statements.

Example 4.2.9. The Łukasievicz language Ł can be tiled with the previously defined set $T_Ł = \{t_1, t_2, t_3, t_4\}$ satisfying the following constraint $C_Ł$:

$$C_Ł = \left(\#(t_4) = 1 \wedge \bigwedge_{i=1}^{3} \#(t_i) = 0 \right) \vee (\#(t_4) = 0 \wedge \#(t_3) = 1 \wedge \#(t_1) = 1 \wedge \#(t_2) \geq 0)$$

The first clause of $C_Ł$ tiles the word $b \in Ł$. The second one is satisfied for every other word in the Łukasievicz language. There is precisely one tile without a predecessor t_1 and one tile without a successor t_3. Unboundedly many t_2 might be used in-between.

4.3 Tile shrinkable languages

Integrative arguments, like pumping and shrinking seem to be important properties of families of formal languages. We give a notion of shrinking auto-generative languages by removing certain tiles.

Definition 4.3.1 (Shrinking)**.** Let P be an auto-generative language in $\mathsf{unEx}k\mathsf{FO}[S]$ that can be tiled with T satisfying C such that:

1. If there is a $w \in P$ such that there is a (non-negated) statement $\#(t) \geq t_n$ in C and w could be tiled with T satisfying $C \wedge (\#(t) \geq (t_n + 1))$ with $l := |v_{\text{core}}(t)|$, then there is a $w' \in P$ with $|w'| = |w| - l$ that can be obtained by *shrinking* w by the core of an instance of t: Let $i_1, \ldots, i_l \in [|w|]$ with $i_1 < i_2 < \cdots < i_{l-1} < i_l$ be the positions in the word w that are mapped to the core nodes of the same instance of tile t. Then we have that
$$w' = w[1, i_1 - 1] \cdot w[i_1 + 1, i_2 - 1] \cdot \cdots \cdot w[i_{l-1} + 1, i_l - 1] \cdot w[i_l + 1, |w|] \ .$$
Since $w' \in P$ it can be tiled with T satisfying the constraint C.
2. If $w \notin P$ then there are $N_1, \ldots, N_k \subseteq [|w|]^2$ such that $G_w^{N_1,\ldots,N_k}$ can be tiled with $T \cup \mathcal{T}_k$ but not with T alone. Let $\hat{N}_1, \ldots, \hat{N}_k \subseteq [|w|]^2$ be some pointer relations such that $G_w^{\hat{N}_1,\ldots,\hat{N}_k}$ can be tiled with $T \cup \mathcal{T}_k$. If there is a $t \in T_{good}$ such that a statement $\#(t) \geq t_n$ in C is satisfied and the tiling of $G_w^{\hat{N}_1,\ldots,\hat{N}_k}$ uses more then t_n instances of t, then there is a $w' \notin P$ with $|w'| = |w| - |v_{\text{core}}(t)|$ that can be obtained by *shrinking* w by the core of an instance of t: Let $l := |v_{\text{core}}(t)|$ and let $i_1, \ldots, i_l \in [|w|]$ with $i_1 < i_2 < \cdots < i_{l-1} < i_l$ be the positions in the word w that are mapped to the core nodes of the same instance of tile t. Then we have that
$$w' = w[1, i_1 - 1] \cdot w[i_1 + 1, i_2 - 1] \cdot \cdots \cdot w[i_{l-1} + 1, i_l - 1] \cdot w[i_l + 1, |w|] \ .$$

If there is such a set of tiles T and a constraint C we call P *tile-shrinkable*.

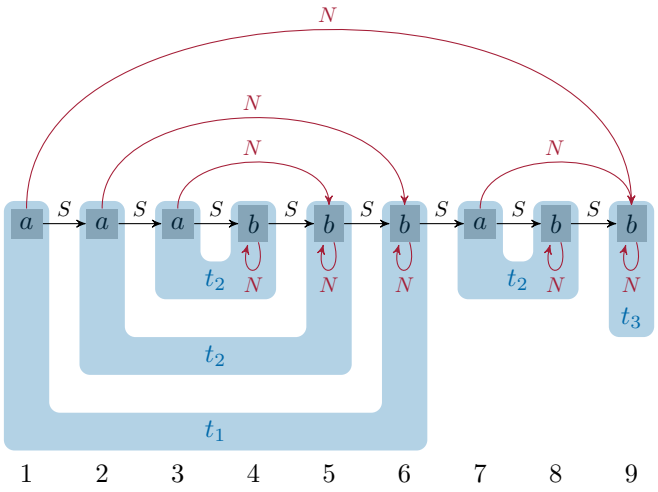
Example 4.3.2. The graph of *aaabbbabb* $\in$ Ł is depicted in Figure 4.3a with previously denoted tiling. The tiling would satisfy the stronger constraint $C_{\text{Ł}} \wedge \#(t_2) \geq 2$. Therefore, we can shrink *aaabbbabb* by the core of one tile t_2 and end up with a Łukasievicz word. We give an exemplary shrinking of a and b on positions 2 and 5 respectively in Figure 4.3b The word $w =$ *aaabbbabba* is not in Ł, thus there is no tiling with $T_{\text{Ł}}$ satisfying $C_{\text{Ł}}$. We can use extra trivial tiles t_a and t_b from $\mathcal{T}_1$, depicted in Figure 4.4 to tile $G_w^{\hat{N}}$ maximally with tiles from $T_{\text{Ł}}$. The maximal tiling of w is shown in Figure 4.5a. We can remove the core of one tile t_2 from the graph $G_w^{\hat{N}}$ which yields the word *aabbabba* $\notin$ Ł and is illustrated in Figure 4.5b.

4.4 Protocol languages

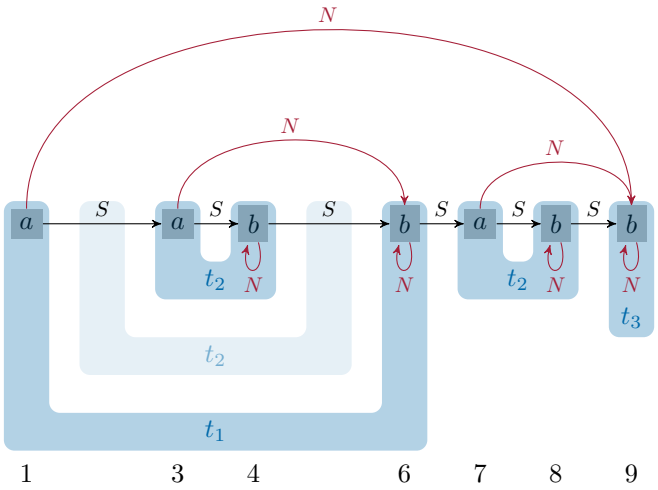
With these requirements we can finally define protocol language.

Definition 4.4.1 (Protocol languages)**.** We call P a *protocol* language if P is auto-generative and tile-shrinkable.

With protocol languages we try to model data structures, and so each word of a protocol language represents the behavior for instance a stack, queue, etc. of a computation model processing some input word. Accordingly, each $\gamma \in \Gamma$ acts as an *operation* or *instruction* of the considered description or computation model.

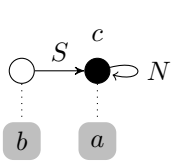


(a) Tiling the word *aaabbbabb* ∈ Ł. The blue markings denote positions of core vertices used in the tiling.

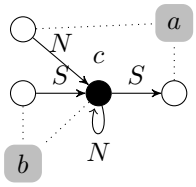


(b) Shrinking the word *aaabbbabb* ∈ Ł by t_2 with the core on positions 2 and 5. The resulting word is *aabbabb* ∈ Ł and can be tiled with $T_Ł$ satisfying $C_Ł$.

Figure 4.3: Exemplary tiling and shrinking for the Łukasievicz word *aaabbbabb*.

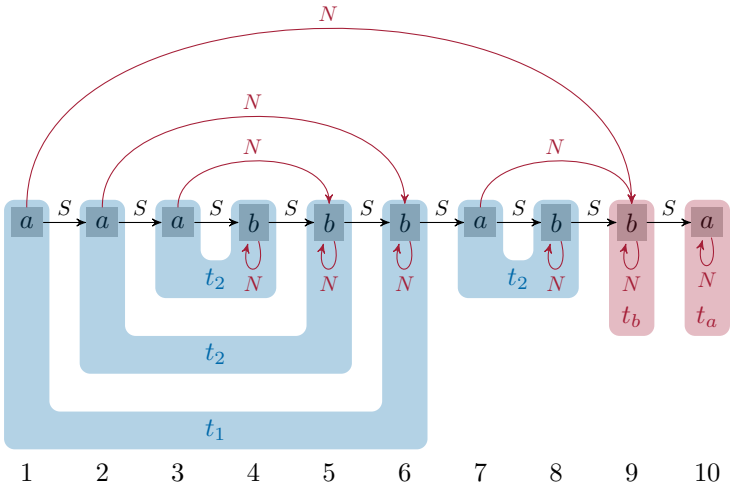


(a) Tile $t_a \in \mathcal{T}_1$

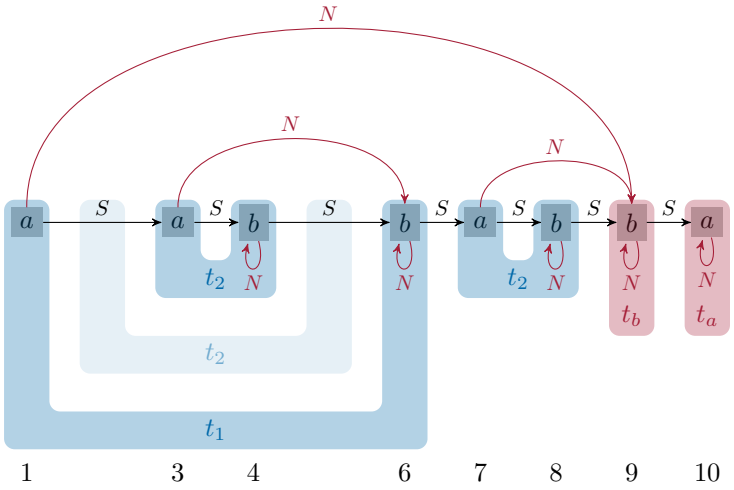


(b) Tile $t_b \in \mathcal{T}_1$

Figure 4.4: Tiles for *aaabbbabba* ∉ Ł.



(a) Maximally tiling the graph $G_w^{\hat{N}}$ of word $w = aaabbbabba \notin \text{Ł}$ with $T_{\text{Ł}} \cup \mathcal{T}_1$ by *guessing* $\hat{N}$. The blue markings denote positions of core vertices of tiles in $T_{\text{Ł}}$. The red ones are tiles from $\mathcal{T}_1$.



(b) Shrinking the word $aaabbbabba \notin \text{Ł}$ by t_2 with the core on positions 2 and 5. The resulting word is $aabbabba \notin \text{Ł}$ and can be tiled with $T_{\text{Ł}} \cup \mathcal{T}_1$ (violating $C_{\text{Ł}}$).

Figure 4.5: Exemplary tiling and shrinking for $aaabbbabb \notin \text{Ł}$.

4.5 Logical extensions of protocol languages

In this section we consider several logical extensions of protocol languages. To this end, we disregard the tiling properties of such languages and focus on the underlying auto-generative characteristics.

First order logic with protocol languages

Definition 4.5.1 ($\mathsf{FO}[P]$)**.** Let $P \subseteq \Gamma^*$ be a protocol language recognized by a formula

$$\Phi_P = \forall x \bigwedge_{\gamma\in\Gamma} \psi_\gamma(x) \iff Q_\gamma(x)$$

in $\mathsf{unEx}k\mathsf{FO}[(Q_\gamma)_{\gamma\in\Gamma}, S]$. We define the class of formulae $\phi \in \mathsf{FO}[P]$ over some alphabet A inductively.

- Φ_P^{shell} is in $\mathsf{FO}[P]$
- $\phi = \Phi_P^{shell} \wedge \theta$ is in $\mathsf{FO}[P]$ for atomic formulae

$$\begin{array}{lll} \theta = & \psi_\gamma(x) & \text{for } \gamma \in \Gamma \text{ and a first order variable } x \ , \\ \theta = & Q_a(x) & \text{for } a \in A \text{ and a first order variable } x \ , \\ \theta = & S(x_1, x_2) & \text{for first order variables } x_1 \text{ and } x_2 \ . \end{array}$$

- For $\Phi_P^{shell} \wedge \theta_1$ and $\Phi_P^{shell} \wedge \theta_2 \in \mathsf{FO}[P]$ the formulae

$$\begin{array}{l} \Phi_P^{shell} \wedge (\theta_1 \wedge \theta_2) \ , \\ \Phi_P^{shell} \wedge (\theta_1 \vee \theta_2) \ , \\ \Phi_P^{shell} \wedge (\neg\theta_1) \ , \\ \Phi_P^{shell} \wedge (\exists x \ \theta_1) \ , \\ \Phi_P^{shell} \wedge (\forall x \ \theta_1) \end{array}$$

  are in $\mathsf{FO}[P]$.

We say that a language $L \subseteq A^*$ is in $\mathsf{FO}[P]$ if there is a formula $\phi_L \in \mathsf{FO}[P]$ such that

$$w \in L \iff w \models \phi_L \ .$$

This means that a language $L \in \mathsf{FO}[P]$ is a first order formula with a numeric successor predicate, letter predicates, the ψ_γ-formulae in conjunction with the shell of some protocol language P.

Example 4.5.2. The formula

$$\Phi_{\text{Ł}}^{shell} \wedge \exists x : \exists y : S^4(y) = x \wedge ((Q_c(x) \iff \psi_a(x)) \wedge (Q_d(x) \iff \psi_b(x)))$$

is in $\mathsf{FO}[\text{Ł}]$ and recognizes all Łukasievicz words of length at least 5 where as are substituted by cs and bs are substituted by ds.

If we further allow the logical formulae to make use of the binary second order variables we end up in the class we call $\mathsf{strongFO}[P]$.

Definition 4.5.3 (strongFO$[P]$)**.** Let $P \subseteq \Gamma^*$ be a protocol language recognized by a formula

$$\Phi_P = \forall x \bigwedge_{\gamma \in \Gamma} \psi_\gamma(x) \iff Q_\gamma(x)$$

in unExkFO$[(Q_\gamma)_{\gamma\in\Gamma}, S]$. We define the class of formulae $\phi \in$ strongFO$[P]$ over some alphabet A inductively.

- Φ_P^{shell} is in strongFO$[P]$
- $\phi = \Phi_P^{shell} \wedge \theta$ is in strongFO$[P]$ for atomic formulae

$$\begin{array}{lll} \theta = & \psi_\gamma(x) & \text{for } \gamma \in \Gamma \text{ and a first order variable } x \ , \\ \theta = & Q_a(x) & \text{for } a \in A \text{ and a first order variable } x \ , \\ \theta = & S(x_1, x_2) & \text{for first order variables } x_1 \text{ and } x_j \ , \\ \theta = & N_i(x_1, x_2) & \text{for } i \in [k] \text{ one of the binary second order variables } N_i \\ & & \text{and first order variables } x_1 \text{ and } x_2 \ . \end{array}$$

- For $\Phi_P^{shell} \wedge \theta_1$ and $\Phi_P^{shell} \wedge \theta_2 \in$ strongFO$[P]$ the formulae

$$\begin{array}{l} \Phi_P^{shell} \wedge (\theta_1 \wedge \theta_2) \ , \\ \Phi_P^{shell} \wedge (\theta_1 \vee \theta_2) \ , \\ \Phi_P^{shell} \wedge (\neg\theta_1) \ , \\ \Phi_P^{shell} \wedge (\exists x \ \theta_1) \ , \\ \Phi_P^{shell} \wedge (\forall x \ \theta_1) \end{array}$$

  are in strongFO$[P]$.

We say that a language $L \subseteq A^*$ is in strongFO$[P]$ if there is a formula $\phi_L \in$ strongFO$[P]$ such that

$$w \in L \iff w \models \phi_L \ .$$

We already mentioned that the Łukasievicz language is the *derivation* language of the context-free grammar in Chomsky normal form with one terminal and one non-terminal symbol. Theorem 4.7.10 can be viewed as an example for a language that is in strongFO[Ł]. There we show that the derivation languages of arbitrary context-free grammars in Chomsky normal form are in strongFO[Ł].

Monadic second order logic with protocol languages

Analogously to FO$[P]$ and strongFO$[P]$ we can define logic classes which may use monadic second order variables.

Definition 4.5.4 (MSO$[P]$)**.** Let $P \subseteq \Gamma^*$ be a protocol language recognized by a formula

$$\Phi_P = \forall x \bigwedge_{\gamma \in \Gamma} \psi_\gamma(x) \iff Q_\gamma(x)$$

in unExkFO$[(Q_\gamma)_{\gamma\in\Gamma}, S]$. We define the class of formulae $\phi \in$ MSO$[P]$ over some alphabet A inductively.

- Φ_P^{shell} is in $\mathsf{MSO}[P]$
- $\phi = \Phi_P^{shell} \wedge \theta$ is in $\mathsf{MSO}[P]$ for atomic formulae

$$\begin{array}{lll} \theta = & \psi_\gamma(x) & \text{for } \gamma \in \Gamma \text{ and a first order variable } x \ , \\ \theta = & Q_a(x) & \text{for } a \in A \text{ and a first order variable } x \ , \\ \theta = & S(x_1, x_2) & \text{for first order variables } x_1 \text{ and } x_2 \ , \\ \theta = & x \in X & \text{for a first order variable } x \text{ and} \\ & & \text{and a monadic second order variable } X \ . \end{array}$$

- For $\Phi_P^{shell} \wedge \theta_1$ and $\Phi_P^{shell} \wedge \theta_2 \in \mathsf{MSO}[P]$ the formulae

$$\begin{array}{l} \Phi_P^{shell} \wedge (\theta_1 \wedge \theta_2) \ , \\ \Phi_P^{shell} \wedge (\theta_1 \vee \theta_2) \ , \\ \Phi_P^{shell} \wedge (\neg\theta_1) \ , \\ \Phi_P^{shell} \wedge (\exists x \ \theta_1) \ , \\ \Phi_P^{shell} \wedge (\forall x \ \theta_1) \ , \\ \Phi_P^{shell} \wedge (\exists X \ \theta_1) \ , \\ \Phi_P^{shell} \wedge (\forall X \ \theta_1) \end{array}$$

  are in $\mathsf{MSO}[P]$.

We say that a language $L \subseteq A^*$ is in $\mathsf{MSO}[P]$ if there is a formula $\phi_L \in \mathsf{MSO}[P]$ such that

$$w \in L \iff w \models \phi_L \ .$$

If we limit the quantification of monadic second order variables to solely existential quantification we end up with *existential monadic second order logic* over P.

Definition 4.5.5 ($\mathsf{EMSO}[P]$)**.** Let $P \subseteq \Gamma^*$ be a protocol language recognized by a formula

$$\Phi_P = \forall x \bigwedge_{\gamma \in \Gamma} \psi_\gamma(x) \iff Q_\gamma(x)$$

in $\mathsf{unEx}k\mathsf{FO}[(Q_\gamma)_{\gamma\in\Gamma}, S]$. We define the class of formulae $\phi \in \mathsf{EMSO}[P]$ over some alphabet A inductively.

- Φ_P^{shell} is in $\mathsf{EMSO}[P]$
- $\phi = \Phi_P^{shell} \wedge \theta$ is in $\mathsf{EMSO}[P]$ for atomic formulae

$$\begin{array}{lll} \theta = & \psi_\gamma(x) & \text{for } \gamma \in \Gamma \text{ and a first order variable } x \ , \\ \theta = & Q_a(x) & \text{for } a \in A \text{ and a first order variable } x \ , \\ \theta = & S(x_1, x_2) & \text{for first order variables } x_1 \text{ and } x_2 \ , \\ \theta = & x \in X & \text{for a first order variable } x \text{ and} \\ & & \text{and a monadic second order variable } X \ . \end{array}$$

- For $\Phi_P^{shell} \wedge \theta_1$ and $\Phi_P^{shell} \wedge \theta_2 \in \mathsf{EMSO}[P]$ the formulae

$$\begin{aligned}
&\Phi_P^{shell} \wedge (\theta_1 \wedge \theta_2) \ , \\
&\Phi_P^{shell} \wedge (\theta_1 \vee \theta_2) \ , \\
&\Phi_P^{shell} \wedge (\exists x \ \theta_1) \ , \\
&\Phi_P^{shell} \wedge (\forall x \ \theta_1) \ , \\
&\Phi_P^{shell} \wedge (\exists X \ \theta_1)
\end{aligned}$$

  are in $\mathsf{EMSO}[P]$.

- For $\Phi_P^{shell} \wedge \theta \in \mathsf{EMSO}[P]$ where θ does *not* contain a sub-formula of the form $\exists X \psi$, i.e. any existential quantification of a monadic second order variable X the formula $\Phi_P^{shell} \wedge (\neg\theta)$ is in $\mathsf{EMSO}[P]$.

We say that a language $L \subseteq A^*$ is in $\mathsf{EMSO}[P]$ if there is a formula $\phi_L \in \mathsf{EMSO}[P]$ such that

$$w \in L \iff w \models \phi_L \ .$$

Example 4.5.6. Monadic second order variables allow to define *closures* of available relations. The closure of the successor is the $<$-relation (and hence $\mathsf{MSO}[S] = \mathsf{MSO}[S, <]$). Therefore, we can define the language $a^+b^+a^+b^+$ which is in $\mathsf{MSO}[S]$ but not in $\mathsf{FO}[S]$:

$$\begin{aligned}
\theta := \exists x_1 \exists x_2 \exists x_3 : x_1 < x_2 < x_3 \wedge \forall y : \\
&(y < x_1 \Rightarrow Q_a(y)) \wedge \\
&(\neg(y < x_1) \wedge y < x_2 \Rightarrow Q_b(y)) \wedge \\
&(\neg(y < x_2) \wedge y < x_3 \Rightarrow Q_a(y)) \wedge \\
&(\neg(y < x_3) \Rightarrow Q_b(y))
\end{aligned}$$

Since $a^+b^+a^+b^+ \cap Ł = \{a^i b^j a^k b^l \mid i + k + 1 = j + l \wedge i \geq j\} = \{w \in \{a, b\}^* \mid w \models \theta \wedge \Phi_Ł^{shell}\}$, we can follow that $\{a^i b^j a^k b^l \mid i + k + 1 = j + l \wedge i \geq j\} \in \mathsf{MSO}[Ł]$ but not in $\mathsf{FO}[Ł]$.

Definition 4.5.7 ($\mathsf{strongMSO}[P]$)**.** Let $P \subseteq \Gamma^*$ be a protocol language recognized by a formula

$$\Phi_P = \forall x \bigwedge_{\gamma \in \Gamma} \psi_\gamma(x) \iff Q_\gamma(x)$$

in $\mathsf{unEx}k\mathsf{FO}[(Q_\gamma)_{\gamma\in\Gamma}, S]$. We define the class of formulae $\phi \in \mathsf{strongMSO}[P]$ over some alphabet A inductively.

- Φ_P^{shell} is in $\mathsf{strongMSO}[P]$
- $\phi = \Phi_P^{shell} \wedge \theta$ is in $\mathsf{strongMSO}[P]$ for atomic formulae

$\theta =$	$\psi_\gamma(x)$	for $\gamma \in \Gamma$ and a first order variable x ,
$\theta =$	$Q_a(x)$	for $a \in A$ and a first order variable x ,
$\theta =$	$S(x_1, x_2)$	for first order variables x_1 and x_2 ,
$\theta =$	$x \in X$	for a first order variable x and and a monadic second order variable X ,
$\theta =$	$N_i(x_1, x_2)$	for $i \in [k]$ one of the binary second order variables N_i and first order variables x_1 and x_2 .

- For $\Phi_P^{shell} \wedge \theta_1$ and $\Phi_P^{shell} \wedge \theta_2 \in \mathsf{strongMSO}[P]$ the formulae
$$\Phi_P^{shell} \wedge (\theta_1 \wedge \theta_2) \ ,$$
$$\Phi_P^{shell} \wedge (\theta_1 \vee \theta_2) \ ,$$
$$\Phi_P^{shell} \wedge (\neg\theta_1) \ ,$$
$$\Phi_P^{shell} \wedge (\exists x \ \theta_1) \ ,$$
$$\Phi_P^{shell} \wedge (\forall x \ \theta_1) \ ,$$
$$\Phi_P^{shell} \wedge (\exists X \ \theta_1) \ ,$$
$$\Phi_P^{shell} \wedge (\forall X \ \theta_1)$$
are in $\mathsf{strongMSO}[P]$.

We say that a language $L \subseteq A^*$ is in $\mathsf{strongMSO}[P]$ if there is a formula $\phi_L \in \mathsf{strongMSO}[P]$ such that
$$w \in L \iff w \models \phi_L \ .$$

An example for $\mathsf{strongMSO}[P]$ will be presented in Section 4.7.1. We show that the analog language to the Łukasievicz language called $\mathcal{G}_2$ is a protocol language which describes deviation languages of grammars in Greibach normal form and show that all context-free languages without the empty word are in $\mathsf{strongMSO}[\mathcal{G}_2]$.

Again, if we limit the quantification of monadic second order variables to solely existential quantification we end up with *strong existential monadic second order logic* over P.

Definition 4.5.8 ($\mathsf{strongEMSO}[P]$)**.** Let $P \subseteq \Gamma^*$ be a protocol language recognized by a formula
$$\Phi_P = \forall x \bigwedge_{\gamma \in \Gamma} \psi_\gamma(x) \iff Q_\gamma(x)$$
in $\mathsf{unEx}k\mathsf{FO}[(Q_\gamma)_{\gamma\in\Gamma}, S]$. We define the class of formulae $\phi \in \mathsf{strongEMSO}[P]$ over some alphabet A inductively.

- Φ_P^{shell} is in $\mathsf{strongEMSO}[P]$
- $\phi = \Phi_P^{shell} \wedge \theta$ is in $\mathsf{strongEMSO}[P]$ for atomic formulae
$$\begin{array}{lll} \theta = & \psi_\gamma(x) & \text{for } \gamma \in \Gamma \text{ and a first order variable } x \ , \\ \theta = & Q_a(x) & \text{for } a \in A \text{ and a first order variable } x \ , \\ \theta = & S(x_1, x_2) & \text{for first order variables } x_1 \text{ and } x_2 \ , \\ \theta = & x \in X & \text{for a first order variable } x \text{ and} \\ & & \text{and a monadic second order variable } X \ , \\ \theta = & N_i(x_1, x_2) & \text{for } i \in [k] \text{ one of the binary second order variables } N_i \\ & & \text{and first order variables } x_1 \text{ and } x_2 \ . \end{array}$$
- For $\Phi_P^{shell} \wedge \theta_1$ and $\Phi_P^{shell} \wedge \theta_2 \in \mathsf{strongEMSO}[P]$ the formulae
$$\Phi_P^{shell} \wedge (\theta_1 \wedge \theta_2) \ ,$$
$$\Phi_P^{shell} \wedge (\theta_1 \vee \theta_2) \ ,$$
$$\Phi_P^{shell} \wedge (\exists x \ \theta_1) \ ,$$
$$\Phi_P^{shell} \wedge (\forall x \ \theta_1) \ ,$$
$$\Phi_P^{shell} \wedge (\exists X \ \theta_1)$$

are in strongEMSO$[P]$.

- For $\Phi_P^{shell} \wedge \theta \in$ strongEMSO$[P]$ where θ does *not* contain a sub-formula of the form $\exists X\psi$, i.e. any existential quantification of a monadic second order variable X the formula $\Phi_P^{shell} \wedge (\neg\theta)$ is in EMSO$[P]$.

We say that a language $L \subseteq A^*$ is in EMSO$[P]$ if there is a formula $\phi_L \in$ EMSO$[P]$ such that

$$w \in L \iff w \models \phi_L \ .$$

Note that by definition we have the following inclusions of the logical extensions for some protocol language P.

1. FO$[P] \subseteq$ EMSO$[P] \subseteq$ MSO$[P]$
2. strongFO$[P] \subseteq$ strongEMSO$[P] \subseteq$ strongMSO$[P]$
3. FO$[P] \subseteq$ strongFO$[P]$
4. EMSO$[P] \subseteq$ strongEMSO$[P]$
5. MSO$[P] \subseteq$ strongMSO$[P]$

4.6 Protocol languages and trio operations

One of the main characteristics of families of formal languages seem to be the closure under trio operations - *homomorphisms*, *inverse homomorphisms* and the *intersection with regular sets*. In this section we analyze our protocol languages and their logical extensions regarding trio operations. Since the tiling aspect again seems disconnected we focus on the auto-generativity of protocol languages.

Proposition 4.6.1. *Let $P \subseteq \Gamma^*$ be a protocol language in* unExkFO$[S]$ *and for some alphabet Y let $f\colon Y \to \Gamma$ be a length-preserving morphism. Then $f^{-1}(P)$ is in* FO$[P]$.

Proof. Assume P is recognized by

$$\Phi_P = \forall x \bigwedge_{\gamma\in\Gamma} \psi_\gamma(x) \iff Q_\gamma(x) \ .$$

A formula $\phi_{f^{-1}(P)}$ in FO$[P]$ recognizing $f^{-1}(P)$ can be construed as follows:

$$\phi_{f^{-1}(P)} = \Phi_P^{shell} \wedge \left(\forall x : \bigwedge_{y\in Y} Q_y(x) \Rightarrow \psi_{f(y)}(x) \right)$$

□

Inverse length-preserving morphic images of protocol languages can be recognized with any of the logical extensions by using the ψ_γ-formulae of the protocol language in combination with letter predicates.

As the locally threshold-testable languages are definable with first order logic and the successor relation and the regular languages are definable with monadic second order logic we have the following immediate closure properties of the logical extension FO$[P]$ and MSO$[P]$.

Proposition 4.6.2. *Let $P \subseteq \Gamma^*$ be a protocol language in* unExkFO$[S]$. *Then* FO$[P]$ *is closed under the intersection with* LocallyThresholdTestable.

Proof. If $I \in \mathsf{FO}[P]$ then there is some formula $\phi_I \in \mathsf{FO}[P]$ with $L(\phi_I) = L$. Since the languages definable with $\mathsf{FO}[S]$ coincide with LocallyThresholdTestable (cf. [Tho82]), for every $L \in$ LocallyThresholdTestable there is a formula $\phi_L \in \mathsf{FO}[S]$ recognizing L. Since $\phi_I \wedge \phi_L \in \mathsf{FO}[P]$ we have that $I \cap L \in \mathsf{FO}[P]$, which proves the claim.

□

Proposition 4.6.3. *Let $P \subseteq \Gamma^*$ be a protocol language in* unExkFO$[S]$. *Then* MSO$[P]$ *is closed under the intersection with regular languages.*

Proof. The proof is analogous to Proposition 4.6.2, as the languages definable in $\mathsf{MSO}[S]$ are the regular languages [Jul61]. □

In Theorem 4.6.4 we show that for a protocol language P the languages definable in $\mathsf{EMSO}[P]$ are the same ones as the length-preserving morphic images of languages definable in $\mathsf{FO}[P]$.

To show that length preserving morphic image of a language over the alphabet Y in $\mathsf{FO}[P]$ can be recognized in $\mathsf{EMSO}[P]$ the idea is that monadic second order variables can be used to simulate the letter predicates which appear in the pre-image of the morphism. We introduce one monadic variable for each letter $y \in Y$ and make sure that each position is contained in exactly one of those variables.

We prove that $\mathsf{EMSO}[P]$ every definable language $L \subseteq \Pi^*$ is homomorphic image of some $\mathsf{FO}[P]$ definable language $I \subseteq Y^*$ by choosing an alphabet Y that encodes Π as well as the power-set of all monadic second order variables used the formula that recognizes L. The length-preserving homomorphisms then is defined as the projection onto the Π component of Y.

Theorem 4.6.4. *Let P be a protocol language in* unExkFO$[S]$. *Then we have*

$$\mathsf{H}_{lp}(\mathsf{FO}[P]) = \mathsf{EMSO}[P] \ ,$$

that is if $I \subseteq Y^*$ is in FO$[P]$ *and $h\colon Y \to \Pi$ is a length-preserving morphism then $h(I)$ is in* EMSO$[P]$ *and for $L \subseteq \Pi^*$ is in* EMSO$[P]$ *then there is an alphabet Y, a language $I \subseteq Y^*$ in* FO$[P]$ *and a length-preserving homomorphism h such that $h(I) = L$.*

Proof. "⇒" We show that if $I \subseteq Y^*$ is in $\mathsf{FO}[P]$ and $h\colon Y \to \Pi$ is a length-preserving morphism then $h(I)$ is in $\mathsf{MSO}[P]$.

Let $I \subseteq Y^*$ be in $\mathsf{FO}[P]$ recognized by a formula $\phi_I = \Phi_P^{shell} \wedge \theta_I$ and let $h\colon Y \to \Pi$ be a length-preserving morphism. We construct a formula $\phi_L = \Phi_P^{shell} \wedge \theta_L \in \mathsf{MSO}[P]$ such that $L(\phi_L) = h(I)$ as follows:

$$\theta_L = \exists (A_y)_{y \in Y} \left(\bigwedge_{\substack{y', y'' \in Y \\ y' \neq y''}} A_{y'} \cap A_{y''} = \emptyset \right) \wedge \left(\forall x : \bigvee_{y''' \in Y} x \in A_{y'''} \right) \wedge$$
$$\left(\forall x : \bigwedge_{y \in Y} Q_{h(y)}(x) \Rightarrow x \in A_y \right) \wedge \theta_I' \ ,$$

where θ_I' is obtained from θ_I by substituting each letter predicate $Q_y(x)$ by $x \in A_y$.

Proof of correctness: We show that if $I \in \mathsf{FO}[P]$ then $h(I) \in \mathsf{MSO}[P]$.

Let $w \in I$. Then $w \models \Phi_P^{shell} \wedge \theta_I$ and therefore $w \models \Phi_P^{shell}$ and $w \models \theta_I$. If $w \models \Phi_P^{shell}$ then $h(w) \models \Phi_P^{shell}$ since h is length-preserving.

If $w \models \theta_I$ and θ_I is free of any letter predicates, then $\theta_I = \theta'_I$ for any valid choice of the monadic second order variables A_y (e.g. trivially one A_y that contains all positions and all other second order variables are *empty*) we have $h(w) \models \theta$. Assume θ_I uses n letter predicates $Q_{y_1}(x_1), \ldots, Q_{y_n}(x_n)$ for $y_1, \ldots, y_n \in Y$ and first order variables $x_1, \ldots, x_n$. Each $Q_{y_i}(x_i)$ is covered by the sub-formula $x_i \in A_{y_i}$. If a letter predicate $Q_y(x)$ evaluates to true, then for $y' \neq y$ no other $Q_{y'}(x)$ can be true, which is ensured for $Q_{h(y)}(x) \Rightarrow x \in A_y$. Therefore, each A_y models the behavior of the letter-predicate Q_y.

So we have $h(w) \models \theta'_I$ and $h(w) \models \Phi_P^{shell}$ which implies $h(w) \models \Phi_P^{shell} \wedge \theta_L$ and therefore $h(w) \in h(I)$ with $h(I) \in \mathsf{MSO}[P, (Q_\pi)_{\pi \in \Pi}, S]$.

"$\Leftarrow$" We show that if $L \subseteq \Pi^*$ is recognized by a formula in $\mathsf{EMSO}[P]$ then there is an alphabet Y, a language $I \subseteq Y^*$ in $\mathsf{FO}[P]$ and a length-preserving homomorphism $h\colon Y \to \Pi$ such that $h(I) = L$.

Assume L is recognized by a formula $\Phi_P^{shell} \wedge \theta_L$ where θ_L is w.l.o.g. in prenex-normal, i.e.

$$\theta_L = \exists A_1 \cdots \exists A_m \mathcal{Q}_1 x_1 \cdots \mathcal{Q}_r x_r \phi_L(A_1, \ldots, A_m, x_1, \ldots, x_r) \ ,$$

where $A_1, \ldots, A_m$ are monadic second order variables and $x_1, \ldots, x_r$ are first order variables and $\mathcal{Q}_i \in \{\exists, \forall\}$ for $i \in [r]$.

Then, let $Y := \{0,1\}^m \times \Pi$ be the alphabet over which we define the first order formula

$$\theta_I := \mathcal{Q}_1 x_1 \cdots \mathcal{Q}_r x_r \phi'_L(x_1, \ldots, x_r) \ ,$$

where ϕ'_L is obtained by substituting each atomic formula $x \in A_i$ in ϕ_L by

$$\bigvee_{\substack{n \in \{0,1\}^m \\ n[i]=1 \\ \pi \in \Pi}} Q_{(n,\pi)}(x)$$

and each $Q_\pi(x)$ by

$$\bigvee_{\substack{n \in \{0,1\}^n \\ \pi \in \Pi}} Q_{(n,\pi)}(x) \ .$$

We have that , $\Phi_P^{shell} \wedge \phi'_L$ is in $\mathsf{FO}[P]$. Let $I = L(\Phi_P^{shell} \wedge \phi'_L)$. Define $h\colon Y \to \Pi$ such that $h((n,\pi)) = \pi$, i.e. h is the projection onto the second component of each Y. Then, $h(I) = L$.

Proof of correctness: If $w \in L$, then $w \models \Phi_P^{shell} \wedge \theta_L$. Since h is length-preserving we have that $u \models \Phi_P^{shell}$ for all $u \in h^{-1}(w)$. Letter predicates in the formula θ_I play two roles simultaneously: While the second component handles the letter predicates of θ_L the first component simulates the existentially quantified monadic second order variables. Since $w \models \theta_L$ each of the m monadic second order variables represents a set of positions of w such that ϕ_L evaluates to true. Vice versa, each word position in w can be contained in the set of positions represented by each of the m monadic second order variables $A_1, \ldots, A_m$ of θ_L. For each position this containment can be encoded as an m-bit vector, which is done in the first component of the alphabet Y. That is,

the position x is contained in A_i if there is a letter on position x such that the i-th bit of the first component is The second component of Y are the letters in Π –therefore, $|Y| = 2^m \cdot |\Pi|$.

□

The same idea can be applied to show that length-preserving morphic images of languages of $\mathsf{strongFO}[P]$ are equal to the languages definable in $\mathsf{strongEMSO}[P]$.

Theorem 4.6.5. *Let P be a protocol language in $\mathsf{unEx}k\mathsf{FO}[S]$. Then we have*

$$\mathsf{H}_{lp}(\mathsf{strongFO}[P]) = \mathsf{strongEMSO}[P] \ ,$$

that is if $I \subseteq Y^*$ is in $\mathsf{strongFO}[P]$ and $h\colon Y \to \Pi$ is a length-preserving morphism then $h(I)$ is in $\mathsf{strongEMSO}[P]$ and for $L \subseteq \Pi^*$ is in $\mathsf{strongEMSO}[P]$ then there is an alphabet Y, a language $I \subseteq Y^*$ in $\mathsf{strongFO}[P]$ and a length-preserving homomorphism h such that $h(I) = L$.

Proof sketch. The proof is analogous to Theorem 4.6.4. Formulae, both in $\mathsf{strongFO}[P]$ and $\mathsf{strongEMSO}[P]$ might additionally make use of the k available binary second order variables of the protocol language as atomic sub-formulae, which has no influence on the monadic second order variables. □

The idea of guessing homomorphic pre-images when showing that $\mathsf{H}_{lp}(\mathsf{FO}[P)] \subseteq \mathsf{MSO}[P]$ in Theorem 4.6.4 can be applied to show that $\mathsf{EMSO}[P]$, $\mathsf{MSO}[P]$, $\mathsf{strongEMSO}[P]$ and $\mathsf{strongMSO}[P]$ are closed under length-preserving morphisms.

Proposition 4.6.6. *Let P be a protocol language in $\mathsf{unEx}k\mathsf{FO}[S]$. Then $\mathsf{EMSO}[P]$, $\mathsf{MSO}[P]$, $\mathsf{strongEMSO}[P]$ and $\mathsf{strongMSO}[P]$ are closed under length-preserving homomorphisms.*

Proof. Let $I \subseteq Y^*$ be in $\mathsf{MSO}[P]$ (in $\mathsf{strongMSO}[P]$, respectively) recognized by a formula $\phi_I = \Phi_P^{shell} \wedge \theta_I$ and $h\colon : Y \to \Pi$ be a length-preserving homomorphisms. A formula for $h(I)$ can be built by using monadic second order variables A_y for all $y \in Y$ to guess the pre-image for each position.

We construct a formula $\phi_L = \Phi_P^{shell} \wedge \theta_L \in \mathsf{MSO}[P]$ ($\mathsf{strongMSO}[P]$) such that $L(\phi_L) = h(I)$ as follows:

$$\theta_L = \exists (A_y)_{y\in Y} \left(\bigwedge_{\substack{y,y'\in Y \\ y'\neq y''}} A_{y'} \cap A_{y''} = \emptyset \right) \wedge \left(\forall x : \bigvee_{y'''\in Y} x \in A_{y'''} \right) \wedge \left(\bigwedge_{y\in Y} \forall x : Q_{h(y)}(x) \Rightarrow x \in A_y \right) : \theta'_I \ ,$$

where θ'_I is obtained from θ_I by substituting each letter predicate $Q_y(x)$ by $x \in A_y$. □

Proposition 4.6.7. *Let $\mathcal{P}_\vee$ be the set of all languages $P \subseteq \Gamma^*$ recognized by a formula $\Phi_P \in \mathsf{unEx}k\mathsf{FO}[S]$ with*

$$\Phi_P = \forall x : \bigwedge_{\gamma\in\Gamma} (Q_\gamma(x) \Leftrightarrow \psi_\gamma(x)) \ ,$$

where each $\psi_\gamma(x)$ is a (non-negated) Boolean combination of statements of the form $W(x) = V(y)$ or $W(x) \neq V(y)$ for $W, V \in \{N_1, \ldots, N_k, S\}^*$. Then $\mathcal{P}_\vee$ is closed under length-preserving morphisms that are injective over P.

Proof. Let $P \subseteq \Gamma$ be in $\mathcal{P}_\vee$ and $h\colon \Gamma \to \Pi$ be an length preserving homomorphism that is injective over P. We show that $h(P) \subseteq \Pi$ is in $\mathcal{P}_\vee$ by giving ψ_π-formulae. Define $\psi_\pi(x)$ as

$$\bigvee_{\substack{\gamma\in\Gamma \\ h(\gamma)=\pi}} \psi_\gamma(x) \ .$$

The resulting ψ_π-formulae are all positive and hence $h(P) \in \mathcal{P}_\vee$ which proves the claim. □

4.7 Protocol languages for some well-known families of formal languages

In this section we will present protocol languages for the regular languages, the context-free languages and the indexed languages. Since we have considered the Łukasievicz language as a running example throughout this chapter we begin the analysis with the context-free languages.

4.7.1 Context-free languages

We provide protocol languages for context-free languages. The Łukasievicz language from the introduction is revisited and analyzed more thoroughly and it is shown that restricted real-time one counter languages are $\mathsf{MSO}[Ł]$. We consider deviation languages of grammars in Greibach normal form $\mathcal{G}_k$ and show that the context-free languages (without λ) are in $\mathsf{strongMSO}[\mathcal{G}_2]$.

Definition 4.7.1 (Derivation language)**.** Let $G = (V, A, P, S)$ be a context-free grammar. For $w \in L(G)$ we define $D(w) = p_1 \dots p_n$ as the *derivation* of w, where

$$\alpha_1 \to_{p_1} \alpha_2 \to_{p_2} \cdots \to_{p_n} \alpha_{n+1} \ ,$$

such that

1. $p_i \in P$ for $1 \leq i \leq n$
2. $\alpha_j \in (V \cup A)^*$ for $1 \leq j \leq n+1$
3. $\alpha_1 = S$,
4. $\alpha_{n+1} = w$
5. p_i is applied to the left-most non-terminal of α_i for $1 \leq i \leq n$.

We call $D(G) := \{D(w) \mid w \in L(G)\} \subseteq P^*$ the *derivation language* of G.

Derivation languages are also known as *Szilard languages* [DPRS97].

We consider derivation languages for context-free grammars in Chomsky normal form. The most simple grammar in CNF which generates an infinite (but notably regular) language is $G_{\mathrm{CNF}} = (\{S\}, \{t\}\{S \to SS \mid t\}, S)$. Still, the derivation trees for all grammars in Chomsky normal form are full binary trees and thus the structure $D(G_{\mathrm{CNF}})$ is the same as for any other such grammar in CNF.

Remark 4.7.2. Let $G_{\text{CNF}} = (\{S\}, \{t\}, \{S \to SS, S \to t\}, S)$ and let

$$a := S \to SS$$
$$b := S \to t$$

Then $D(G_{\text{CNF}}) = Ł = \mathbb{D}_1 b \subseteq \{a, b\}^*$, where $\mathbb{D}_1$ is the Dyck language with one pair of brackets.

Lemma 4.7.3. *Let $w \in Ł$. Then,*

1. *$|w|$ is odd*
2. *Substituting b in $w \in Ł$ with the factor abb yields a word $w' \in Ł$.*
3. *Substituting a factor abb in $w \in Ł$ with the b yields a word $w' \in Ł$.*
4. *$\#_b(w) > 0$ for all $w \in Ł$*
5. *Every word $w \in Ł$ of length n can be generated from some $w' \in Ł$ with $|w'| = n - 2$ by substituting some b in w' by the factor abb.*

Proof.
1. Recall that Ł is recognized by the grammar $G_Ł = (\{S\}, \{a, b\}, \{S \to aSS \mid b\}, S)$. The first rule elongates the derivation by 2, the rule $S \to b$ does not change the length. Since, each derivation is started with a single S each derived word has odd length.
2. Each b in w is derived by an application of the rule $S \to b$. By substituting this rule by an $S \to aSS$ and terminating rules the b can be replaced by abb yielding some $w' \in Ł$.
3. Analogous to 2.
4. The terminating rules in $G_Ł$ produce bs. Therefore, each $w \in Ł$ contains at least one b.
5. Follows from 1., 2. and 3.

□

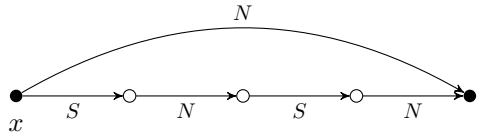
Proposition 4.7.4. *The Łukasievicz language $Ł = D(G_{CNF})$ is an auto-generative language in* unEx1FO$[S]$.

Proof. We construct formulae ψ_γ for $\gamma \in \{a, b\}$ for

$$\Phi_Ł = \forall x : \bigwedge_{\gamma \in \Gamma} (Q_\gamma(x) \Leftrightarrow \psi_\gamma(x))$$

such that $L(\Phi_Ł) = D(G_{\text{CNF}})) = Ł$:

$\psi_a(x) : N(x) = N(S(N(S(x))))\wedge$
$N(\min) = \max$



$\psi_b(x) : N(x) = x\wedge$
$N(\min) = \max$

Correctness of $\Phi_Ł$:

"Ł $\subseteq L(\Phi_{\text{Ł}})$" We show this via induction over the length of words. Let $w \in$ Ł with $|w| = 1$. Then $w = b$ and hence $\psi_b(1)$ holds where the second order variable is $N(1) = 1$.

Assume the claim holds for some word length $n \in \mathbb{N}$. Following Lemma 4.7.3-1 there are no words of even length in Ł. Therefore, consider n to be odd. Let $w \in$ Ł with $|w| = n$ recognized by $\Phi_{\text{Ł}}$ and some pointer relation N. Let $d \in [n]$ be a position in w such that $w[d] = b$. Lemma 4.7.3-4 shows the existence of such a position. Substituting the b at $w[d]$ by the factor abb in w yields according to Lemma 4.7.3 $w' \in$ Ł with $|w'| = n + 2$. We construct N' for w' from N such that $w \models \Phi_{\text{Ł}}$ as follows. All positions that were pointing to the removed b are pointing to the end of the inserted factor. The remaining pointers are shifted by 2.

$$N'(i) = \begin{cases} N(i) & \text{if } i, N(i) < d \ , \\ N(i) + 2 & \text{if } i < d, N(i) \geq d \ , \\ d + 2 & \text{if } i = d \ , \\ d + 1 & \text{if } i = d + 1 \ , \\ d + 2 & \text{if } i = d + 2 \ , \\ N(i - 2) + 2 & \text{if } i, N(i) > d + 2 \ . \end{cases}$$

Therefore, we have a $w' \models \Phi_{\text{Ł}}$ with a pointer relation N'.

"$L(\Phi_{\text{Ł}}) \subseteq$ Ł" We show this via induction over the length of words. Let $|w| = 1$ and $w \models \Phi_{\text{Ł}}$. If $w = a$ then $\psi_a(1)$ can not be true, since 1 has no successors. But, $w = b \in$ Ł with $N(1) = 1$. Assume the claim holds for any word length $n \in \mathbb{N}$. We only consider odd word lengths since following Lemma 4.7.3-1 Ł only contains words of odd length.

Let $|w| = n + 2$ and $w \models \Phi_P$ with some N. We first show that w contains the factor abb. We have $w[1] = a$, since if $w[1] = b$ then, following $\psi_b(1)$ we would need $N(1) = 1$ but would violate $N(\min) = \max$. Similarly, $w[\max] = b$ and $w[\max - 1] = b$ since $\psi_a(\max)$ or $\psi_a(\max - 1)$ would imply the last position in the word had a successor. Suppose that $w[2, n]$ does not contain the factor abb, i.e. $w[2, n] \notin \Gamma^* abb \Gamma^*$.

We have to consider the following cases:

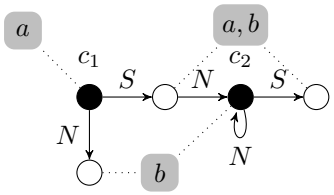
1. If $n = 1$ then by definition we have $w[2, 1] = \lambda$ and therefore $w = abb$.
2. If $w[2, n] \in \{a, b\}^* a$, it ends on an a then $w[n, n + 2] = abb$.
3. If $w[2, n] \in b^+$ then $w[1, 3] = abb$.
4. If $w[2, n] \in (b|\lambda)(ab)^+$ then $w[n - 1, n + 1] = abb$.
5. If $w[2, n] \in (a|\lambda)(ba)^+$ then $w[n - 1, n + 2] = abb$.

Since $(A^* abb A^*)^c \subseteq \{a, b\}^* a \cup b^+ \cup (b|\lambda)(ab)^+ \cup (a|\lambda)(ba)^+$ the cases distinction is complete. Thus, w contains the factor abb, say on $w[d, d + 2]$ for $1 \leq d \leq n$. Then, following $\psi_b(d+1)$ and $\psi_b(d+2)$ we have $N(d+1) = d+1$ and $N(d+2) = d+2$, which together with $\psi_a(d)$ implies $N(d) = d + 2$. We remove the factor ab on $w[d, d+1]$ from w and obtain w' with $|w'| = n$. The pointers N are shifted by 2 positions to fit w' defining N' for $1 \leq i \leq n$ as

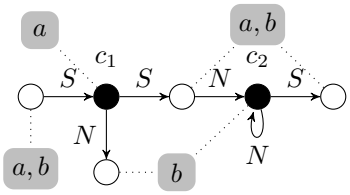
$$N'(i) = \begin{cases} j & \text{if } i, j < d \text{ and } N(i) = j \ , \\ j - 2 & \text{if } i < d, j \geq d \text{ and } N(i) = j \ , \\ j & \text{if } i, j \geq d \text{ and } N(i + 2) = j + 2 \ . \end{cases}$$

□

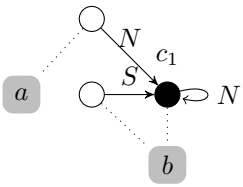
The formula $\Phi_{\text{Ł}}$ of Proposition 4.7.4 defines a unique binary second order relation N for each word in Ł. Recall that each Łukasievicz word represents the pre-order traversal of the derivation tree of some word generated by a grammar in Chomsky normal form. We give an illustration of the pointer relation in context of the derivation tree in Figure 4.7.



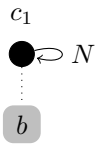
(a) Tile t_1. Note that c_1 has no S-predecessor and therefore only fits on the first position of a word.



(b) Tile t_2 matches as and bs which can be shrunk.



(c) Tile t_3. The core c_1 has S-successor and can therefore only be placed on the last position of a word



(d) Tile t_4 – can only be used to tile the word $b \in$ Ł.

Figure 4.6: Tiles for Ł. Black nodes denote core vertices and white nodes port vertices. Each node is connected to a gray rectangle that shows what label τ_{core} and τ_{port} assign to it. t_1 and t_2 both *match* an a and b together. Tile t_3 covers the final position of each word. Finally, t_4 is a tile that is only used when tiling the word $b \in$ Ł.

The pointer structure is deeply connected to the derivation tree each word in Ł represents. In the following we specify a relation on derivation trees of context-free grammars in Chomsky normal form and show that this structure is *captured* in the behavior of the Łukasievicz language.

Definition 4.7.5. Let $G = (V, A, P, S)$ be a context-free grammar in Chomsky normal form and for $w \in A^+$ and let $d(w) = p_1 \dots p_n$ be the derivation of w. Note that $d(w)$ is the *pre-order* traversal of the derivation tree of w. Each letter in $d(w)$ is the root of a sub-tree of the derivation tree. For $1 \leq i \leq n$ let

$$T_w(i) := \{j \in \{i, \dots, n\} \mid p_j \text{ is in the derivation sub-tree of } w \text{ with root } p_i\}$$

the nodes of the derivation sub-tree of w induced by p_i. Recall that for $n \in \mathbb{N}$ we write $[n]$ for the set $\{1, \dots, n\}$. We define the relation $N_w \subseteq [|d(w)|] \times [|d(w)|]$ for $i \in [|d(w)|]$ as

$$N_w(i) = \max(T(i)) \ .$$

We define $\mathcal{N}_G := \{N_w \mid w \in L(G)\}$ as the set of all N-relations for words generated by G and

$$\mathcal{N} := \bigcup_{G \text{ grammar in CNF}} \mathcal{N}_G$$

Proposition 4.7.6. *For the context-free grammar* $G_{CNF} = (\{S\}, \{s\}, \{S \to SS, S \to s\}, S)$ *we have* $\mathcal{N}_{G_{CNF}} = \mathcal{N}$.

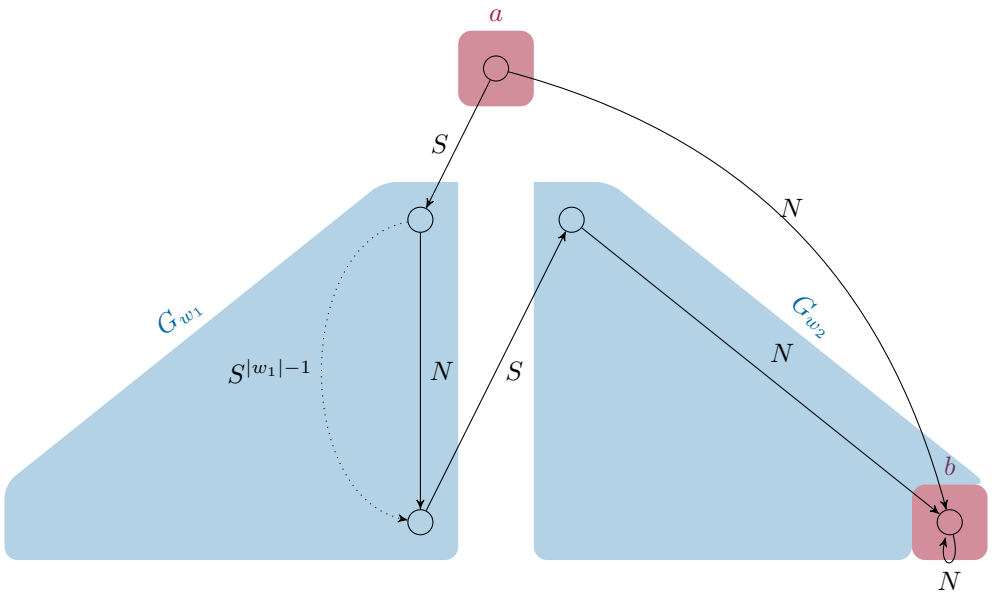


Figure 4.7: N-structure of a pair of a and b for some word w in Ł. Between a and b are sub-structures G_{w_1}, G_{w_2} (the latter containing b) of sub-words $w_1, w_2 \in$ Ł.

Proof. "$\subseteq$" Since G_{CNF} is a context-free grammar in Chomsky normal form $\mathcal{N}_{G_{\text{CNF}}} \subseteq \mathcal{N}$.

"$\supseteq$" We will show that for all context-free grammars in Chomsky normal form $G = (V, A, P, V_S)$ the set of pointer relations generated by it, $\mathcal{N}_G$, is contained in $\mathcal{N}_{G_{\text{CNF}}}$. Since G is in Chomsky normal form, each word in $L(G)$ has a derivation tree, which is a full binary tree. Rules of the form $B \to CD$ form the inner nodes of the derivation tree and leafs are generated by rules of the form $B \to b$ for $B, C, D \in V$ and $b \in A$.

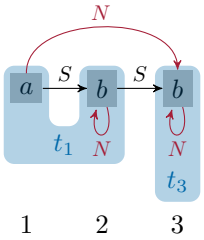
Replacing all rules of the form $B \Rightarrow CD$ by $S \to SS$ and rules $B \to b$ by $S \to t$ yields deviation trees of words $w \in L(G_{\text{CNF}})$. Consequently, if $N \in \mathcal{N}_G$ then $N \in \mathcal{N}_{G_{\text{CNF}}}$.

□

Proposition 4.7.7. *Ł can be tiled with the set $T_Ł = \{t_1, t_2, t_3, t_4\}$ depicted in Figure 4.6 satisfying the constraint C with*

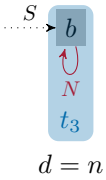
$$C_Ł = \left(\#(t_4) = 1 \wedge \bigwedge_{i=1}^{3} \#(t_i) = 0 \right) \vee (\#(t_4) = 0 \wedge \#(t_3) = 1 \wedge \#(t_1) = 1 \wedge \#(t_2) \geq 0)$$

Proof. We prove this over induction over word length. Let $|w| = 1$ and $w \in$ Ł. Then $w = b$ and $N(1) = 1$. Therefore b can be tiled with a single instance of t_4 and therefore, satisfy $(\#(t_4) = 1 \wedge \bigwedge_{i=1}^{3} \#(t_i) = 0)$ and hence $C_Ł$. The only word $w \in$ Ł with $|w| = 3$ is abb with

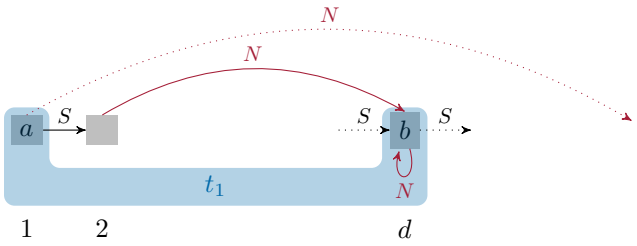


and satisfying $\#(t_4) = 0 \wedge \#(t_3) = 1 \wedge \#(t_1) = 1 \wedge \#(t_2) \geq 0$. Assume the claim holds for any $n \in \mathbb{N}$. We only consider odd word lengths since following Lemma 4.7.3-1 Ł only contains words of odd length. Let $w \in$ Ł with $|w| = n$. Then w can be tiled with one instance of t_1 and t_3 and an $(n-3)/2$ instances of t_2 by some function $\xi : [n] \to T_{Ł} \times \bigcup_{t \in T_{Ł}} V^t_{core}$. Let d be some position with $w[d] = b$. There are three cases how d can be assigned to a core of some tile:

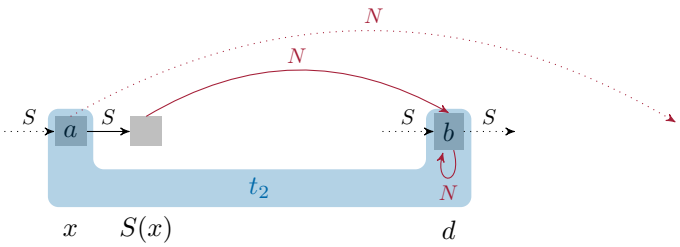
1. If $d = n$ then $\xi(d) = (t_3, c_1)$.



2. If $d = N(S(1))$ then $\xi(d) = (t_1, c_2)$.

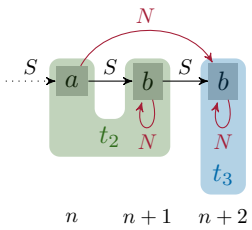


3. Otherwise $\xi(d) = (t_2, c_2)$ where the *matching* a is on some position x such that $N(S(x)) = d$.

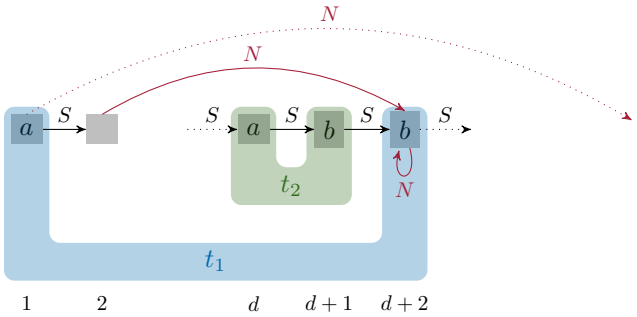


Let w' with $|w'| = n + 2$ be generated from w by substituting the letter b at some position d in w by abb. Following Lemma 4.7.3-2 we have $w' \in$ Ł. For the three different cases this yields the following picture.

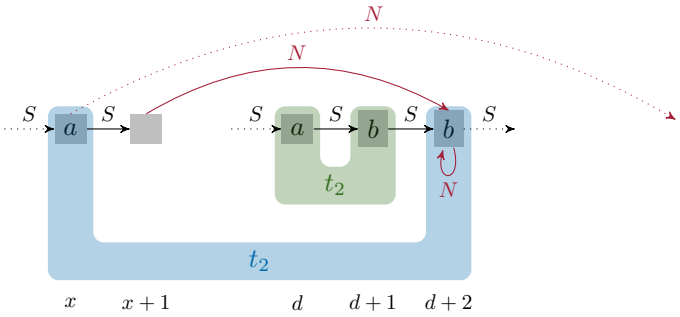
1. If $d = n$ then $\xi'(d+2) = \xi(d) = (t_3, c_1)$.



2. If $d = N(S(1))$ then $\xi(d) = (t_1, c_2)$.



3. Otherwise $\xi(d) = (t_2, c_2)$ where the *matching* a is on some position $x \neq 1$ such that $N(S(x)) = d$.



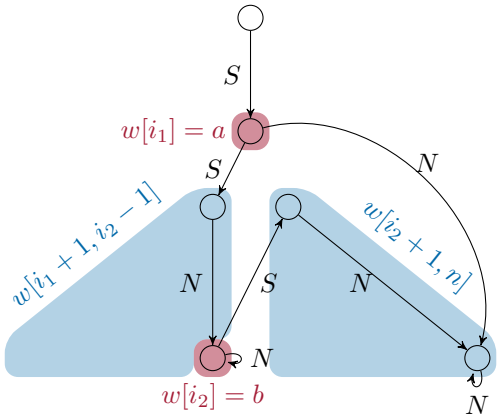
Then, w' can be tiled with the function $\xi' \colon [n+2] \to T_Ł \times \bigcup_{t \in T_Ł} V^t_{core}$:

$$\xi'(i) = \begin{cases} \xi(i) & \text{if } i < d \ , \\ (t_2, c_1) & \text{if } i = d \ , \\ (t_2, c_2) & \text{if } i = d+1 \ , \\ \xi(i-2) & \text{if } i > d+1 \ . \end{cases}$$

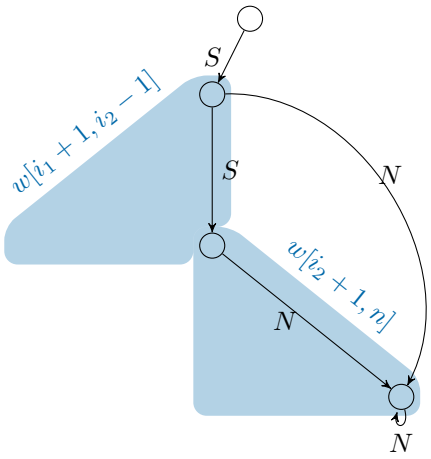
That means the newly introduced a and b are the cores of an extra tile t_2. All other positions in the word keep their previous assignment. Since we insert two letters at position d we have to account for the offset. □

Proposition 4.7.8. *The Łukasievicz language Ł is tile-shrinkable.*

Proof. Following Proposition 4.7.7 every $w \in Ł$ can be tiled with $T_Ł$ satisfying $C_Ł$. Every word $w \in Ł$ that contains at least one instance of t_2 can be shrunk, i.e. all $w \in Ł$ with $|w| \geq 5$. Let $|w| = n \geq 5$ and let i_1 and i_2 be positions in w such that i_1 is mapped to c_1 and i_2 is mapped to c_2 of the same instance of a tile t_2, i.e. we have $N(S(i_1)) = i_2$ and $w[i_1] = a$ and $w[i_2] = b$. Recall that each $w \in Ł$ is an pre-order-traversal of binary tree where an a symbolizes a bifurcation and a b a termination rule.



By removing i_1 and i_2, i.e. the core of some tile t_2 and connecting the right sub-tree i_1 into the *hole* of i_2 in the left sub-tree of i_1 we obtain a coherent derivation sub-tree:



Conversely, the pre-order-traversal of this *shrunk* derivation tree is the word $w[1, i_1 - 1]w[i_1 + 1, i_2 - 1]w[i_2 + 1, n] \in Ł$.

This same shrinking argument works for words $w \notin Ł$: Every core t_2 tile can removed by the structure of w and the resulting word w' will not be in Ł. □

Theorem 4.7.9. *Ł is a protocol language in* unEx1FO$[S]$.

Proof. Proposition 4.7.4 shows that Ł is auto-generative and Proposition 4.7.8 shows that Ł is tile-shrinkable which proves the claim. □

In the following we show that the derivation languages of all context-free grammars in Chomsky normal form are definable in strongFO[Ł].

Theorem 4.7.10. *Let $G = (V, A, P, S)$ be a context-free grammar in Chomsky normal form. Then $D(G) \in \mathsf{strongFO}[\text{Ł}]$.*

Proof. We construct a formula $\Phi_G \in \mathsf{strongFO}[\text{Ł}]$ which makes use of the $\psi_a(x)$ and $\psi_b(x)$ sub-formulae of $\Phi_{\text{Ł}}$ and the binary second order variable N as follows.

$$\Phi_G = \Phi_{\text{Ł}}^{shell} \wedge \texttt{inital}(\min) \wedge \forall x : \texttt{bifurcate}(x) \wedge \texttt{terminate}(x)$$

The sub-formula `initial` verifies that each derivation stats with some rule deviating the start symbol S:

$$\texttt{inital}(x) : \bigvee_{\substack{p \in P \\ p = S \to \beta \\ \beta \in A \cup V^2}} Q_p(x) \wedge \psi_{f(p)}(x)$$

If a derivation of the form $X \to YZ$ is used, we have to make sure that both Y and Z are derived at the correct positions.

$$\texttt{bifurcate}(x) : \bigwedge_{\substack{p \in P \\ p = X \to YZ \\ X,Y,Z \in V}} \left(Q_p(x) \Leftrightarrow \left(\begin{array}{l} \psi_a(x) \\ \wedge \bigvee\limits_{\substack{q \in P \\ q = Y \to \beta \\ \beta \in A \cup V^2}} Q_q(S(x)) \\ \wedge \bigvee\limits_{\substack{r \in P \\ r = Z \to \beta \\ \beta \in A \cup V^2}} Q_r(S(N(S(x)))) \end{array} \right) \right)$$

Finally, for terminating rules we have to confirm that we have the correct structure:

$$\texttt{terminate}(x) : \bigwedge_{\substack{p \in P \\ p = X \to y \\ y \in V, y \in A}} Q_p(x) \Leftrightarrow \psi_b(x) \ .$$

This formula, recognizes the derivation language of G, i.e.

$$D(G) = L(\Phi_G) \ .$$

□

We give the definition of one-counter automata according to Berstel [Ber79].

Definition 4.7.11 (Restricted real-time one-counter automata)**.** A restricted real-time one-counter automaton (RROCA) is a tuple $M = (Q, A, B, \Delta, Q_0, F)$, where Q is a finite set of *states*, A is the *input alphabet*, $B = \{b, b_0\}$ is the *stack alphabet* consisting of two symbols where b_0 is the *bottom of the stack*, $Q_0 \subseteq Q$ is a set of *initial states*, $F \subseteq Q$ is a set of *final states* and $\Delta \subseteq Q \times A \times B \times (B^2 \cup \{\lambda\}) \times Q$ the *transition relation*.

A configuration of M is a triple $(q, x, y) \in Q \times A^* \times ((b^* b_0) \mid \lambda)$. For configurations (q, w, y) and (q', w', y') we write $(q, w, y) \vdash_M (q', w', y')$ if there is some transition $(q, a, z, \gamma, q') \in \Delta$ such that $w = aw'$ and there is an $\alpha \in b^* b_0$ with $z\alpha = y$ and $\gamma\alpha = y'$. Let

$$L(M) = \{w \in A^* \mid \exists q_0 \in Q_0 : \exists q_f \in F : (q_0, w, b_0) \vdash_M^* (q_f, \lambda, \lambda)\}$$

be the language recognized by M. We say $L(M)$ is a *restricted real-time one-counter language*. We denote the family of restricted real-time one-counter languages as RROCL.

We will use a *Nivat-like* [Niv68] argument to characterize the restricted real-time one-counter languages with Ł.

Proposition 4.7.12. *Let* $L \in \mathsf{RROCL}$. *Then* $L \in \mathsf{EMSO}[Ł]$.

Proof. If $L \subseteq \Pi^*$ is a restricted one-counter language then there is a RROCA $M = (Q, A, \{b, b_0\}, \Delta, Q_0, F)$ with $L(M) = L$. Let $R \subseteq \Delta^*$ be the local language

$$R = \left\{ (q_0, a_0, y_0, \gamma_0, q_0')(q_1, a_1, y_1, \gamma_1, q_1') \cdots (q_n, a_n, y_n, \gamma_n, q_n') \middle| \begin{array}{l} q_0 \in Q_0, q_n' \in F, y_0 = b_0 \text{ and} \\ q_i' = q_{i+1} \text{ for } i \in \{0, \ldots, n-1\} \end{array} \right\}$$

and let $f\colon \Delta \to \{a, b\}$ be the length-preserving morphism defined as

$$f((q, a, y, \gamma, q')) = \begin{cases} a & \text{if } |\gamma| = 2 \ , \\ b & \text{if } |\gamma| = 0 \ . \end{cases}$$

R being a local language implies that $R \in \mathsf{FO}[S]$. Following Proposition 4.6.1 the length-preserving inverse morphisms of a protocol language P is in $\mathsf{FO}[P]$ and Proposition 4.6.2 states that $\mathsf{FO}[P]$ is closed under the intersection with $\mathsf{FO}[S]$ definable languages. Therefore, we have that $f^{-1}(Ł) \cap R \in \mathsf{FO}[Ł]$. The set $f^{-1}(Ł) \cap R$ describes all accepting computations of M: The local language R verifies that the states of successive transitions are *matching*, as well as that the first transition starts with an initial state and the final transition ends in an accepting state. $f^{-1}(Ł)$ ensures that the counter is non-negative and that the bottom of the stack is popped with the final transition. Since M is a real-time automaton every word $w \in L(M)$ is accepted with a computation of length $|w|$. Define the length-preserving morphism $h\colon \Delta \to A$ with

$$h((q, a, y, \gamma, q')) = a \ .$$

Then $h(f^{-1}(Ł) \cap R) = L(M)$. □

When investigating the inverse morphic image of protocol languages for not length-preserving morphisms we were able to use of additional second-order variables to work with a *shorter* image. We define the morphism f that maps c to aa an and d to b and consider the inverse image of Ł. To *simulate* the ψ_γ formulae of $\Phi_Ł$ we make *two copies* of the N and S – one for the first letter of the image of c and one for the second letter. Since d is mapped b all copies S and N behave like the originals.

Example 4.7.13. Let $f\colon \{c, d\}^* \to \{a, b\}^*$ the homomorphisms defined as $f(c) = aa$ and $f(d) = b$. Then $f^{-1}(Ł)$ is an auto-generative language in $\mathsf{unEx4FO}[S]$.

Proof. The ψ_c and ψ_d formulae for $f^{-1}(Ł)$ can be constructed as follows:

$$\begin{aligned} \psi_d(x)\colon &\quad N_1(x) = x \wedge N_2(x) = x \wedge S_1(x) = S(x) \wedge S_2(x) = S(x) \\ \psi_c(x)\colon &\quad \psi_{f(c)[1]}(x) \wedge \psi_{f(c)[2]}(x) \wedge S_1(x) = S(x) \wedge S_2(x) = S(x) \end{aligned}$$

where the sub-formulae $\psi_{f(c)[1]}$ and $\psi_{f(c)[2]}$ are defined as

$$\begin{aligned} \psi_{f(c)[1]}\colon &\quad N_1(S_1(N_2(S_1(x)))) \\ \psi_{f(c)[2]}\colon &\quad N_1(S_1(N_1(S_2(x)))) \ . \end{aligned}$$

□

This approach could be generalized: If one would consider a morphism $f_i(c) = a^i$ and $f_i(d) = b$ then $f_i^{-1}(Ł)$ could be defined with i copies of the N-pointer and i copies of the successor relation.

Let $G = (V, A, P, S)$ be a context-free grammar in Chomsky normal form. Then there is an *erasing* morphism $h : P^* \to A$ such that $L(G) = h(D(G))$. The morphism erases every rule $B \to CD \in P$ and maps rules $B \to b \in P$ to b. Since the elements in $D(G)$ are left-derivations of words in $L(G)$ applying the h to derivation of some word $w \in L(G)$ precisely yields w.

In the following section we will consider another context-free protocol language: For derivations languages of grammars in Greibach normal form this relations holds for length-preserving morphisms instead of erasing ones.

Greibach normal form

In the following we will define a series of protocol languages which are derivation languages of context-free grammars in Greibach normal form.

Definition 4.7.14. For $m \geq 0$ let G^m_{GNF} denote grammar in Greibach normal form with

$$G^m_{\mathrm{GNF}} = (\{S\}, \{t\}, \{S \to t \mid tS \mid tSS \mid \cdots \mid tS^m\}, S) \ .$$

Proposition 4.7.15. *For $m \geq 0$ let $\mathcal{G}^m$ be derivation language of G^m_{GNF}, where we abbreviate the rule $S \to t$ with b and for $1 \leq i \leq m$ rules $S \to tS^i$ with a_i. Let $\Gamma = \{a_1, \ldots, a_m, b\}$ Then $\mathcal{G}^m \subseteq \Gamma^*$ is auto-generative and recognizable by a formula $\Phi_{\mathcal{G}^m} \in \mathsf{unEx1FO}[S]$.*

$$\Phi_{\mathcal{G}^m} := \forall x : \bigwedge_{\gamma \in \Gamma} (Q_\gamma(x) \Leftrightarrow \psi_\gamma(x)) \ ,$$

where

$$\psi_b(x) : N(x) = x$$

and for $1 \leq i \leq m$

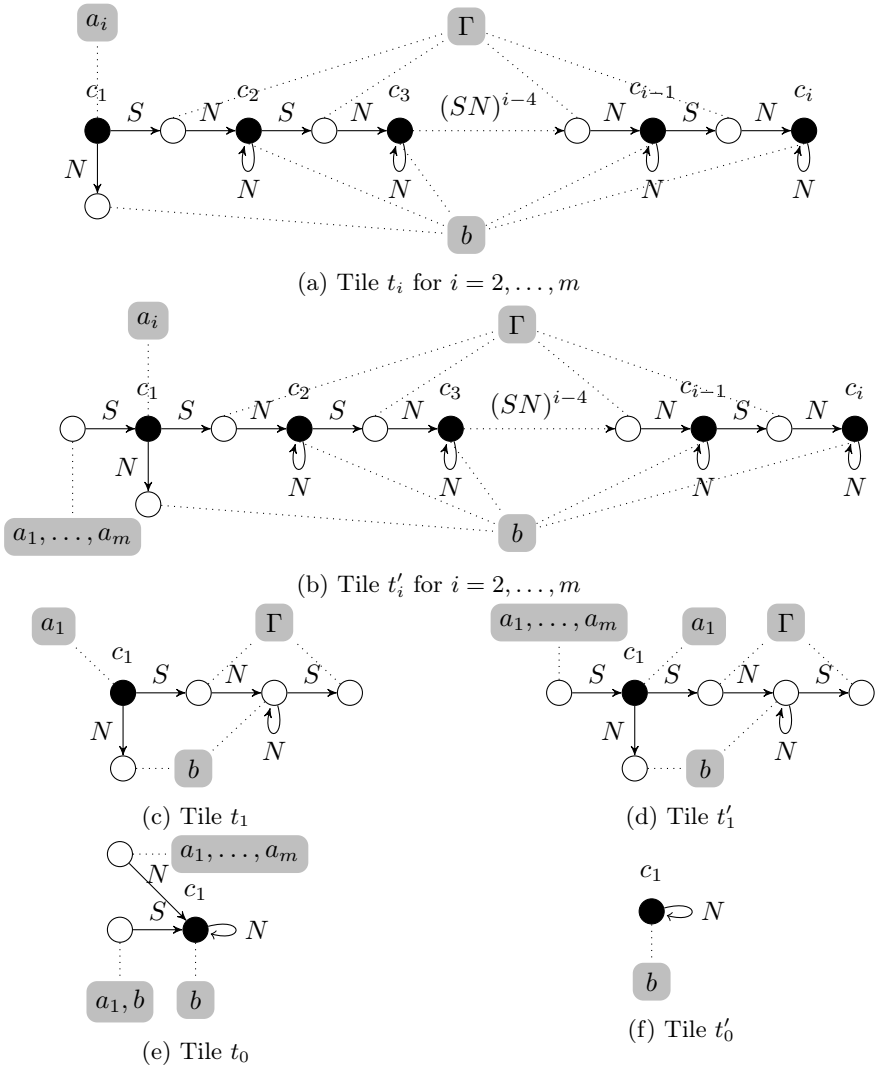
$$\psi_{a_i}(x) : N(x) = (NS)^i(x)$$

Proposition 4.7.16. *For $m \geq 0$ the auto-generative language $\mathcal{G}^m$ can be tiled with the set of tiles $T_m = \{t_0, t_0', t_1, t_1', t_2, t_2', \ldots, t_m, t_m'\}$ shown in Figure 4.8 satisfying constraint C_m:*

$$C_m = \left(\#(t_0') = 1 \wedge \#(t_0) = 0 \wedge \bigwedge_{i=1}^{m} \#(t_i) = 0 \right) \vee \left(\#(t_0') = 0 \wedge \#(t_0) = 1 \wedge 1 = \sum_{i=1}^{m} \#(t_i) \wedge \bigwedge_{i=1}^{m} \#(t_i') \geq 0 \right)$$

Proof. The tiling and shrinking arguments are analog to the tiling and shrinking of the Łukasievicz language. □

Corollary 4.7.17. *For all $m \geq 0$ we have that $\mathcal{G}_m$ is a protocol language.*



(a) Tile t_i for $i = 2, \ldots, m$

(b) Tile t'_i for $i = 2, \ldots, m$

(c) Tile t_1

(d) Tile t'_1

(e) Tile t_0

(f) Tile t'_0

Figure 4.8: Tiles for $\mathcal{G}_m$. Black nodes denote core vertices and white nodes port vertices. Each node is connected to a gray rectangle that shows what label τ_{core} and τ_{port} assign to it.

Proof. We showed in Proposition 4.7.15 that $\mathcal{G}_m$ is auto-generative and in Proposition 4.7.16 shows that $\mathcal{G}_m$ can be tiled with a set T_m satisfying constraint C_m, which together yields that $\mathcal{G}_m$ is a protocol language. □

Proposition 4.7.18. *Let $G = (V, A, P, S)$ be a grammar in Greibach normal form and let $m := \max|\{\beta \in V^* \mid \exists A \in V : A \to a\beta \in P\}|$ be the length of the longest sequence of non-terminals a non-terminal in G can be derived to. Then $D(G) \in \mathsf{strongFO}[\mathcal{G}^m]$.*

Proof. We construct a formula $\Phi_G \in \mathsf{strongFO}[\mathcal{G}^m]$ as follows.

$$\Phi_G = \Phi_{\mathcal{G}^m}^{shell} \wedge \texttt{inital}(\min) \wedge \texttt{terminate}(x) \wedge \forall x : \bigwedge_{i=1}^{k} \texttt{fork}_i(x)$$

The sub-formula `initial` verifies that each derivation stats with some rule deviating the start symbol S:

$$\texttt{inital}(x) : \bigvee_{\substack{p \in P \\ p = S \to \beta \\ \beta \in AV*}} Q_p(x) \wedge \psi_{f(p)}(x)$$

`terminate` checks that terminating rules we have the correct structure:

$$\texttt{terminate}(x) : \bigwedge_{\substack{p \in P \\ p = X \to y \\ y \in V, y \in A}} Q_p(x) \Rightarrow \psi_b(x) \ .$$

For $1 \leq i \leq m$ and rules of the form $X \to yY_1 \dots Y_i$ the sub-formula $\texttt{fork}_i$ ensures that each Y_j is derived at the correct position:

$$\texttt{fork}_i(x) : \bigwedge_{\substack{p \in P \\ p = X \to yY_1 \dots Y_i \\ X, Y_1, \dots Y_i \in V, y \in A}} Q_p(x) \Leftrightarrow \psi_{a_i}(x) \wedge \bigwedge_{j=1}^{i} \left(\bigvee_{\substack{q \in P \\ q = Y_j \to \beta_j \\ \beta_j \in AV^*}} Q_q((NS)^j(x)) \right)$$

This formula then recognizes the derivation language of G, i.e.

$$D(G) = L(\Phi_G) \ .$$

□

Remark 4.7.19 ([HU79])**.** For every context-free language that does not contain λ there is a grammar $G = (V, A, P, S)$ in Greibach normal form such that

$$\max\{|\beta| \mid \exists B \in V, \exists b \in A, \exists \beta \in V* : B \to b\beta\} = 2 \ ,$$

that is every rule on the right-hand side has at most two non-terminals.

Lemma 4.7.20. *Let $G = (V, A, P, S)$ be a context-free grammar in Greibach normal form. Then, for all $w \in L(G)$ we have $|w| = |D(w)|$.*

Proof. Each $p \in P$ has precisely one non-terminal symbol on left hand side and one terminal symbol on the right hand side. Therefore, a derivation of some word $w \in L(G)$ with $|w| = n$ is derived with n elements in P. □

Proposition 4.7.21. *Let $G = (V, A, P, S)$ be a grammar in Greibach normal form and let $m := \max|\{\beta \in V^* \mid \exists A \in V : A \to a\beta \in P\}|$ be the length of the longest sequence of non-terminals a single non-terminal in G can be derived to with a rule in P. Then there is a length preserving morphism $h\colon P^* \to A^*$ such that $h(D(G)) = L(G)$.*

Proof. Let $p = X \to y\beta$ with $X, \in V, y \in A$ and $\beta \in V^*$ We define the homomorphism h as $h(X \to x\beta) = x$. Then $h(D(G)) = L(G)$.

Since G is in Greibach normal form, following Lemma 4.7.20 we have $|d_w| = |w|$. For all $w \in L(G)$ there is a derivation $S \to_{p_1} \alpha_1 \to_{p_2} \cdots \to \alpha_{|w|-1} \to_{p_{|w|}} w$ with $p_i \in P$. We have that $\alpha_i \in A^+V^+$ if $|w| > 1$. In fact, the one-to-one relationship of non-terminal to terminal symbol in GNF grammars imply that $\alpha_i \in A^iV^*$ for $i = 1 \ldots, |w|-1$. Therefore, we have that $w[1, i] = \alpha_i[1, i]$ for $i = 1 \ldots, |w| - 1$. For all $i = 1 \ldots, |w| - 1$ the first non-terminal in α_i is on position $i + 1$. Hence the rule $p_{i+1} = (d_w[i + 1] \to w[i + 1]\beta_{i+1})$ with $\beta_{i+1} \in V^*$ is applied to $\alpha_i[i + 1]$ yielding α_{i+1}. Since $h(p_{i+1}) = w[i + 1]$ we can follow that $h(p_1 \ldots p_n) = w$. □

Theorem 4.7.22. *Every context-free language that does not contain λ is in* $\mathsf{strongEMSO}[\mathcal{G}_2]$.

Proof. Let $L \subseteq \Pi^*$ be a context-free language that does not contain λ. Following Remark 4.7.19 there is a context-free grammar $G = (V, \Pi, P, S)$ in Greibach normal form such that the longest sequence of non-terminals on a right-hand side of a rule is at most 2. Proposition 4.7.18 states that $D(G) \in \mathsf{strongFO}[\mathcal{G}_2]$. Finally, Proposition 4.6.5 states length preserving morphic images of languages in $\mathsf{strongFO}[\mathcal{G}_2]$ are in $\mathsf{strongEMSO}[\mathcal{G}_2]$ which proves the claim. □

The Dyck language $\mathbb{D}_1$

Languages of correctly matched parenthesis are called *Dyck languages*. In this section we show that a modified version of the Dyck language with one type of parenthesis is auto-generative.

Definition 4.7.23. The *Dyck* language with one pair of *parenthesis* $\mathbb{D}_1$ is recognized by the context-free grammar $G = (\{S\}, \{a, b\}, \{S \to aSbS \mid \lambda\}, S)$.

Unfortunately, our attempts to show directly that $\mathbb{D}_1$ is a protocol language failed. The similarity between the Łukasievicz and the Dyck language give the impression that $\mathbb{D}_1$ should have protocol properties, too. We succeeded with a *tagged* version of $\mathbb{D}_1$. When an extra letter for a is introduced into a Łukasievicz word that is matched with the last b and is *tagged* as $\hat{a}$ we obtain (apart from the tagging) a Dyck word.

Definition 4.7.24. The *tagged Dyck* language with one pair of *parenthesis* $\hat{\mathbb{D}}_1$ is recognized by the context-free grammar $G = (\{S, T\}, \{a, \hat{a}, b\}, \{S \to aTS \mid \hat{a}T, T \to aTT \mid b\}, S)$.

Proposition 4.7.25. *Let $\Gamma = \{a, b, \hat{a}\}$. The tagged Dyck language $\hat{\mathbb{D}}_1 \subseteq \Gamma^*$ is auto-generative and is recognized by the formula $\Phi_{\hat{\mathbb{D}}_1} \in \mathsf{unEx1FO}[S]$.*

$$\Phi_{\hat{\mathbb{D}}_1} = \forall x : \bigwedge_{\gamma \in \Gamma} (Q_\gamma(x) \Leftrightarrow \psi_\gamma(x)) \quad ,$$

where

$$\begin{aligned} \psi_a &: \quad N(x) = N(S(N(S(x)))) \wedge N(\min) = \max \\ \psi_b &: \quad N(x) = x \\ \psi_{\hat{a}} &: \quad N(x) = N(S(x)) = \max \end{aligned}$$

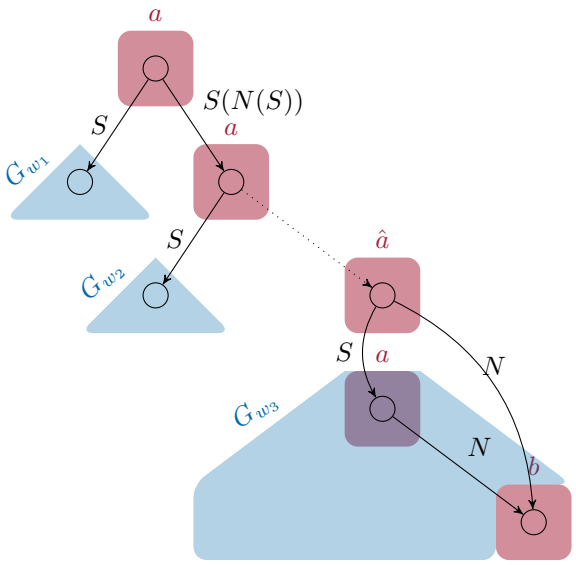


Figure 4.9: N-structure of some word w in $\hat{\mathbb{D}}_1$. There are sub-structures G_{w_1}, G_{w_2} and G_{w_3} (the latter containing b) of sub-words $w_1, w_2w_3 \in Ł$.

Analogous to the tagged $\hat{\mathbb{D}}_1$ one can define *tagged* $\hat{\mathbb{D}}_k \subseteq \{a_1, \ldots, a_k, b_1, \ldots, b_k, \hat{a}_1, \ldots, \hat{a}_k\}$ language as the Dyck language with k pairs of brackets where the last position in a word $w \in \hat{\mathbb{D}}_k$ is matched with $\hat{a}_i$ for $i \in [k]$ when $w[|w|] = b_i$.

Proposition 4.7.26. *For $k \in \mathbb{N}$ the* tagged *Dyck language with k pairs of brackets $\mathbb{D}_k$ is in* $\mathsf{strongFO}[\hat{\mathbb{D}}_1]$.

Proof. With the formula

$$\phi_{\hat{\mathbb{D}}_k} = \phi^{shell}_{\hat{\mathbb{D}}_1} \wedge \forall x : \bigwedge_{i=1}^{k} Q_{a_i}(x) \Rightarrow \psi_a(x) \wedge Q_{b_i}(N(S(x))) \wedge Q_{\hat{a}_i}(x) \Rightarrow \psi_{\hat{a}}(x) \wedge Q_{b_i}(N(x))$$

we have $\phi_{\hat{\mathbb{D}}_k} \in \mathsf{strongFO}[\hat{\mathbb{D}}_1]$ and verify that the bracket types are matched correctly and therefore $\hat{\mathbb{D}}_k = L(\phi_{\hat{\mathbb{D}}_k})$. □

4.7.2 A non-context-free protocol language in $\mathsf{unEx1FO}[S]$

While working with protocol languages in $\mathsf{unEx1FO}[S]$ we had the conjecture that solely context-free languages were definable with an extra binary second order variable. We could disprove this conjecture by showing that a *tagged* version of the poster child of non-context-free languages $\{a^n b^n c^n \mid n \geq 1\}$ is a protocol language in $\mathsf{unEx1FO}[S]$.

Example 4.7.27. The context-sensitive language

$$L_{\text{cs}} = \left\{ a^n \hat{a} b^n \hat{b} c^n \hat{c} \mid n \geq 1 \right\}$$

is an auto-generative language in $\mathsf{unEx1FO}[S]$:

$$\Phi_{L_{\text{cs}}} = \forall x : \bigwedge_{\gamma \in \Gamma} (Q_\gamma(x) \Leftrightarrow \psi_\gamma(x)) \quad ,$$

with

$$\begin{aligned}
\psi_a(x) = \quad & N^2(x) = N^3(x) \wedge x \neq N(x) \wedge N(x) \neq N^2(x) \wedge \\
& N(x) = S(N(S(x))) \\
\psi_{\hat{a}}(x) = \quad & N(x) = S(x) \wedge \\
& N^2(x) = \max \\
\psi_b(x) = \quad & x \neq N(x) \wedge N(x) = N^2(x) \\
& N(x) = S(N(S(x))) \\
\psi_{\hat{b}}(x) = \quad & N(x) = S(x) \\
& N(x) = N^2(x) \\
\psi_c(x) = \quad & x = N(x) \wedge N(S(x)) = S(x) \\
\psi_{\hat{c}}(x) = \quad & x = \max \wedge N(x) = x
\end{aligned}$$

Therefore, L_{cs} is an auto-generative language in $\mathsf{unEx1FO}[S]$. An illustration of the N-structure can be found on Figure 4.10. The set of tiles T for L_{cs} is given in Figure 4.11. L_{cs} can be tiled with T satisfying the following constraint C:

$$C := \#(t_1) = 1 \wedge \#(t_2) \geq 0 \wedge \#(t_3) = 1 \wedge \#(t_4) = 1 \wedge \#(t_5) = 1$$

The tile t_2 contains in its core precise one a, b and c each. Therefore, we can shrink all words in L_{cs} with word length at least 6.

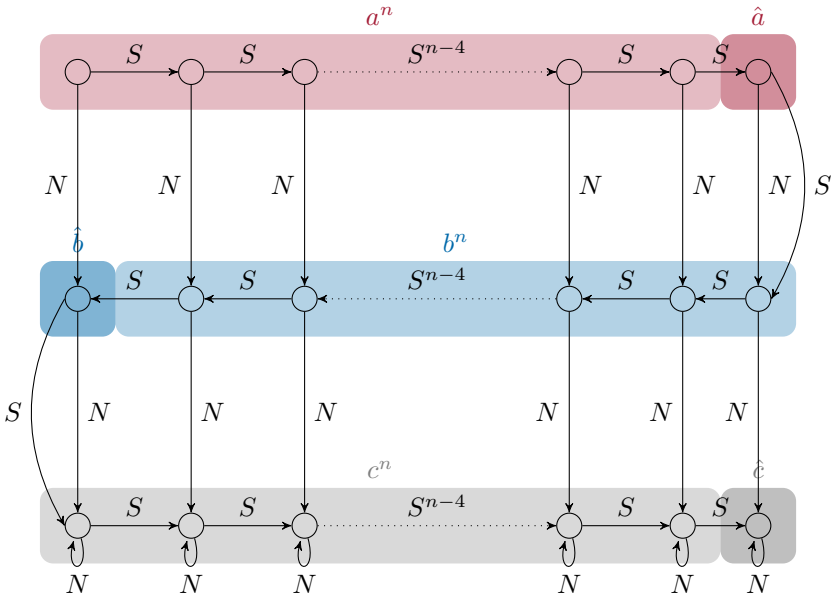
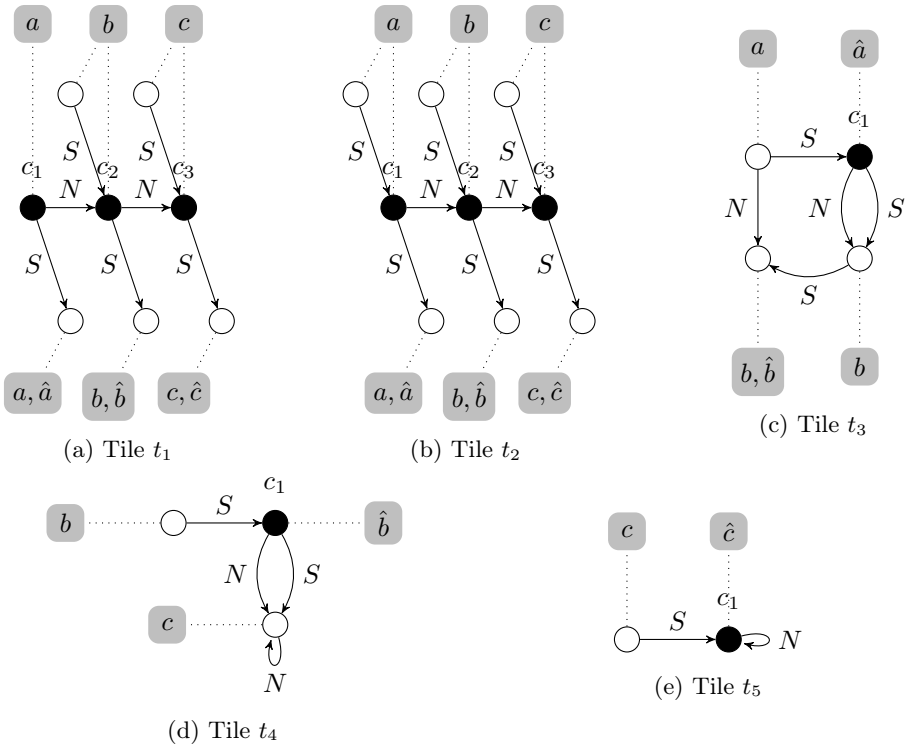


Figure 4.10: Visualization of the N-structure for a word $a^n\hat{a}b^n\hat{b}c^n\hat{c}$ in L_{cs}.

4.8 Protocol languages for indexed languages

In this section we analyze protocol languages for indexed languages. We give only proof sketches and point the interested reader to the forthcoming thesis [San19]. We extend the notion of derivation languages to indexed languages.



(a) Tile t_1

(b) Tile t_2

(c) Tile t_3

(d) Tile t_4

(e) Tile t_5

Figure 4.11: Tiles for $a^n\hat{a}b^n\hat{b}c^n\hat{c}$. Black nodes denote core vertices and white nodes port vertices. Each node is connected to a gray rectangle that shows what label τ_{core} and τ_{port} assign to it.

Definition 4.8.1 (Derivation language)**.** Let $G = (V, A, F, P, S)$ be an indexed grammar. For $w \in L(G)$ we define $D(w) = p_1 \dots p_n$ as the *derivation* of w, where

$$\alpha_1 \to_{p_1} \alpha_2 \to_{p_2} \cdots \to_{p_n} \alpha_{n+1} \ ,$$

such that

1. $p_i \in P$ for $1 \leq i \leq n$
2. $\alpha_j \in (V \cup A \cup F)^*$ for $1 \leq j \leq n+1$
3. $\alpha_1 = S$,
4. $\alpha_{n+1} = w$
5. p_i is applied to the left-most non-terminal (and the potential stack symbols successive to it) of α_i for $1 \leq i \leq n$.

We call $D(G) := \{D(w) \mid w \in L(G)\} \subseteq P^*$ the *derivation language* of G.

Derivation languages of indexed grammars were considered before in e.g. [Mon87].

We consider the analogous language to the Łukasievicz language, which forms the frame for context-free derivations to the indexed derivation languages.

Definition 4.8.2. Let G_{index} be the indexed grammar $G_{\text{index}} = (V = \{S\}, A = \{t\}, F = \{f\}, P = \{S \to SS \mid t \mid S_f, S_f \to S\}, S)$ and abbreviate the derivation rules in P with

$$\begin{aligned} a :=& S \to SS \ , \\ b :=& S \to t \ , \\ c :=& S \to S_f \ , \\ d :=& S_f \to S \ . \end{aligned}$$

We define $Z := D(G_{\text{index}}) \subseteq \{a, b, c, d\}^*$, i.e. the derivation language of G_{index}.

Proposition 4.8.3. *The derivation language of the indexed grammar in normal form with one non-terminal, one terminal and one stack symbol Z is auto-generative and recognized by a $\Phi_Z \in \mathsf{unEx2FO}[S]$.*

Proof sketch. Z is recognized with two additional pointer relations, we reference with N and V. The N-relation is the same one as the one used in the formula for Ł in Proposition 4.7.4: Each word in Z is the *pre-order-traversal* of a derivation tree for a word produced by the grammar G_{index}. The N pointer of each position is constrained such that it points onto the last position representing the same sub-tree. The newly introduced pointer V indicates for each position where the top-most stack symbol was pushed on. If the stack of the current position is empty, the V-relation points to the last position of the word. Let $\Gamma = \{a, b, c, d\}$. Then Φ_z can be defined as follows:

$$\Phi_Z := \forall x : \bigwedge_{\gamma \in \Gamma} (Q_\gamma(x) \Leftrightarrow \psi_\gamma(x)) \ ,$$

with

$$\begin{aligned} \psi_a(x) : \quad & = N(x) = N(S(N(S(x)))) \wedge N(\min) = \max \\ & \quad V(x) = V(S(x)) \wedge V(x) = V(S(N(S(x)))) \\ \psi_b(x) : \quad & = N(x) = x \wedge \\ & \quad V(x) = \max \\ \psi_c(x) : \quad & = N(x) = N(S(x)) \wedge \\ & \quad V(S(x)) = x \\ \psi_d(x) : \quad & = N(x) = N(S(x)) \wedge \\ & \quad V(S(x)) = V(V(x)) \wedge \psi_c(V(x)) \end{aligned}$$

The correctness of the formula Φ_Z is proved in [San19]. □

In ψ_a and ψ_b the requirements to the N-pointers are the same as for Ł. An a on some position x is a bifurcation rule, where each of its two children will inherit its stack content. Therefore, the V-pointers of both children point the same location as the V-pointer of x. A terminating rule b might only occur once its stack is empty, i.e. the V-pointer points to the last position in the word. In ψ_c and ψ_d are index producing and index consuming rules, respectively. If there is a c on some position x then it must be verified that V-pointer of the next position points to x. Finally, a d pops one index and points to the position of the second last c.

Proposition 4.8.4. *The derivation language Z is a protocol language.*

Proof sketch. A language is a protocol language if it is auto-generative and the tileable with a set of tiles satisfying some constraint. Proposition 4.8.3 yields the first part, for the second part we refer to [San19]. □

Proposition 4.8.5. *Let G be an indexed grammar in normal form. Then its derivation language $D(G)$ is in* strongFO$[Z]$.

Proof sketch. A strongFO$[Z]$ formula recognizing $D(G)$ may use the ψ_γ formulae and the binary second order variables of Φ_Z as well as letter-predicates. The proof idea is to use the Q-predicates in combination with the ψ_γ-formulae to assure that correct derivation rules are chosen. The conjunction of the Φ_Z^{shell}-formula ensures that the overall structure of the word is consistent. For a detailed proof see [San19]. □

4.9 Protocol languages for regular languages

Regular languages can be defined with grammars with rules of the form $S \to tU$ and $S \to t$ for non-terminals S, U and terminal symbol t. As before, in the context-free case we consider derivation languages of regular languages and to that end will first focus on an *elementary* grammar with a single terminal and non-terminal symbol.

Definition 4.9.1. Let G_{Reg} be the regular grammar $(\{S\}, \{t\}, \{S \to t \mid tS\}, S)$ and abbreviate the derivation rules in P with

$$a := S \to tS \ ,$$
$$b := S \to t \ .$$

We define $J := \{a^n b | n \geq 0\} = D(G_{\text{Reg}})$, i.e. the derivation language of G_{Reg}.

Proposition 4.9.2. *The language J is auto-generative and recognized by a formula $\Phi \in$* unEx0FO$[S]$ *with*

$$\forall x : \forall x : \bigwedge_{\gamma \in \Gamma} (Q_\gamma(x) \Leftrightarrow \psi_\gamma(x)) \ ,$$

and

$$\psi_a(x) : x \neq \max$$
$$\psi_b(x) : x = \max$$

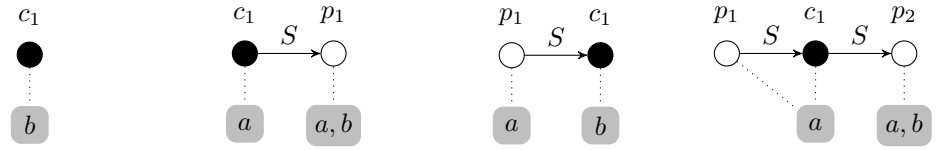
Proposition 4.9.3. *The language J can be tiled with the following set of tiles $T = \{t_1, t_2, t_3, t_4\}$ which is given graphically in Figure 4.12 satisfying the following constraint C:*

$$C = \left(\#(t_1) = 1 \wedge \bigwedge_{i=2}^{4} \#(t_i) = 0 \right) \vee$$
$$(\#(t_2) = 1 \wedge \#(t_3) = 1 \wedge \#(t_1) = 0 \wedge \#(t_4) \geq 0)$$

Proof. J is equal to $\mathcal{G}_0$ and hence tiled analogous to Ł. □

Very similarly, we can show that a^* is a protocol language. Since it is a single letter language, the $\psi_a(x)$ formula is the Boolean constant 1. The tiles of a^* are isomorphic to those of J and only differ in the sense that each τ maps each node to a.

Proposition 4.9.4. *For the protocol language a^* and its logical extensions there are the following equivalences:*



(a) t_1 (b) t_2 (c) t_3 (d) t_4

Figure 4.12: The tile set T for J. Black nodes denote core vertices and white nodes port vertices. Each node is connected to a gray rectangle that shows what label τ_{core} and τ_{port} assign to it.

1. $\mathsf{LocallyThresholdTestable} = \mathsf{FO}[a^*] = \mathsf{strongFO}[a^*]$
2. $\mathsf{Reg} = \mathsf{MSO}[a^*] = \mathsf{Trio}(a^*)$

Proof. 1. Since $\psi_a = 1$ the formula $\Phi_{a^*}^{shell}$ is the Boolean constant 1 we have that $\mathsf{FO}[a^*] = \mathsf{FO}[S]$. The equivalence of $\mathsf{FO}[a^*] = \mathsf{strongFO}[a^*]$ is given by the lack of binary second order variables. Thomas [Tho82] showed that the locally threshold testable language coincide with the languages definable with first order logic and the successor relation.

2. With the same argumentation as in 1) we have that $\mathsf{MSO}[a^*] = \mathsf{MSO}[S]$ and $\mathsf{MSO}[S] = \mathsf{strongMSO}[S]$. Büchi [Büc60] Elgot [Elg61] and Trakhtenbrot [Tra61] showed that the regular languages are precisely the sets of words definable with monadic second order logic with the successor-relation which proves the claim.

□

4.10 The simulation of letter predicates

In this section we show that the introduction of more pointer relations allows to *simulate* letter predicates. In the language $\textsc{Copy} = \{(w\#)^n \mid n \geq 1, w \in A^+\}$ the letters a and b *behave the same way*. Therefore, we should not expect that that $\textsc{Copy}$ is auto-generative in $\mathsf{unEx1FO}[S]$.

By introducing a pointer relation for each letter of the copied word w we were to overcome this hurdle: For each $a \in A$ we define a pointer $N_a(x)$ that points to x when $w[x] = a$ and to the last position in the word otherwise. With this technique we were able to show that the *tagged* Copy-language

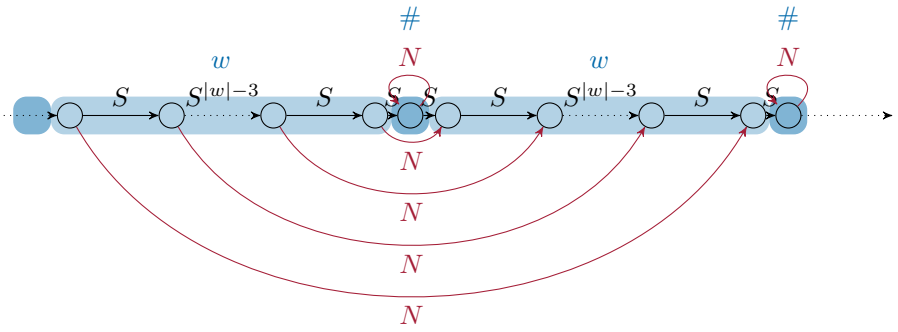
$$\widehat{\textsc{Copy}} = \left\{ (w\#)^n \hat{w}\hat{\#} \;\middle|\; \begin{array}{l} n \geq 1, w \in \{a,b\}^+ \text{ and} \\ \forall i \in [|w|] : w[i] = a \iff \hat{w}[i] = \hat{a} \wedge w[i] = b \iff \hat{w}[i] = \hat{b} \end{array} \right\}$$

is an auto-generative language in $\mathsf{unEx4FO}[S]$. In the last repetition of the word all letters and the final marker are tagged.

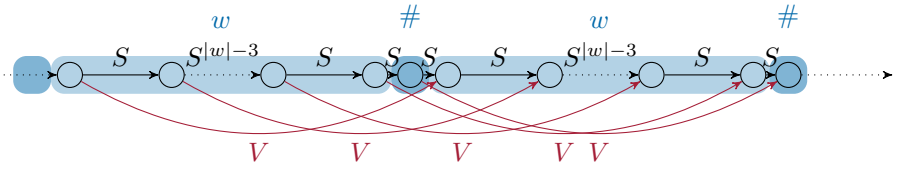
Proposition 4.10.1. $\widehat{\textsc{Copy}} \subseteq \{a, b, \#, \hat{a}, \hat{b}, \hat{\#}\}^*$ *is an auto-generative language in* $\mathsf{unEx4FO}[S]$.

Proof. We use the binary second order variables N, V, N_a, N_b to define the formula $\Phi_{\widehat{\textsc{Copy}}}$ with $L(\Phi_{\widehat{\textsc{Copy}}}) = \widehat{\textsc{Copy}}$

$$\Phi_{\widehat{\textsc{Copy}}} = \forall x : \bigwedge_{\gamma \in \Gamma} (Q_\gamma(x) \Leftrightarrow \psi_\gamma(x)) \quad ,$$



(a) N-pointer structure of $\widehat{\text{Copy}}$.



(b) V-pointer structure of $\widehat{\text{Copy}}$.

Figure 4.13: The N and V pointer structure for two successive repetitions in a word in $\widehat{\text{Copy}}$.

with

$$
\begin{aligned}
\psi_a(x): &= \mathtt{cs}(x) \wedge \mathtt{cf}(x) \wedge N_a(x) = x \wedge N_b(x) = \max \wedge \\
&\quad N_a(V(x)) = V(x) \wedge N_b(V(x)) = \max \\
\psi_b(x): &= \mathtt{cs}(x) \wedge \mathtt{cf}(x) \wedge N_a(x) = \max \wedge N_b(x) = x \wedge \\
&\quad N_a(V(x)) = \max \wedge N_b(V(x)) = x \\
\psi_\#(x): &= \mathtt{cs}(x) \wedge N(x) = x \wedge N_a(x) = \max \wedge N_b(x) = \max \wedge \\
&\quad N_a(V(x)) = \max \wedge N_b(V(x)) = \max \\
\psi_{\hat{a}}(x): &= N(x) = x \wedge V(x) = x \wedge N_a(x) = x \wedge N_b(x) = \max \\
\psi_{\hat{b}}(x): &= N(x) = x \wedge V(x) = x \wedge N_a(x) = \max \wedge N_b(x) = x \\
\psi_{\hat{\#}}(x): &= x = \max \wedge N(x) = \max \wedge V(x) = \max \wedge N_a(x) = \max \wedge N_b(x) = \max
\end{aligned}
$$

The sub-formulae $\mathtt{cs}(x)$ and $\mathtt{cf}(x)$ are abbreviations for

$$
\begin{aligned}
\mathtt{cs}(x): &\quad N(S(x)) = S(N(x)) \\
\mathtt{cf}(x): &\quad N(x) = S(N(S(x))) \ .
\end{aligned}
$$

The structural visualizations of $\mathtt{cs}(x)$ and $\mathtt{cf}(x)$ are depicted in Figure 4.13. □

Van Leeuwen [VL75] showed that ET0L contains NP-complete languages by proving that the satisfiability problem of Boolean formulae could be encoded as an ET0L language. We can apply this encoding to show that the satisfiability problem is in MSO[$\widehat{\text{Copy}}$].

Proposition 4.10.2. Sat *is in* MSO[$\widehat{\text{Copy}}$].

Proof. We can interpret $w \in \{a,b\}^m$ as a sequence of m Boolean values such that x_i is true if $w[i] = a$ and x_i is false if $w[i] = b$ (analogously for $\hat{a}$ and $\hat{b}$). Then, words in $\widehat{\textsc{Copy}}$ is an be viewed as repetitions of m Boolean values.

Let $Y = (\{a,b,\hat{a},\hat{b}\} \times \{\oplus,\ominus,\circ\}) \cup \{\#,\hat{\#}\}$ be an alphabet and $f\colon Y \to \{a,b,\hat{a},\hat{b},\#,\hat{\#}\}$ be the homomorphism

$$f(y) = \begin{cases} \gamma & \text{if } y = (\gamma, s) \in \{a,b,\hat{a},\hat{b}\} \times \{\oplus,\ominus,\circ\} \ , \\ d & \text{if } y = d \in \{\#,\hat{\#}\} \ . \end{cases}$$

Following Proposition 4.6.1 we have that $f^{-1}(\widehat{\textsc{Copy}}) \in \mathsf{FO}[\widehat{\textsc{Copy}}]$ and since $\mathsf{FO}[\widehat{\textsc{Copy}}] \subseteq \mathsf{MSO}[\widehat{\textsc{Copy}}]$ also $f^{-1}(\widehat{\textsc{Copy}}) \in \mathsf{MSO}[\widehat{\textsc{Copy}}]$. Then $f^{-1}(\widehat{\textsc{Copy}})$ is a sequence of word $w_1\#w_2\#\cdots w_n\hat{\#}$ that be viewed as a Boolean formula in conjunctive normal form with $|w_i|$ Boolean constants where each w_i is a clause. For $\gamma \in \{a,b,\hat{a},\hat{b}\}$ we have that

1. if $w_i[j] = (\gamma, \oplus)$ then x_i is positive in the i-th clause.
2. if $w_i[j] = (\gamma, \ominus)$ then x_i is negated in the i-th clause.
3. if $w_i[j] = (\cdot, \circ)$ then x_i is not contained in the i-th clause.

Define the regular language $R \subseteq Y^*$ that contains all words that contain at least one of $(a,\oplus),(\hat{a},\oplus),(b,\ominus),(\hat{b},\ominus)$ between two $\#$ (or $\hat{\#}$). Then, $w_1\#w_2\#\cdots w_n\hat{\#}$ in $R \cap f^{-1}(\widehat{\textsc{Copy}})$ represents a sequence of n clauses that all evaluate to true. Let $\Pi = \{\oplus,\ominus,\circ,\#\}$ and define the length-preserving homomorphism $h\colon Y \to \Pi$ as:

$$h(y) = \begin{cases} s & \text{if } y = (\gamma, s) \in \{a,b,\hat{a},\hat{b}\} \times \{\oplus,\ominus,\circ\} \ , \\ \# & \text{if } y \in \{\#,\hat{\#}\} \ . \end{cases}$$

Then $w_1\#w_2\#\cdots w_n\#$ in $h(R \cap f^{-1}(\widehat{\textsc{Copy}}))$ is Boolean formula in conjunctive normal form with $|w_i|$ variables that can be satisfied and therefore $h(R \cap f^{-1}(\widehat{\textsc{Copy}})) = \textsc{Sat}$. Since h is a length-preserving morphism and $R \cap f^{-1}(\widehat{\textsc{Copy}})$ is in $\mathsf{MSO}[\widehat{\textsc{Copy}}]$ following 4.6.6 we have that $\textsc{Sat} \in \mathsf{MSO}[\widehat{\textsc{Copy}}]$. □

4.11 Deciding the emptiness of protocol languages

The tiling and shrinking aspect of protocol languages comes to play when deciding their emptiness.

Theorem 4.11.1. *Let $P \subseteq \Gamma^*$ be a protocol language in $\mathsf{unExk FO}[S]$. Then the emptiness of P is decidable.*

Proof. Assume P can be tiled with a set T satisfying some constraint C, where C is a Boolean combination of expressions $\#(t) \geq n_t$. Assume without loss of generality that C is in disjunctive normal form, i.e. $C = c_1 \vee c_2 \vee \cdots c_n$. Then, each of the finitely many clauses of C can be considered as a way to tile words in P and there is a (not necessarily disjoint) set cover $P = P_1 \cup P_2 \cup \cdots \cup P_m$ such $P_i \subseteq P$ can be tiled with constraint c_i. Let c_i consist

of $|c_i|$ literals which are possibly negated expressions $\#(t) \geq n_t$ where $t \in T$ and $n_t \in \mathbb{N}_0$. Then c_i is of the form

$$\bigwedge_{j=1}^{|c_i|} \underbrace{(\neg)\#(t_{ij}) \geq n_{ij}}_{e_{ij}} \quad .$$

Let l_i be the number of core nodes that are required for c_i to be satisfied:

$$l_i := \sum_{\substack{j \in [|c_i|] \\ e_{ij} \text{ is positive in } c_i}} |v_{\text{core}}(t_j)| \cdot n_{t_j}$$

Since P is a protocol language it is also tile shrinkable. Thus, if $w \in P_i$ and $|w| > l_i$ and then w can be *shrunk* to some $w' \in P_i$ with $|w'| < |w|$. If $w \in P_i$ and $|w| = l_i$ then there is no shorter word in P_i. Therefore, by repeatedly shrinking a word in P_i we obtain one of length l_i. This gives us an upper bound of words we have to consider for each P_i. Hence, we can decide the emptiness of P by testing the finitely many words in Γ^* up to length $\min\{l_i \mid i = 1, \ldots, n\}$. □

Theorem 4.11.2. *Let $P \subseteq \Gamma^*$ be a protocol language in $\mathsf{unEx}k\mathsf{FO}[S]$. Then $int_{\mathsf{Loc}}(P)$ is decidable, i.e. it is decidable given some local language L if $P \cap L = \emptyset$.*

Proof. Assume P be tiled with a set T satisfying a constraint C. Let the local language $L \subseteq \Gamma^*$ be given as triple (S_L, E_L, F_L) with $S_L, E_L \subseteq \Gamma$ the allowed first and last letters respectively and $F_L \subseteq \Gamma^2$ the set of *forbidden successors*.

If T contains a tile with no S-labeled edges then P contains a word w of length 1. $P \cap L \neq \emptyset$ if $w \in L$.

Therefore, assume that in each tile there is at least one path that uses *successor edges* E_S. Since the nodes are assigned label in Γ these paths define some word(s). When a tile is placed onto the graph of some word in P, this label sequence has to match the underlying sub-word. Thus, in order to obtain the intersection of $P \cap L$ we modify the set of tiles such that the label sequences of the tiles are factors of words in L.

Let $t \in T$ be a tile with $t = (V, E_S \cup E_{N_1} \cup \cdots E_{N_k}, \tau_{\text{core}}, \tau_{\text{port}})$.

Let $p = (v_1, \ldots, v_m)$ be a directed path in t with $(v_i, v_{i+1}) \in E_S$ for $i = 1, \ldots, m-1$ such that there is neither a $v_0 \in V$, nor a $v_{m+1} \in V$ with $(v_0, v_1) \in E_S$ or $(v_m, v_{m+1}) \in E_S$. We call such a path p *maximal*. Let $\mathcal{P}_t$ be the set of all maximal paths in t. Note, that $\mathcal{P}_t$ is finite. For $i = 1, \ldots, m$ let

$$a_i := \begin{cases} \tau_{\text{core}}(v_i) & \text{if } v_i \in V_{\text{core}} \quad , \\ \tau_{\text{port}}(v_i) & \text{if } v_i \in V_{\text{port}} \quad . \end{cases}$$

Note that if $v_i \in V_{\text{core}}$ then a_i is a single letter. Otherwise a_i is a finite set of letters. We build a regular expression of each path: Let $L(p) = L(a_1 \ldots a_m)$ be the *path language* of a path $p = (v_1, \ldots, v_m)$. Recall that the neighborhood of core nodes in tiles are described exhaustively: If a core node has no E_S-predecessor then it can only be placed on the first position of a word in P. Alternatively, if a core node has no E_S-successor it can only be placed on the final position of a word.

If $v_1 \in V_{\text{core}}$ then t is a tile which is placed on the beginning of some $w \in P$, since v_1 has in-degree 0 and therefore in particular no predecessor concerning the successor relation.

Conversely, if t is placed on the end of some word, then v_m is a core node without a successor. We intersect the path language of p with the local set L as follows:

$$I(p) := \begin{cases} L(p) \cap (S\Gamma^* \setminus \Gamma^* F \Gamma^*) & \text{if } v_1 \in V_{\text{core}} \text{ and } v_m \in V_{\text{port}} \ , \\ L(p) \cap (\Gamma^* E \setminus \Gamma^* F \Gamma^*) & \text{if } v_m \in V_{\text{core}} \text{ and } v_1 \in V_{\text{port}} \ , \\ L(p) \cap (S\Gamma^* E \setminus \Gamma^* F \Gamma^*) & \text{if } v_1, v_m \in V_{\text{core}} \ , \\ L(p) \cap (\Gamma^* \setminus \Gamma^* F \Gamma^*) & \text{otherwise} \ . \end{cases}$$

Since $L(p)$ is finite, $I(p)$ is finite, too. For each $x \in I(p)$ let $\tau^x \colon V_{\text{core}} \cup V_{\text{port}} \to \Gamma$ with $\tau^x(v_i) = x[i]$ for $i = 1, \ldots, m$. Note that for all $v \in V_{\text{core}}$ and all $x \in I(p)$ we have that $\tau^x(v) = \tau_{\text{core}}(v)$. Let p and q be maximal paths and let $x \in I(p)$ and $y \in I(q)$. We say that x and y *agree* if for $v \in \{u \mid p = (p_1, \ldots, u, \ldots, p_l) \text{ and } q = (q_1 \ldots, u, \ldots, q_g)\}$ we have $\tau^x(v) = \tau^y(v)$. Analogously, for maximal paths $p_1, \ldots, p_z$ node labels $x_1, \ldots, x_z$ with $x_i \in I(p_i)$ *agree* with each other if all x_i and x_j agree pairwise with each other. We define the set of $|\mathcal{P}_t|$-tuples of words

$$S_L^t := \left\{ (x_1, \ldots, x_{|\mathcal{P}_t|}) \,\middle|\, \begin{array}{l} x_i \in I(p_i) \text{ with } p_i \in \mathcal{P}_t \text{ and} \\ x_1, \ldots, x_{|\mathcal{P}_t|} \text{ agree with each other} \end{array} \right\}$$

Since the words (i.e. components) of each $\mathbf{x} = (x_1, \ldots, x_{|\mathcal{P}_t|}) \in S_L$ agree with each other we can now define a *finer* τ_{port}-function which restricts the port nodes labels in t to be in accord with the local language L:

$$\tau_{\text{port}}^{\mathbf{x}}(v) := \tau^{x_i}(v) \ ,$$

such that $v \in p_i$ and x_i is the i-th component of $\mathbf{x}$.

We define a modified set of tiles with restricted port labels.

$$T^L = \left\{ (V, E, \tau_{\text{core}}, \tau_{\text{port}}^{\mathbf{x}}) \,\middle|\, \begin{array}{l} t = (V, E, \tau_{\text{core}}, \tau_{\text{port}}) \text{ and} \\ \mathbf{x} \in S_L^t \end{array} \right\}$$

In C each statement $\#(t) \geq n_t$ for $t \in T$ is substituted by

$$\left(\sum_{\mathbf{x} \in S_t^L} \#(t_{\mathbf{x}}) \right) \geq n_t \ .$$

Each statement $\neg(\#(t) \geq n_t)$ is substituted by $n_t \geq \sum_{\mathbf{x} \in S_t^L} \#(t_{\mathbf{x}})$, which can be constructed as a Boolean combination of basic statements $\#(t_{\mathbf{x}}) \geq n_{\mathbf{x}}$ since n_t is fixed. Let C^L denote this modification of C. We now have that w can be tiled with T^L satisfying C^L iff $w \in P \cap L$.

Therefore, we can then make use of Theorem 4.11.1 and test whether P can be tiled with T^L satisfying C^L to decide the emptiness of the intersection of P with the local language L. □

Theorem 4.11.3. *Let $P \subseteq \Gamma^*$ be a protocol language* unExk$\mathsf{FO}[S]$ *and $h \colon Y \to \Gamma$ be a length-preserving homomorphism for some alphabet Y. Then $h^{-1}(P) = \emptyset$ is decidable.*

Proof. Assume that P can be tiled with a set $T = T$ satisfying a constraint C. The idea to decide the emptiness is the application of the inverse morphisms on the labels of each tile and subsequently using the results of Theorem 4.11.1. Since core vertices are assigned exactly one label we need to make *copies* of each tile.

We modify T as follows: For each $t \in T$ with $t = (V^t_{\text{core}} \cup V^t_{\text{port}}, E^t, \tau^t_{\text{core}}, \tau^t_{\text{port}})$ we construct $\prod_{v \in V^t_{\text{core}}} |h^{-1}(\tau^t_{\text{core}}(v))|$ different versions of t. We presume that the vertices in V^t_{core} are in some arbitrary, but fixed order. Let

$$D_t := \left\{ \left(y_1, \ldots, y_{|v^t_{\text{core}}|}\right) \middle| y_i \in h^{-1}(\tau^t_{\text{core}}(v_i)) \text{ for } i = 1, \ldots, \left|V^t_{\text{core}}\right| \right\}$$

be the set of all combinations assigning each core vertex of tile t an element of its label's pre-image. For $\mathbf{y} \in D_t$ let $\tau_{\mathbf{y}} \colon V^t_{\text{core}} \to Y$ be the mapping with $\tau_{\mathbf{y}}(v_i) = y_i$ for $i = 1, \ldots, |V^t_{\text{core}}|$. We define $z^t_{\mathbf{y}} := (V^t_{\text{core}} \cup V^t_{\text{port}}, E^t, \tau_{\mathbf{y}}, h^{-1} \circ \tau^t_{\text{port}})$. Then the reformed set of tiles T' is defined as

$$T' = \bigcup_{t \in T} \{ z^t_{\mathbf{y}} \mid \mathbf{y} \in D_t \} \ .$$

In C each statement $\#(t) \geq n_t$ for $t \in T$ is substituted by

$$\bigvee_{\mathbf{y} \in D_t} \#(t_{\mathbf{y}}) \geq n_t \ .$$

which can be constructed as a Boolean combination of basic statements $\#(t_{\mathbf{y}}) \geq k$ since n_t is fixed. Following Theorem 4.11.1 we can decide whether there is some $w \in h^{-1}(P)$ that can be tiled with T' satisfying C' and hence decide the emptiness of $h^{-1}(P)$. □

Theorem 4.11.4. *Let $P \subseteq \Gamma^*$ be a protocol language $\mathsf{unEx}k\mathsf{FO}[S]$ and $h \colon Y \to \Gamma$ be a length-preserving homomorphism for some alphabet Y. Then $int_{\mathsf{Loc}}(h^{-1}(P))$ is decidable, i.e. it is decidable given some local language L if $h^{-1}(P) \cap L = \emptyset$.*

Proof. Assume P can be tiled with a set T satisfying a constraint C. We first apply the modifications used in Theorem 4.11.3 to T and C. Subsequently, the Theorem 4.11.3 is applied to determine the intersection emptiness of $h^{-1}(P)$ with the local language L. □

Corollary 4.11.5. *Let $P \subseteq \Gamma^*$ be a protocol language $\mathsf{unEx}k\mathsf{FO}[S]$. Then $int_{\mathsf{Reg}}(P)$ is decidable, i.e. given a regular language R it is decidable if $P \cap R = \emptyset$.*

Proof. Each regular language is the morphic image of a length preserving homomorphism applied to a local language [MP71] This means there is an alphabet Y, some local language $L \subseteq Y^*$ and a length-preserving homomorphism $h \colon Y^* \to \Gamma^*$ such that $R = h(L)$. Then, in order to decide if $P \cap R \neq \emptyset$ we can decide whether $P \cap h(L) \neq \emptyset$. Note that for any function $f \colon X \to Y$ and $A \subseteq X$ and $B \subseteq Y$ the following equivalence holds:

$$f(A) \cap B = \emptyset \iff A \cap f^{-1}(B) = \emptyset \ .$$

To decide whether $P \cap h(L) \neq \emptyset$ we therefore can test if $h^{-1}(P) \cap L \neq \emptyset$ which is decidable following Theorem 4.11.3. □

Following Theorem 3.6.1 we have that the full AFL of a language for which int_{Reg} is decidable also has a decidability of the emptiness of regular intersection. Therefore the full AFL of a protocol language has a decidable emptiness of regular intersection. Corollary 4.11.5 also implies that protocol languages have a decidable word problem, as the question whether $w \in P$ is equivalent to $P \cap \{w\} \neq \emptyset$.

CHAPTER 5

Discussion

In the search for a characterization of families of formal languages besides plain enumeration we analyzed two notions.

The first candidate was that a set of languages is *formal* if the emptiness of regular intersection is decidable. It turns out that every complexity class contains complete problems for which the emptiness of regular intersection is decidable. That is, for any decidable language L we can construct a language L' such that L and L' are AC^0-reducible to each other and $int_{\mathsf{Reg}} L'$ is decidable. We call the technique used to obtain this result *hiding*, which was inspired by the ideas used to obtain the *dense completeness* results in [KL12a].

This was made stronger by the result that the full AFL generated by any language for which int_{Reg} is decidable also has a decidable emptiness of regular intersection. Together these results refute the decidability of int_{Reg} as a *formality* criterion.

An intermediate result when working with int_{Reg} was published in [GKLW18]. We showed for regular sets of encoded quantified Boolean formulae it is decidable whether any true formulae are contained. The encoding we used is similar to the encoding in [Sto76] who showed that the TQBF is PSPACE-complete. Wrathall [Wra76] showed that the set of true quantified Boolean formulae with quantification depth k are complete for Σ_k^P if the first quantifier is existential and Π_k^P-complete if the first one is universal. Therefore, we obtain complete languages for every level of the polynomial hierarchy and PSPACE for which int_{Reg} is decidable. In contrast to the PSPACE-complete languages which are obtained from the *hiding* process for which int_{Reg} is decidable, the sets $L_{k\text{-TQBF}}$ and L_{TQBF} are more *natural*. Even though we have shown the decidability of int_{Reg} for quantified Boolean formulae we have not done a complexity analysis of the methods. Since for all words w we have that $L_{\text{TQBF}} \cap \{w\} \neq \emptyset$ if and only if $w \in L_{\text{TQBF}}$ we can expect an actual implementation of our approach to be of high complexity.

An open question is the role of the encoding of TQBF – i.e. is there an encoding of quantified Boolean formulae such that int_{Reg} is undecidable? Using for example a binary encoding of the quantification depth and index would make it still possible to construct literal transition sets $\Lambda_{qq'}$. Also the principle of *spreading* existentially quantified variables and *uniting*

universally quantified variables in the transition sets would still be applicable and yield finite representatives and thus yield the decidability of the emptiness.

Following the argumentation in the Cook-Levin theorem, two successive configurations of a fixed Turing machine differ only at up to three positions from another. Therefore we can give a regular language which contains only strings of two successive configurations which are letter-wise shuffled, i.e. for configurations w and v write $w[1]v[1]w[2]v[2]\cdots w[n]v[n]$ and use placeholders if $|w| \neq |v|$. Such encodings have been used to show undecidability results [CK98] and could possibly be applicable to encode literals of quantified Boolean formulae to obtain an undecidable int_{Reg} property. Our co-author Wolf in [GKLW18] analyzes several natural languages regarding int_{Reg} in [Wol18].

The main contribution of this thesis is the introduction of the notion of *protocol languages*. They capture in some sense the *data-structure* underlying a family of formal languages and are *visible* in transition or derivation languages.

We build upon the ideas of Lautemann et al. [LST94] who give a logic characterization of the context-free languages by second order relations which are semantically constrained to describe *nestings*. We establish the logic fragment $\mathsf{unEx}k\mathsf{FO}[S]$ which allows the use of k binary second order variables which are semantically constrained to behave like total mappings and are defined unambiguously by each word. *Auto-generative languages* are defined within $\mathsf{unEx}k\mathsf{FO}[S]$ where each letter in a word induces a certain sub-structure. Vice versa the structure of the binary second order variables defines the letter for each position. These sub-structures are defined by the ψ_γ-formulae for each letter γ, which are conjuncted (un-)equalities of endpoints of paths in the graphs induced by the binary relations and can be used as a substitute for the letter predicates. Auto-generative languages are therefore of deterministic nature. If a word models the formula of an auto-generative language there are k uniquely determined binary second order variables. Regarding the positions of such a word as nodes and the successor as well as the second order variables as edges each (word) auto-generative language gives rise to a graph language.

We apply the ideas of [MP11, Tho91] to define *tiling* on the resulting graphs of auto-generative languages. Madhusudan and Parlato [MP11] have used tiling-based graph recognizers to show the decidability of the emptiness problem for (word-)automata with different types of auxiliary storage mechanisms. They apply a simplified version of tiling introduced by Thomas [Tho91] which further allows counting (up to constants) the number of instances a tile was used. Thomas defines his graph recognizers on classes of labeled graphs which have at most one incoming and one outgoing edge of each label, which results in a bounded degree. The families of graphs auto-generative (word-)languages induce have a fixed out-degree: For an auto-generative language in $\mathsf{unEx}k\mathsf{FO}[S]$ the k binary second order variables are constrained such that they are total functions. As the successor relation defines another edge the out-degree of each vertex is $k+1$ (except for the last position which has no successor). In contrast, the in-degree, of a node is unbounded: Consider, for instance, the following subset of the Łukasievicz language $\{(ab)^n b \mid n \geq 0\} \subset Ł$ which defines the set of all right degenerate binary trees. There, we have that $N(i) = \max$ if $w[i] = a$, i.e. every position with an a points to the last position in the word. We adapt the definition tiling with its numerical constraints of Thomas to tile the graphs of auto-generative languages.

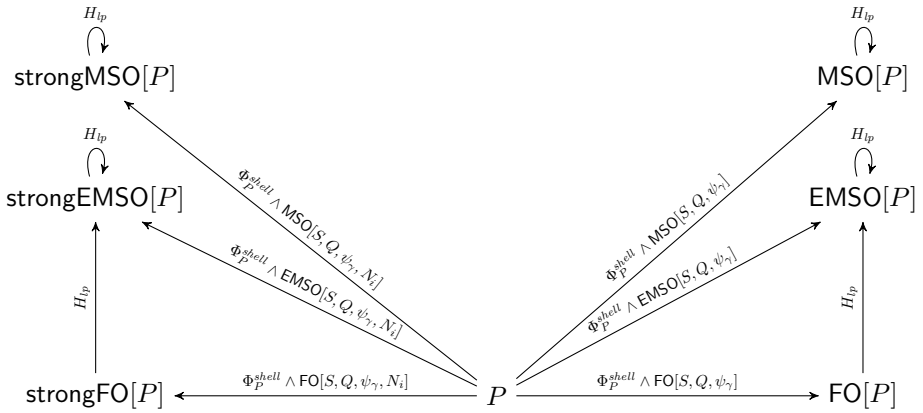
The constraints on the tiles allow us to define *shrinking* for auto-generative languages and their complements. If a word of an auto-generative language is *long enough* such that the tiling constraints are *(over-)satisfied* we can constructively shrink it to a shorter word in the language. Iteratively shrinking a word at some yields a *shortest* word which cannot be shrunk anymore. We use this principle to show the decidability of the emptiness of protocol languages. If a word is not in an auto-generative language the shrinking is defined on the

(not necessarily uniquely determined) graph of the word that can be tiled with the maximal number of *good* tiles. The tiling and shrinking of words in the complement is currently only defined existentially – future work in this subject might yield constructive shrinking arguments for complements of protocol languages (cf. [San19]) We define *protocol languages* as the tile-shrinkable auto-generative languages.

We propose logical extensions of protocol languages. The *shell* of a protocol language verifies that the pointer structure of a word is correct, but does not test for any letters. For a protocol language $P \subseteq \Gamma^*$ we define $\mathsf{FO}[P]$, $\mathsf{MSO}[P]$, $\mathsf{EMSO}[P]$, $\mathsf{strongFO}[P]$, $\mathsf{strongMSO}[P]$ and $\mathsf{strongEMSO}[P]$. These extensions are defined by formulae which are in conjunction with the formula defining the shell of P. All extensions may use the $\psi_\gamma(x)$-formulae as atomic formulae. The *strong* classes allow the conjunction of first order and monadic second order formulae respectively which additionally make use the binary second order variables defined by the protocol language. This was used to prove for instance that $\hat{\mathbb{D}}_k$, the tagged Dyck language with k pairs of parenthesis is in $\mathsf{FO}[\hat{\mathbb{D}}_1]$. This implies that the *strong* logical extensions of a protocol language P is not contained in the trio of P.

We analyzed these six extensions in regard to trio operations and showed that length-preserving inverse morphic images of a protocol language P is in $\mathsf{FO}[P]$. It was shown that $H_{lp}(\mathsf{FO}[P]) = \mathsf{EMSO}[P]$. The direction $H_{lp}(\mathsf{FO}[P]) \subseteq \mathsf{EMSO}[P]$ was proved by using the available monadic variables to existentially guess the morphic pre-image – the same approach could be applied to prove that $H_{lp}(\mathsf{strongFO}[P]) \subseteq \mathsf{strongEMSO}[P]$ and that $\mathsf{EMSO}[P]$, $\mathsf{MSO}[P]$ $\mathsf{strongEMSO}[P]$ and $\mathsf{strongMSO}[P]$ are closed under length-preserving homomorphisms. The other inclusion, $\mathsf{EMSO}[P] \subseteq H_{lp}(\mathsf{FO}[P])$, was obtained with a standard technique: To *recreate* a monadic second order formula with first order means, the alphabet of the first order formula is chosen as the power-set of the monadic second order variables in combination with the actual letter predicates.

$\mathsf{FO}[P]$ still captures languages which are deterministic/visible. As the homomorphism are not necessarily injective, $\mathsf{MSO}[P]$ also contains non-deterministic languages. Both extensions are still contained in the trio generated by the protocol language P. This is not the case for the *strong* extensions of P, as for instance $\hat{\mathbb{D}}_k$ is not in the trio generated by $\hat{\mathbb{D}}_1$.



An open question is whether one can extend the closure of $\mathsf{MSO}[P]$ under length-preserving morphisms to more general morpishms. One approach could be to use monadic variables to non-deterministically *guess* for each position some tile's core and make use of the shrinkabilty of protocol languages which could be used to give a bound on the length of the pre-image. Showing the closure under erasing morphisms would yield a language family that stands in strong contrast to complexity classes, as they are typically not closed under erasing

morphisms.

The length-preserving inverse morphisms of a protocol language P are contained in $\mathsf{FO}[P]$ and $\mathsf{MSO}[P]$. In Example 4.7.13 we demonstrate how to construct a formula for an inverse morphism of Ł which is not length-preserving by using additional pointer relations. We conjecture that for a protocol language $P \subseteq \Gamma^*$ in $\mathsf{unEx}k\mathsf{FO}[S]$ and some arbitrary homomorphism $f\colon Y^* \to \Gamma^*$ with $l := \max\{|f(y)| \mid y \in Y\}$ the inverse image $f^{-1}(P)$ is auto-generative in $\mathsf{unEx}(k+1)l\mathsf{FO}[S]$. A thorough treatment of closures properties regarding inverse morphisms for fragments of monadic second order logic was done by Kufleitner and Lauser [KL12b], which might be a fruitful source for techniques applicable for the logic fragment used for protocol languages.

We gave protocol languages for regular, context-free and indexed languages. Starting with the context-free languages we have shown that the Łukasievicz language is a protocol language. We proved that the restricted real-time one counter languages are in $\mathsf{MSO}[Ł]$. As Ł is the derivation language of the context-free grammar in Chomsky normal form with one non-terminal and one terminal symbol, we showed that $\mathcal{G}_k$ is the derivation language for analogous context-free grammars in Greibach normal form with right-hand sides up to length $k+1$. We proved that all context-free languages that do not contain λ are in $\mathsf{strongMSO}[\mathcal{G}_2]$. If one were to show that $\mathsf{strongMSO}[P]$ is closed under erasing morphisms, it would yield that $\mathsf{CFL} = \mathsf{strongMSO}[Ł]$, the derivation left-derivations of context-free grammars in Chomsky normal form are in $\mathsf{strongFO}[Ł]$ and erasing morphisms could remove any bifurcating rule.

For the regular languages we gave the protocol languages $\mathcal{G}_1$ and a^* in $\mathsf{unEx0FO}[S]$ which is equal to $\mathsf{FO}[S]$. This equivalence yields that $\mathsf{Reg} = \mathsf{MSO}[a^*] = \mathsf{strongMSO}[a^*]$ and $\mathsf{Locally\ Threshold\ Testable} = \mathsf{FO}[a^*] = \mathsf{strongFO}[a^*]$.

We defined the analogous language to Ł for deviation languages indexed grammars in normal form which we call Z and gave proof sketches that Z is a protocol language in $\mathsf{unEx2FO}[S]$. We point the interested reader to [San19] for the full proofs of the auto-generativity of Z and how it can be tiled and shrunk.

As we have a series of *natural* protocol languages which are all deviation based for $\mathsf{OI}^0 = \mathsf{Reg}$, $\mathsf{OI}^1 = \mathsf{CFL}$ and $\mathsf{OI}^2 = \mathsf{Index}$ an open question is to find protocol languages for higher levels in the OI-hierarchy.

We demonstrated that the *tagged* version of the Copy language is a protocol language and that the NP-complete problem Sat is in $\mathsf{MSO}[\widehat{\text{Copy}}]$. Another open question is if Set, the set of languages recognized by *set automata* [LR96] contains protocol languages – a particular candidate that Set does not contain protocol languages could be Element-Distinctness $= \{w_1\#w_2\#\dots w_n \mid n \geq 1, w_i \in \{a,b\}^* \wedge i \neq j \Rightarrow w_i \neq w_j\}$ which can be viewed as a *complementary* language to the Copy language.

The decidability of the emptiness of a protocol language P was obtained with tiling and shrinking arguments. We proved that the emptiness of inverse length-preserving morphic images of protocol languages $H_{lp}^{-1}(P)$ is also decidable. This was done by constructing a new set of tiles where the tile-labels were modified according to the homomorphism. The same idea was used to show that the emptiness of $P \cap \mathsf{Loc}$ and $H_{lp}^{-1}(P) \cap \mathsf{Loc}$ is decidable which implies that $int_{\mathsf{Reg}}(P)$ is decidable. The decidability of $int_{\mathsf{Reg}}(P)$ implies the that the emptiness of regular intersection is decidable for the full AFL of P.

It is still an open question whether the logical extensions $\mathsf{FO}[P]$ and $\mathsf{MSO}[P]$ and their *stronger* versions have a decidable emptiness.

The analysis of protocol languages in regard to the question of *formality* of languages families is still in its earlier stages. If the series of a^* and $\mathcal{G}_2$ for the regular and context-free languages

respectively could be generalized to higher levels of the OI-hierarchy and to their sub-families, one hypothesis could be that every family of formal language $\mathcal{F}$ is equal to $\mathsf{strongMSO}[P_{\mathcal{F}}]$ for some protocol language $P_{\mathcal{F}}$.

If such a characterization of families of formal languages should be possible, the question of the *robustness* of this notion remains open: Can complexity classes be expressed with extensions of protocol languages? Fagin [Fag73, Fag74] proved that the languages definable with existential second order logic are precisely the languages in NP, which was extended by Stockmeyer [Sto76] to $\mathsf{SO} = \mathsf{PH}$. Since the logic used to define protocol languages and their extensions are a fragment of second order logic, showing for a complexity class $\mathcal{C}$ with $\mathsf{PH} \subseteq \mathcal{C}$ that a $\mathcal{C}$-complete language is a protocol language would imply that $\mathsf{PH} = \mathcal{C}$. Therefore, it is rather unlikely that complexity classes *above* PH could be characterized by second order logic. The upper bound PH also gives rise to speculation that there are no *hiding*-arguments for protocol language, i.e. it is unlikely that for every decidable language L one can construct a language L' of the same complexity such that L' is a protocol language.

Showing the closure of $\mathsf{MSO}[P]$ under erasing morphisms would also strengthen the theory of describing *formality* with protocol languages, as complexity classes typically lack this particular closure property.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[Aho68] Alfred V. Aho. Indexed grammars—an extension of context-free grammars. *Journal of the ACM (JACM)*, 15(4):647–671, 1968.

[Ber79] Jean Berstel. *Transductions and context-free languages*. Teubner, Stuttgart, 1979.

[Büc60] Julius R. Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

[CK98] Hugues Calbrix and Teodor Knapik. A string-rewriting characterization of muller and schupp's context-free graphs. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 331–342. Springer, 1998.

[Dam82] Werner Damm. The IO-and OI-hierarchies. *Theoretical Computer Science*, 20(2):95–207, 1982.

[DPRS97] Jürgen Dassow, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. Handbook of formal languages, 1997.

[DPS79] Jürgen Duske, Rainer Parchmann, and Johann Specht. Szilard languages of io-grammars. *Information and Control*, 40(3):319–331, 1979.

[EGG00] Thomas Eiter, Georg Gottlob, and Yuri Gurevich. Existential second-order logic over strings. *Journal of the ACM (JACM)*, 47(1):77–131, 2000.

[Elg61] Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98:21–51, 1961.

[Eng02] Joost Engelfriet. The delta operation: from strings to trees to strings. In *Formal and natural computing*, pages 39–56. Springer, 2002.

[Eng14] Joost Engelfriet. Context-free grammars with storage. *arXiv preprint arXiv:1408.0683*, 2014.

[Fag73] Ronald Fagin. *Contributions to the model theory of finite structures.* University of California, Berkeley, 1973.

[Fag74] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation*, 1974.

[Fis68] Michael J. Fischer. Grammars with macro-like productions. In *Switching and Automata Theory, 1968., IEEE Conference Record of 9th Annual Symposium on*, pages 131–142. IEEE, 1968.

[GKLW18] Demen Güler, Andreas Krebs, Klaus-Jörn Lange, and Petra Wolf. Deciding regular intersection emptiness of complete problems for PSPACE and the polynomial hierarchy. In *International Conference on Language and Automata Theory and Applications*, pages 156–168. Springer, 2018.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. Introduction to automata and formal languages. *Reading: Addison-Wesley*, 1979.

[Jul61] Julius R. Büchi. On a decision method in restricted second order arithmetic. In *Congress on Logic Methodology and Philosophy of Science*, pages 133–144. Stanford University Press, 1961.

[KL12a] Andreas Krebs and Klaus-Jörn Lange. Dense completeness. In *International Conference on Developments in Language Theory*, pages 178–189. Springer, 2012.

[KL12b] Manfred Kufleitner and Alexander Lauser. Lattices of logical fragments over words. In *International Colloquium on Automata, Languages, and Programming*, pages 275–286. Springer, 2012.

[Koz12] Dexter C. Kozen. *Automata and computability.* Springer Science & Business Media, 2012.

[Lan96] Klaus-Jörn Lange. Complexity and structure in formal language theory. *Fundamenta Informaticae*, 25(3, 4):327–352, 1996.

[LR96] Klaus-Jörn Lange and Klaus Reinhardt. Set automata. In *Combinatorics, Complexity and Logic; Proceedings of the DMTCS'96.* Springer, 1996.

[LST94] Clemens Lautemann, Thomas Schwentick, and Denis Thérien. Logics for context-free languages. In *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, pages 205–216, 1994.

[Mäk84] Erkki Mäkinen. On context-free and szilard languages. *BIT Numerical Mathematics*, 24(2):164–170, Jun 1984.

[Mon87] Burkhard Monien. About the complexity of the Derivation Languages of Index Grammars. *manuscript*, 1987.

[MP71] Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. Research Monograph No. 65).* The MIT Press, 1971.

[MP11] Parthasarathy Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In *ACM SIGPLAN Ntotices*, volume 46, pages 283–294. ACM, 2011.

[MS85] David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[Niv68] Maurice Nivat. Transductions des langages de chomsky. *Ann. Inst. Fourier*, 18(1):339–455, 1968.

[Pap94] Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[San19] Magdalena Sannwald. N-structures and protocol languages. Master's thesis, Eberhard Karls Universität Tübingen, 2019. Forthcomming.

[Sil19] Silke Czarnetzki. *Duality in Computer Science.* PhD thesis, Eberhard Karls Universität Tübingen, 2019. Forthcoming.

[Sto76] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[Str94] Howard Straubing. *Finite automata, formal logic, and circuit complexity.* Birkhäuser, Boston, 1994.

[Tho82] Wolfgang Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25(3):360–376, 1982.

[Tho91] Wolfgang Thomas. On logics, tilings, and automata. In *International Colloquium on Automata, Languages, and Programming*, pages 441–454. Springer, 1991.

[Tra61] Boris Avraamovich Trakhtenbrot. Finite automata and the logic of single-place predicates. In *Doklady Akademii Nauk*, volume 140, pages 326–329. Russian Academy of Sciences, 1961.

[VL75] Jan Van Leeuwen. The membership question for ETOL-languages is polynomially complete. *Information Processing Letters*, 3(5):138–143, 1975.

[Wol18] Petra Wolf. Decidability of the Regular Intersection Emptiness Problem. Master's thesis, Universität Tübingen, 2018.

[Wra76] Celia Wrathall. Complete Sets and the Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.