Typed Semigroups, Majority Logic, and Threshold Circuits

Dissertation

der Fakultät für Informations- und Kognitionswissenschaften
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer.nat.)

vorgelegt von
Dipl.-Inf. Andreas Krebs
aus Stuttgart

Tübingen
2008

Tag der mündlichen Qualifikation:	16.07.2008
Dekan:	Prof. Dr. Michael Diehl
1. Berichterstatter:	Prof. Dr. Klaus-Jörn Lange
2. Berichterstatter:	Prof. Howard Straubing, Ph.D.

Zusammenfassung

Eine der Gründe für den Erfolg der Mathematik ist die Tatsache, dass es ihr immer wieder gelingt scheinbar unzusammenhängende Teilgebiete miteinander zu verknüpfen. Diese Verbindungen erlauben es Methoden und Einsichten, die für ein Gebiet gefunden wurden, auf das andere anzuwenden und die wechselseitige Befruchtung führt oft zu neuen Erkenntnissen. Diese Dissertation befindet sich thematisch im Bereich der formalen Sprachen, in dem Halbgruppentheorie, Logik und Komplexitätstheorie zusammentreffen.

Endliche Halbgruppen haben als Transformationshalbgruppen eine enge Beziehungen zu endlichen Automaten, die zum Erkennen von formalen Sprachen verwendet werden. Mit der Logik der ersten und zweiten Stufen können formale Sprachen durch logische Formeln beschrieben werden. Schaltkreise mit konstanter Tiefe und polynomieller Größe bilden die Brücke zu der Komplexitätstheorie.

Das Ziel ist dabei Komplexitätsklassen voneinander zu trennen, wobei über die formalen Sprachen die Beschreibungen der Komplexitätsklassen in der Logik oder durch Halbgruppen untersucht werden. Durch diese Verbindung gibt es hier in der Komplexitätstheorie, wenn auch nur wenige, so doch nichttriviale Trennungsresultate.

In dieser Dissertation werden die Beziehungen, die bisher nur für reguläre Sprachen bekannt waren, auf beliebige Sprachklassen erweitert. Genauer gesagt wird das Varietätentheorem von Eilenberg über die Korrespondenz zwischen Varietäten von regulären Sprachen und Varietäten von endlichen Halbgruppen, auf eine Korrespondenz zwischen beliebigen Varietäten von Sprachen und sogenannten *getypten Halbgruppen* erweitert.

Dann wird diese Beziehung verwendet, um Logikklassen, die bisher nicht algebraisch betrachtet werden konnten, zu analysieren. Wir beschäftigen uns mit Majority Logik und analyiseren die regulären Sprachen, die sich als Majority Formeln mit zwei Variablen schreiben lassen. Außerdem wird ein Trennungsergebnis von Logikklassen gezeigt, das auch eine Trennung von Schaltkreisklassen nach sich zieht.

Im folgenden werden die Ergebnisse der Dissertation genauer dargestellt. Mit der Definition der *getypten Halbgruppen* wird die Grenze von regulären Sprachen und endlichen Halbgruppen überschritten. Es wird gezeigen, dass die getypten

Halbgruppen auf natürliche Weise eine Kategorie bilden, und das Varietätentheorem von Eilenberg von Eilenberg für die Kategorie erweitert wird.

Um besser auf einige Details von getypten Halbgruppen eingehen zu können, wird die übliche Situation beim Erkennen von Sprachen per Homomorphismus genau aufgezeigt: Sei L eine formale Sprache über dem Alphabet Σ und S die syntaktische Halbgruppe. Folglich gibt es einen syntaktischen Morphismus h von Σ^+ nach S und eine Teilmenge $A \subseteq S$, so dass sich die Sprache als Urbild von dieser Menge schreiben läßt: $L = h^{-1}(A)$. Es gibt also neben der syntaktischen Halbgruppe noch weitere algebraische Objekte,die die Sprache beschreiben: den Homomorphismus h und die akzeptierende Menge A. Diese beide Objekte werden in die Definition von getypten Halbgruppen einfließen.

Eine getypte Halbgruppe besteht aus einem Tripel: einer Halbgruppe, einer Booleschen Algebra über der Halbgruppe, und einer Menge von Einheiten. Die Typen sind die Elemente der Boolesche Algebra und werden verwendet, um die akzeptierenden Teilmengen einzuschränken, während die Menge der Einheiten verwendet wird, um einen Längenbegriff auf den getypten Halbgruppen zu erhalten.

Um eine strukturelle Beziehung zwischen Logik und den getypten Halbgruppen herzustellen, wird das Block Produkt für getypte Halbgruppen verallgemeinert. Dadurch kann für jede Logik, die durch eine Menge von Quantoren und eine Menge von Prädikaten definiert ist, eine algebraische Charakterisierung durch getypte Halbgruppen angeben werden. Dazu wird das Block Produkt Prinzip, das im endlichen Fall für Logik mit zwei Variablen verwendet wurde, auf unendliche Halbgruppen erweitert und erstmalig auch konsequent für den Fall von unbeschränkt vielen Variablen verwendet.

Für Schaltkreisklassen wird ein gleichartiges Resultat erzielt: Jede Schaltkreisklasse, die durch ihre Basis (=Gattertypen) und ihre Uniformität festgelegt ist, kann durch eine Klasse von getypten Halbgruppen charakterisiert werden. Um verbesserte Resultate zu erhalten, wird eine eigene Definition der Uniformitätssprache, die näher an der Logik orientiert ist, verwendet.

Diese neue algebraische Kategorie wird verwendet um Ergebnisse für Majority Logik und Threshold Schaltkreise herzuleiten. Es stellt sich heraus, dass Majority Logik in einem direkten Zusammenhang zu Block Produkten der ganzen Zahlen steht. Im Fall, dass die Logik auf zwei Variablen beschränkt ist, läßt sich das Block Produkt in $\mathbb{Z}$ Module zerlegen.

Diese lassen sich leicht in die Euklidische Geometrie einbetten. Mit Hilfe dieser Interpretation können Majority Formeln mit nur einem Quantor als einen Halbraum aufgefassen werden, und so Formeln der Tiefe eins als Mengen von Halbräumen beschrieben werden. Dies ermöglicht es induktiv für eine Majority Formel, die nur zwei Variablen besitzt, Paare von Wörter zu konstruieren, die sich von der Formel

nicht unterscheiden lassen. Da geometrische Beweise zwar intuitiv zu verstehen sind, aber nicht leicht zu verifizieren, werden sie algebraisch bewiesen.

Diese geometrische Auffassung erlaubt es eine obere und untere Schranke für die Menge der regulären Sprachen anzugeben.

Diese Logik wird schließlich noch um alle regulären und unären Predikate, sowie Modulo Quantoren erweitert, und an Hand der algebraischen Charakterisierung wird gezeigt, dass auch in dieser Erweiterung noch nicht alle regulären Sprachen erkannt werden können. Dies führt auf der Schaltkreisseite zu einer Trennnung von konstant tiefen und linear großen Threshold Schaltkreisen, die eine bestimmte Uniformität haben, von linear großen und logarithmisch Tiefen Schaltkreisen.

Preface

This thesis covers part of my research in the area of circuits, logic and algebra. I thank my adviser Klaus-Jörn Lange for his guidance and advice, and for giving me the freedom of doing this research work.

I am grateful for the scientific framework, especially the "MiniAG" – Christoph Behle, Klaus-Jörn Lange, Stephanie Reifferscheid – for providing a fruitful environment for discussion and research. Special thanks to Christoph Behle for his insistence on improving the quality of our papers, and Stephanie Reifferscheid for forcing mathematical correctness in our papers. Also many thanks to my coauthor Mark Mercer for his work and fruitful discussions on logic with two variables. This collaboration manifested in many papers "Characterizing TC^0 in Terms of Infinite Groups" [KLR05, KLR07], "Linear Circuits, Two-Variable Logic and Weakly Blocked Monoids" [BKM07], "Separating a subclass of TC^0 from NC^1" [BKRb] and "Regular Languages definable by Majority Quantifiers with two Variables" [BKRa].

I also want to thank Pascal Tesson for profitable discussions on logic and semigroups. Also many thanks to thank Pierre McKenzie for discussions on circuits and his moral advice on how to write papers.

I thank Viola Brunner and Martin Gruber for careful proofreading most parts of this thesis.

The Chapters 3 and 4 have their basis in [KLR05, KLR07], where finitely typed groups were introduce. These definitions are extended and presented in such a way that they match with abstract categorical definitions.

Chapter 4 has roots in [BKM07] where similar methods were used to show that majority logic with two variables corresponds to certain restricted morphisms into finitely typed monoids.

The Chapters 6 and 7 about majority logic with two variables originate from the yet unpublished papers "Regular Languages definable by Majority Quantifiers with two Variables" [BKRa] and "Separating a subclass of TC^0 from NC^1" [BKRb] though the proofs are presented in a different way.

Contents

List of Figures

Chapter 1

Introduction

If a new result has value it is when, by binding together long known elements, until now scattered and appearing unrelated to each other, it suddenly brings order where there reigned apparent disorder.

Henri Poincaré

Mathematics often plays a unifying role, tying together seemingly different areas of science and allowing results from one area to be carried over to the other. It is precisely the connection between algebra, logic and complexity theory that motivates the fascination with our subject of study, the mathematical theory of formal languages.

One of the first discoveries of concern to us is Kleene's theorem, that builds bridges between formal languages and algebra by way of a relation between regular languages and finite automata. Equally seen as a bridge between regular languages and transformation semigroups, this relation led to Eilenberg's in-depth study [Eil76] of the correspondence between varieties of regular languages and varieties of finite semigroups.

Büchi [Büc60] then brought together logic and formal languages, proving that the languages that could be described by monadic second order logic with order are exactly the regular languages. McNaughton and Pappert [MP71] refined this with a proof that the languages describable by first order logic with order are exactly the starfree languages.

It is not obvious how to tell whether a language is starfree or equivalently whether it is describable by a first-order formula with order. Fortunately, Schützenberger [Sch65] had shown that the starfree languages correspond to the aperiodic semigroups. Since aperiodic semigroups are exactly those semigroups that contain no groups, a simple algorithm can tell whether an appropriately presented semigroup is aperiodic. The results of [Sch65] and [MP71] thus allow carrying this algorithm over to formal languages and logic, yielding decidability results there.

These early results triggered extensive research that uncovered a broad entanglement between extensions of first order logic and classes of semigroups. The

block product defined by Rhodes and Tilson [RT89], used as a basic building block in semigroup theory, arose as a fundamental tool to prove these connections [Str94, STT95, CPS06]. From a more general perspective, Straubing exposed a meta-explanation for this phenomenon [Str02].

Temporal logic is another area that became integrated into the framework. Although the quantifiers used in linear temporal logic are quite different from first order logic quantifiers, Kamp [Kam68] showed that the languages describable by linear temporal logic are exactly the starfree languages. From the study of restricted temporal logic emerged a tight link between this logic and first-order logic with only two variables [CPP93, EVW97, TW98, ST03]. This also gave more weight to results within first-order logic [PW97]. Research on extensions of first order logic restricted to two variables brought to light some new fruitful connections with algebra [EVW97, TW98, ST03], suggesting that natural restrictions in logic have natural counterparts in algebra.

Complexity theory finally entered the scene when constant depth circuits of polynomial size were connected to first-order logic with arbitrary numerical predicates. This was first discovered by [GL84, Imm87] who noticed that constant depth circuit families with AND and OR gates recognize exactly the languages describable by first-order logic with arbitrary numerical predicates. Emerging from the study of branching programs, the computational model of a program over a finite semigroup was then defined and shown closely tied to circuits [BT88].

As could be expected, programs over various classes of semigroups were found to have as natural counterparts circuit families with various gate types. Ample results regarding different sets of gate types confirmed the robustness of the connection [Str94, LMSV01, Str92, RS06, KLPT06]. Also, the previously examined logic restriction involving two variables was shown to have, as its counterpart, linear size circuits [KLPT06, KPT05, TT05].

An important issue when comparing circuit complexity classes with the logic classes and semigroups described above is uniformity. Circuits are highly nonuniform, i.e. for each input length there is a separate structure recognizing the set of words of that length in the language, whereas logic and algebra are uniform in the sense that the same formula or semigroup is used for all input lengths. Given the wide uniformity gap between the two models, there are two possibilities to set up a meaningful framework common to circuit families on the one hand and to logic and algebra on the other.

One is to weaken the uniformity of logic and algebra, which is accomplished in logic by using numerical predicates that depend on the length of the input and programs on the algebraic side. The other possibility is to limit the structural change of the circuits for one input length to another. Barrington, Immerman and Straubing [BIS90] used the notion of a uniformity language, to show connections between DLOGTIME-uniform circuits and logic classes with addition and multiplication, and by a result of Behle and Lange this connection was extended to more uniform circuits corresponding to logic with other sets of predicates [BL06].

Uniformity can also be enforced indirectly, for instance by restricting the languages in such a way that the length of a word contains only limited information. Demanding a neutral letter can be viewed as such a restriction. The ill-fated Crane Beach conjecture postulated that a neutral letter would impose the same level of uniformity as the tightest uniformity reasonably imposed on circuits. Unfortunately this intuition failed and the conjecture was proven false [BIL$^+$05]. Performing the outright intersection of a uniform class with the class of interest can also be viewed as imposing uniformity on the latter [MTV08].

One of the circuits that are rather disparate are threshold circuits as introduced in [HMP$^+$87, HMP$^+$93]. Unlike previous circuits they can recognize nonregular languages by the very character of their gates even in very uniform settings. Though a counting or majority quantifier creates a logical characterization of threshold circuits, this link is not as pleasant as for the other gate types. It is for example necessary to differentiate between majority quantifiers over single variables and those over tuples as examined in [BIS90]. On the other hand majority formulas containing only the order predicate can already simulate addition as presented by Lange [Lan04].

Since the nature of a threshold gate is non-regular, connections to algebra seem to have no counterparts as programs or even morphisms over finite semigroups. Despite that, or actually because of it, a characterization by infinite structures was given in [KLR07] providing a basis to view the languages in this circuit class as inverse morphic images of this infinite structure. This link works for various predicate sets, and also the restriction in logic to two variables has the desired algebraic correspondence [BKM07].

On the algebraic side, separating classes of semigroups is relatively easy, compared to separating complexity classes. Since the latter task is very intricate, there are only few separation results known. The sparse results are thus surveyed even more, starting with the result of Sipser [FSS81] presenting a combinatorial proof that AC^0 cannot compute parity, thus $AC^0 \neq ACC^0$. Improving this Yao and Hastad showed that this requires even exponential circuits [Yao85, Hås87]. Further, Razborov showed in [Raz87] that $ACC^0[2]$ circuits cannot compute modulo 3, which was enhanced by Smolensky showing that $ACC^0[p]$ for a prime p cannot compute modulo q unless q is a power of p. The results of [BST90] leading to a separation of $CC^0[p]$ and $ACC^0[p]$ showed that AND cannot be computed by $CC^0[p]$ for a prime p.

For TC^0 there are some results suggesting a separation from NC^1. Hajnal et. al. [HMP$^+$87, HMP$^+$93] gave combinatorial proof separating TC^0 circuits of depth two from depth three by an explicit regular language. A quite different result of Ruhl [Ruh99] and Lautemann, McKenzie, Schwentick, Vollmer [LMSV01] asserted that multiplication is not computable by $MAJ[<,+]$ formulas, which separates $MAJ[<,+]$ from NC^1 by means of uniformity.

The current frontier on splitting complexity classes, even for the ones considered, is disappointing, so we prefer to summarize the open questions. Neither of the complexity classes $CC^0[q]$, $ACC^0[q]$ for q not a prime power, e.g. 6, is separated

from NP yet, though it is conjectured they fail to compute L_{AND} in the first case, and L_{MOD_p} in both cases unless p divides q. Both are contained in TC^0, implicating that there is no known upper bound for $TC0$ below NP, but again there are reasons to believe that not all regular languages are in TC^0. As a consequence of [Bar89], the only candidates are the group languages for non-solvable groups, and moreover either all regular languages are in TC^0 or none of the non-solvable group languages.

Results of this thesis

In this thesis we will go beyond regular languages and finite semigroups. We extend the theory of Eilenberg providing a correspondence between regular languages and finite semigroups, to a correspondence between arbitrary languages and *typed semigroups*. Previous definitions of categories of infinite objects [Sak76] have the basic drawback, that they have no structural correspondence to logic or circuit theory. We avoid these problems in our category of typed semigroups.

Before giving more details on the typed semigroups, we look at an arbitrary language $L \subseteq \Sigma^+$ and its syntactic semigroup S. The syntactic morphism $h : \Sigma^+ \to S$ guarantees there is a set $A \subseteq S$ such that $L = h^{-1}(A)$. So besides the semigroup S we have a morphism h and an accepting set A, describing the language. We will use this to define our new algebraic structure, the typed semigroup $(S, \mathfrak{S}, \mathcal{E})$, consisting of a triple: a semigroup S, a Boolean algebra $\mathfrak{S}$ over this semigroup, and a set of elements of the semigroup $\mathcal{E}$. The elements of the Boolean algebra $\mathfrak{S}$ are sets of S and will be the only sets allowed as accepting sets. In the case that the semigroup $(S, \mathfrak{S}, \mathcal{E})$ is not free the set $\mathcal{E}$ will allow us to define a notion of length preserving morphism, by the requirement that h maps single letters to $\mathcal{E}$. Together with the notion of a typed morphism the typed semigroups form a category.

We can embed the category of finite semigroups into the typed semigroups, which shows that the regular languages are just a special case of the classes of languages captured in our theory. Using typed semigroups we can prove theorems even in the finite case that cannot be stated only in the terms of semigroups without types and units. We can, for example, give an algebraic counterpart to the logic class FO[mod], since typed semigroups enable us to differentiate between the modulo predicates and the modulo quantifiers. This extends the result of Eilenberg, yielding a correspondence between any class of languages and a class of typed semigroups, giving a much finer structure previously known only for varieties of regular languages.

In order to describe the typed semigroups corresponding to logic classes we introduce a block product in a similar way to [KLR07], being more involved than in the finite case. Using this definition we show that for any logic class given by a set of quantifiers and a set of predicates, there is a class of typed semigroups that recognizes exactly the set of languages recognized by the logic class. We prove this by adopting the block product principle [Str94, TW04] to typed semigroups and free variables, where the latter allows to use the block product principle for an unbounded number of variables. So given any set of quantifiers, any set of predicates, a fixed or unbounded

number of variables, a fixed way the quantifiers are nested or arbitrary nesting, we get an algebraic characterization using the block product for this class of languages.

We obtain a similar result for circuit classes where we can find a direct correspondence to classes of typed semigroups. For this purpose we introduce a new form of the uniformity language, giving tighter connections between logic, circuits and algebra. The results can also be modified for previously known versions of the uniformity language. Again this results, for any set of gate types, uniformity and linear or polynomial size, in an algebraic characterization of the languages recognized by these circuit families.

Having established this algebraic theory we apply it to majority logic and threshold circuits. The algebraic characterization of $\mathrm{MAJ}_2[<]$, i.e. $\mathrm{MAJ}[<]$ with only two variables, results in the smallest variety closed under weak block products with powers of $\mathbb{Z}$. We can embed the powers of $\mathbb{Z}$ nicely in the Euclidean space. Using this embedding we can interpret majority formulas of depth one as half planes. Given a fixed formula with this intuition one can construct a pair of words that cannot be separated by the formula.

Proofs relying to much on geometric intuition tend to be improper. To avoid this caveat we go the elaborate way of transforming the proofs to algebra, where all the intuitive steps can be computed, and hence be checked easily. So given any fixed language or variety described by equations with the techniques developed it is an easy task to see whether a language belongs to $\mathrm{MAJ}[<]$ with two variables or not.

Using the characterization of a variety by equations we are able to give an upper bound. As a formal tool we use prefix and suffix mappings or later restrictions that perform a similar task as in the proof of [FSS81] by fixing certain positions of the input.

We also extend the logic class from $\mathrm{MAJ}_2[<]$ to $(\mathrm{FO}+\mathrm{MOD}+\widehat{\mathrm{MAJ}})_2[\mathrm{reg}, \mathrm{arb}-\mathrm{un}]$ and show that this logic still cannot recognize any non-solvable semigroup language, hence separating this class from NC^1. Compared to the master goal to separate TC^0 from NC^1 this may be considered only a small step, but TC^0 equals $\mathrm{MAJ}[\mathit{arb}]$ and we argue that being interested only in regular languages the gap between these two logic classes is not big. Our separation result may therefore be extended to a separation of TC^0 from NC^1 in future work.

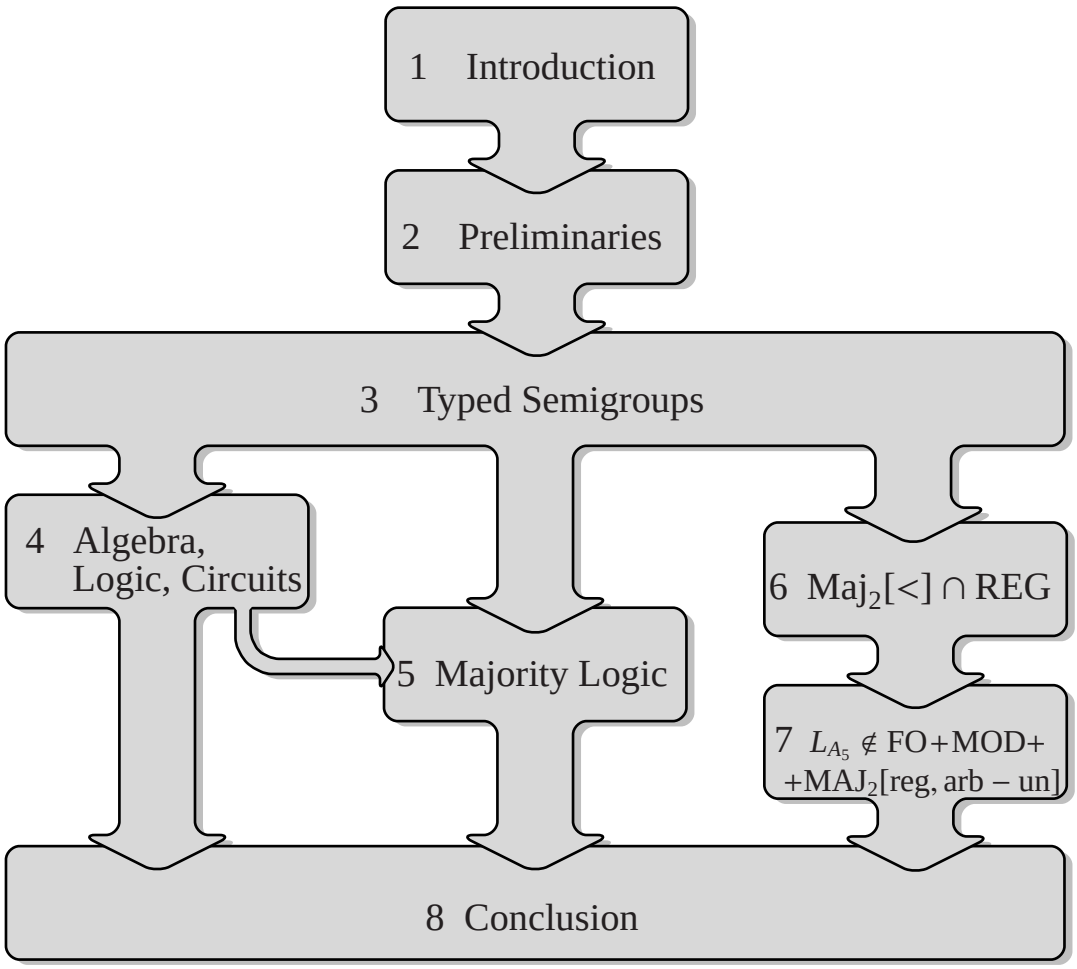
Structure of the thesis

The thesis is structured in the following way. We begin with the preliminaries reviewing the basics that are commonly used in logic, algebra and circuit theory. We move on to the chapter about typed semigroups, where we extend the notion of the syntactic semigroup in a way useful to the non-regular case. Typed semigroups are then used to characterize varieties of (non-regular) languages by morphisms. This allows us to expose connections between algebra, logic and circuits in the following chapter, where we use typed semigroups instead of finite semigroups.

Thereupon we examine majority logic for different quantifiers and predicate sets and explore how to find a simpler algebraic characterization than the characterization by the previous constructions. We also show some finer connections between logic and algebra in the case of threshold quantifiers.

Having laid out the needed tools we apply them in chapter *Regular languages in* $\mathrm{MAJ}_2[<]$, to find an upper and lower bound for the power of majority logic when recognizing regular languages. Finally we show that even by extending majority logic by the first-oder and modulo quantifiers and the regular predicates are still not able to recognize non-solvable group, thus separating this class from NC^1.

Since this thesis lays out a complete framework for an algebraic counterpart for logic and circuits and then applies it to prove a non-trivial upper bound, some chapters might be skipped. We give an overview in the following graphic.



Chapter 2

Preliminaries

In this chapter we present the necessary definitions and facts commonly used in the areas of logic, algebra and circuits, and give a short review of this topic. We will not give a complete survey of the topic since this would go beyond the scope of this thesis; for a complete treatment we recommend Straubing's book ([Str94]). For a brief introduction the survey article of Tesson and Therien ([TT07]) on logic and algebra covers everything needed here.

2.1 Basics

We denote by $\mathbb{Z}$ the set of the integer, by $\mathbb{Z}^+$ the positive integers and by $\mathbb{Z}_0^-$ the negative integers and zero. The set of natural numbers is denoted by $\mathbb{N}$ and the set of the square numbers by $\mathbb{S}$.

Given two sets S and T we denote by 2^S the power set of S, and by T^S a $|S|$-tuple with values in T, which is equivalent to a function $S \to T$. We will switch between these interpretations when one notion seems to be more convenient. Given a function $f : S \to T$ we write $f(s)$ for the value of f at s or simply f_s. For every set S the identity map is denoted by $\mathbf{1}_S$ or simply $\mathbf{1}$.

A binary relation $\preceq$ on a set S is a preorder if it is reflexive, i.e. $s \preceq s$ for all $s \in S$, and transitive, i.e. $s \preceq t$ and $t \preceq u$ implies $s \preceq u$ for all $s, t, u \in S$. If the preorder is antisymmetric, i.e. $s \preceq t$ and $t \preceq s$ implies $s = t$ for all $s, t \in S$ then it is a partial order.

A Boolean algebra $\mathfrak{S}$ over a set S is a subset of the power set of S that is closed under union, intersection and complement, i.e. $\mathcal{S}, \mathcal{T} \in \mathfrak{S}$ implies $\mathcal{S} \cup \mathcal{T}, \mathcal{S} \cap \mathcal{T}, S \setminus \mathcal{S} \in T$. The join $\vee$ of two Boolean algebras $\mathfrak{S} \vee \mathfrak{T}$ is the smallest Boolean algebra containing $\mathfrak{S}$ and $\mathfrak{T}$. A Boolean algebra $\mathfrak{S}$ is coarser than $\mathfrak{T}$ if $\mathfrak{S} \subseteq \mathfrak{T}$. If we have two subsets $\mathfrak{S}, \mathfrak{T}$ of the power set of S than we say $\mathfrak{S}$ is coarser than $\mathfrak{T}$, if this is true for the smallest Boolean algebras that contain $\mathfrak{S}$ and $\mathfrak{T}$.

We will work with alphabets Σ that are always finite. By Σ^* (Σ^+) we denote all words (except the empty word ε). Given a word $w \in \Sigma^*$, we denote by $|w|$ the length of

w, and by $\#_\sigma(w)$ the number of occurrences of σ in w. Also we let $\#(w)$ be the Parikh vector of the word, that is the tuple $(\#_\sigma(w))_{\sigma\in\Sigma}$. Sometimes we write $\#(w)_\sigma$ which denotes the σ component of the tuple, hence $\#(w)_\sigma = \#_\sigma(w)$.

Given a word $w = w_1 \dots w_n$ and an integer $i = 1, \dots, n$, we denote by $w_{<i}$ the prefix $w_1 \dots w_{i-1}$, by $w_{\leq i}$ the prefix $w_1 \dots w_i$, by $w_{>i}$ the suffix $w_{i+1} \dots w_n$ and by $w_{\geq i}$ the suffix $w_i \dots w_n$.

In what follows languages L are always subsets of Σ^+, that is $L \subseteq \Sigma^+$, though there is no reason why the thesis could not have been written for languages in Σ^*. The results are all the same; in some cases however it is algebraically easier to ignore the empty word. We call the languages Σ^+ and $\emptyset$ the trivial languages.

2.2 Logic over words

We use the usual notion for first order logical formulas, with the addition of generalized quantifiers. We omit a formal definition and just focus on extended quantifiers, which are similar to monoidal quantifiers used in [BIS90]. Further $\mathcal{V}$-structures as defined in the book of Straubing are utilized, though we usually denote the set of variables by X.

The atomic formulas consist of the query predicate $\mathfrak{c}_\sigma(x)$ and numerical predicates $\mathfrak{p}(x_1, \dots, x_k)$. These formulas can be combined by the usual Boolean operations as well as by extended quantifier application as follows: Given formulas $\varphi_1, \dots, \varphi_k$, a variable x and a quantifier $Q^{(k)}$, then $Q^{(k)}\ x\ \langle\varphi_1, \dots, \varphi_k\rangle$ is a formula. We call a quantifier a normal quantifier if $k = 1$ and an extended quantifier for $k > 1$. If there is no confusion possible we drop the index (k) from the quantifier to ease notation. Given a quantifier $Q\ x\ \vec{\varphi}$ we call $Q^{<y}\ x\ \vec{\varphi}$ and $Q^{>y}\ x\ \vec{\varphi}$ the relative quantifiers corresponding to Q, where $w_{y=j} \models Q^{<y}\ x\ \vec{\varphi}$ iff $w_{<j} \models Q\ x\ \vec{\varphi}$ and $w_{y=j} \models Q^{>y}\ x\ \vec{\varphi}$ iff $w_{>j} \models Q\ x\ \vec{\varphi}$.

The semantics of the query predicate $\mathfrak{c}_\sigma(x)$, the numerical predicates and the Boolean closure is defined as usual. For a quantifier $Q^{(k)}$ we have a function $\kappa_{Q^{(k)}} : (2^k)^+ \to S$, where S is a semigroup such that $w \models Q^{(k)}\ x\ \langle\varphi_1, \dots, \varphi_k\rangle$ iff $\kappa(\{j \mid w_{x=1} \models \varphi_j\} \dots \{j \mid w_{x=n} \models \varphi_j\}) \in A$ for a set $A \subseteq S$. We call quantifiers that quantify over more than one formula extended quantifiers.

By $\mathfrak{Q}[\mathfrak{P}]$ we denote all first order formulas with quantifiers in $\mathfrak{Q}$ and predicates in $\mathfrak{P}$, by $\mathfrak{Q}_2[\mathfrak{P}]$ the formulas with 2 variables only and by $\mathfrak{Q}_1[\mathfrak{P}]$ the formulas with 1 variables only or equivalently the formulas of depth 1.

Definition 2.1. Let Σ be an alphabet and X be a set of variables and φ be a formula, where all free variables of φ are in X.

We define $L_\varphi^{\Sigma,X} = \{w_{x_1=i_1,\dots,x_k=i_k} \mid w \in \Sigma^+, w_{x_1=i_1,\dots x_k=i_k} \models \varphi\}$. We simply write L_φ and omit Σ, X if the alphabet and the set of variables is clear from the context. For a set of sentences Φ, we let $\mathcal{L}^\Sigma(\Phi) = \{L_\varphi^{\Sigma,\emptyset} \mid \varphi \in \Phi\}$ be a set of languages and $\mathcal{L}(\Phi) = \bigcup_\Sigma \mathcal{L}^\Sigma(\Phi)$.

For a set of formulas Φ with all free variables in X, we let $\mathcal{P}^{\Sigma,X}(\Phi) = \{L_\varphi^{\Sigma,X} \mid \varphi \in \Phi\}$ be a set of languages and $\mathcal{P}(\Phi) = \bigcup_{\Sigma,X} \mathcal{P}^{\Sigma,X}(\Phi)$. Finally we denote the set of languages with one free variable by $\mathcal{P}_1(\Phi) = \bigcup_{\Sigma,|X|=1} \mathcal{P}^{\Sigma,X}(\Phi)$.

The following concept was first explicitly stated in [TW04], but originates from [Str94]. It was mainly used for two variable sentences, since in the definition mentioned it handles only one free variables bound by the block product principle. We define this concept in a way such that all formulas can be decomposed by the substitution principle.

Definition 2.2 (Substitution)**.** Let Σ be an alphabet, $\Phi = \{\varphi_1, \ldots, \varphi_k\}$ a set formulas over Σ and X a set of variables with one distinguished variable x. A Φ-*substitution* θ with the variable x is a function mapping any sentence ψ over the alphabet 2^Φ to a formula $\theta(\psi)$ over the alphabet Σ with free variables $X \setminus \{x\}$ by replacing each occurrence of the predicate $\mathfrak{c}_S(y)$ with the formula $\bigvee_{s\in S} \varphi_s(x := y)$. Also we define an *adjoint operation* ϑ to be a map from $\Sigma^+ \otimes X \setminus \{x\}$ to $2^\Phi \otimes X \setminus \{x\}$ by $\vartheta((w_1 \ldots w_n)_{x_1=i_1,\ldots,x_k=i_k}) = (u_1 \ldots u_n)_{x_1=i_1,\ldots,x_k=i_k}$ with $u_i = \{\varphi_j \mid w_{x=i} \models \varphi_j\}$.

Lemma 2.3. *Let θ be a Φ-substitution and ϑ be the adjoint operation, then for any formula ψ we have $w \models \theta(\psi)$ iff $\vartheta(w) \models \psi$.*

Proof. The proof in [TW04], can be adopted to arbitrary quantifiers and predicates, this being straightforward we omit it here. □

Definition 2.4 ($\Gamma \circ \Lambda$)**.** Let Γ, Λ be two classes of formulas, then $\Gamma \circ \Lambda$ are the Λ-substitutions of Γ.

Let $\mathfrak{Q}[\mathfrak{P}]$ be a class of formulas. We show that we can decompose these formulas by substitutions into formulas of $\mathfrak{Q}_1[\mathfrak{P}]$. If the formula has depth 1 this is clear. Assume the formula φ has higher depth, and let Φ be all subformulas of φ of lower depth, then φ is a Φ substitution of a formula of depth one. By induction we can decompose all formulas from the outside to the inside into formulas of depth one, i.e. $\mathfrak{Q}[\mathfrak{P}] = \mathfrak{Q}_1[\mathfrak{P}] \circ (\mathfrak{Q}_1[\mathfrak{P}] \circ (\mathfrak{Q}_1[\mathfrak{P}] \circ \ldots))$. Please note that this decomposition depends heavily on the fact that substitution is also defined for formulas with free variables.

Also if φ is a formula with only two variables, than all innermost formulas of depth one have only one free variable. Let Φ be the set of all formulas of depth one of φ, then φ is a $\mathfrak{Q}_1[\mathfrak{P}]$-substitution of a formula of lower depth. By induction we conclude all formulas with 2 variables can be decomposed from the inside out, i.e. $\mathfrak{Q}_2[\mathfrak{P}] = ((\cdots \circ \mathfrak{Q}_1[\mathfrak{P}]) \circ \mathfrak{Q}_1[\mathfrak{P}]) \circ \mathfrak{Q}_1[\mathfrak{P}]$.

For the decomposition from inside out it is important that the formulas with 2 variables have subformulas with only one free variable. Actually we could decompose arbitrary formulas depending on the nesting of the variables and obtain a much finer relation between the bracketing structure of the substitution and the nesting depth of the usage of variables.

2.3 Algebra

The tools here used from semigroup theory are rather basic, we refer to the books on semigroup theory [How95] and [Alm95], or for an introduction related more strongly to formal languages to [Eil76] and [Pin86].

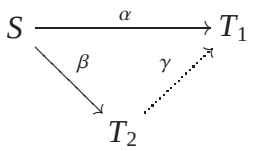
A semigroup S is a nonempty set with an associative binary operation called multiplication. If the multiplication has an identity element 1_S, the semigroup is called a monoid. Also, if in a monoid for any element $s \in S$ there is an inverse element $s^{-1} \in S$, i.e. $ss^{-1} = s^{-1}s = 1_S$, then the monoid is a group.

The smallest semigroup consists only of one element and is called the trivial semigroup denoted by $\mathbb{I}$. A generator set of a semigroup S is a subset of S such that every element of S can be written as a product of elements from S. If the generator set is fixed we call the elements of the generator set the generators. A semigroup that has a finite generator set if called finitely generated.

If a semigroup has a generator set such that element has a unique representation as a product of elements of the generator set then the semigroup is called a free semigroup. We denote by $\mathbb{F}_k$ the free semigroup with k generators.

For two semigroups S, T, a morphism $\alpha : S \to T$ is a map from S to T such that $\alpha(s_1 s_2) = \alpha(s_1)\alpha(s_2)$; a monoid morphism additionally maps 1_S to 1_T. We denote by $\mathbf{1}$ the identity morphism which is equal to the identity map. A subset $S' \subseteq S$ is a subsemigroup if it is closed under multiplication, a submonoid for a monoid S if $1_S \in S'$, and a subgroup if S' is a group.

For two morphisms $\alpha : S \to T_1, \beta : S \to T_2$ we say that α factors through β if for any pair $s_1, s_2 \in S$, $\beta(s_1) = \beta(s_2)$ implies $\alpha(s_1) = \alpha(s_2)$. In this situation we know there is a morphism γ such that the following diagram commutes.



Given a morphism $\alpha : S \to T$ we can define an equivalence relation $\equiv_\alpha$ on S by $s_1 \equiv_\alpha s_2 \iff \alpha(s_1) = \alpha(s_1)$. This relation is compatible with multiplication in S, i.e. if $s_1 \equiv_\alpha s_2$ and $s_1' \equiv_\alpha s_2'$ then $s_1 s_1' \equiv_\alpha s_2 s_2'$. Conversely, given an equivalence relation $\equiv$ on S compatible with multiplication we get a factor semigroup $S/\equiv$, consisting of the equivalence classes with the inherited multiplication. An equivalence relation compatible with the multiplication of the semigroup is known as a congruence relation.

We also have a notion for division of semigroups; we say S divides T, written $S \prec T$, if S is a morphism image of a subsemigroup of T. We define division for monoids equivalently.

Now we will briefly define the Green's relations in the way needed here. For more details we refer to the book of John Howie [How95]. For a semigroup S, two elements $s_1, s_2 \in S$ are in the same $\mathcal{D}$-class if there are elements $x_1, y_1, x_2, y_2 \in S$ such that $x_1 s_1 y_1 = s_2$ and $x_2 s_2 y_2 = s_1$. Intuitively we reach one element from the other by multiplying elements to the left or to the right. The $\mathcal{H}$-classes are a finer relation, two elements $s_1, s_2 \in S$ belong to the same $\mathcal{H}$-class iff there are elements $x_1, y_1, x_2, y_2 \in S$ such that $x_1 s_1 = s_2$ and $s_1 y_1 = s_2$ and $x_2 s_2 = s_1$ and $s_2 y_2 = s_1$. So here can choose multiplying an element to the left or to the right to reach one element from the other.

For formal language theory semigroups play an important role, especially in

[Eil76] the connection between the transformation semigroup of a finite state automaton and the language recognized by the automaton is made. A language $L \subseteq \Sigma^+$ is recognized by a semigroup S iff there exists a morphism $h : \Sigma^+ \to S$, and a subset $A \subseteq S$ such that $L = h^{-1}(A)$.

Definition 2.5. Let Σ be an alphabet and X be a set of variables and S be a semigroup. We let $\mathcal{L}^{\Sigma}(S) = \{L \mid L \subseteq \Sigma^+ \text{ is recognized by } S\}$ and $\mathcal{L}(S) = \bigcup_{\Sigma} \mathcal{L}^{\Sigma}(S)$. We let $\mathcal{P}^{\Sigma,X}(S) = \{L \mid L \subseteq \Sigma^+ \otimes X \text{ is recognized by } S\}$ and $\mathcal{P}(S) = \bigcup_{\Sigma,X} \mathcal{P}^{\Sigma,X}(S)$. Finally we denote the set of languages with one free variable by $\mathcal{P}_1(S) = \bigcup_{\Sigma,|X|=1} \mathcal{P}^{\Sigma,X}(S)$.

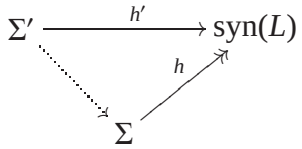
For a set of semigroups **S** be define $\mathcal{L}(\mathbf{S}), \mathcal{P}(\mathbf{S}), \mathcal{P}_1(\mathbf{S})$ equivalently.

The syntactic semigroup is defined as follows: Define the following relation $\equiv_L$ on Σ^+ by $u \equiv_L v$ iff for all words $w, w' \in \Sigma^*$ we have $wuw' \in L \iff wvw' \in L$. This relation is a congruence relation on Σ^+ called the syntactic congruence, and the syntactic semigroup $\mathrm{syn}(L)$ is $\Sigma^+ / \equiv_L$. Please note that the syntactic semigroup $\mathrm{syn}(L)$ of a language L is the smallest semigroup that recognizes L, in the sense that if S recognizes L, then $\mathrm{syn}(L) \prec S$. For any subset $M \in \Sigma^+$ be write $M/\equiv_L$ for the set of all equivalence classes of $\Sigma^+ / \equiv_L$ that contain elements of M.

2.3.1 Varieties

Given a language one might asked what are the other languages recognized by the syntactic semigroup of the first language, in other words what is the relation between L and $\mathcal{L}(\mathrm{syn}(L))$? Obtaining a different language than L by $\mathrm{syn}(L)$, we have two possibilities: change the morphism or change the accepting set.

Given any morphism $h' : \Sigma' \to \mathrm{syn}(L)$ such that $L' = h'^{-1}(A)$, we know, since h is surjective, that h' factors though h.



So L' is an inverse morphic image of L. On the other hand since the application of two morphisms is a morphism any inverse morphic image of L is recognized by $\mathrm{syn}(L)$.

If we change the accepting set things are more complicated. A shift of the accepting set, i.e. $A' = s^{-1}At^{-1}$, leads to the language $u^{-1}Lv^{-1}$, where $u \in h^{-1}(s)$, and $v \in h^{-1}(t)$ can be chosen arbitrarily. Conversely we can recognize all shifted languages of L by $\mathrm{syn}(L)$.

But for different choices of A' that are not shifts, one may need direct Boolean combinations of L to describe L'. For example let $L = (aaa)^+$, then $L' = (aaa)^+ \cup a(aaa)^+$ is recognized by the same semigroup $\mathbb{Z}_3$, and we can describe it as $L' = L \cup (aa)^{-1}L$.

We will not further immerse into this topic but it should now be obvious if one characterizes sets of languages by sets of semigroups, there are certain closure

properties necessary. These observations as in the book of Eilenberg [Eil76] led to the following concepts and results.

Definition 2.6 (Variety of Languages)**.** A *variety $\mathcal{V}$ of languages* is a set of languages that is closed under the following operations:

- $L_1, L_2 \subseteq \Sigma^+$: $L_1, L_2 \in \mathcal{V}$ implies $\Sigma^+ \setminus L_1, L_1 \cap L_2 \in \mathcal{V}$ (Boolean operations).
- $L \subseteq \Sigma^+$, $u, v \in \Sigma$: $L \in \mathcal{V}$ implies $u^{-1}Lv^{-1} \in \mathcal{V}$ (Cuts/Shifting).
- $L \subseteq \Sigma^+$, $h : \Sigma'^+ \to \Sigma^+$: $L \in \mathcal{V}$ implies $h^{-1}(L) \in \mathcal{V}$ (Inverse morphisms).

Definition 2.7 (Variety of Semigroups)**.** A *variety* $\mathbf{V}$ *of semigroups* is a set of semigroups that is closed under division and direct products.

For each variety of language $\mathcal{V}$ we can associate the smallest variety of semigroups $\mathbf{V}$ that contains all syntactic semigroups of the language variety, and we see that $\mathcal{L}(\mathbf{V}) = \mathcal{V}$.

Theorem 2.8 (Correspondence [Eil76, Theorem 3.4s])**.** *Varieties of semigroups and varieties of regular languages are in a one to one correspondence:*

- *Let $\mathcal{V}$ a variety of languages and $\mathbf{V}$ the smallest variety of semigroups that recognizes all languages in $\mathcal{V}$, then $\mathcal{L}(\mathbf{V}) = \mathcal{V}$.*
- *Let $\mathbf{V}$ be a variety of finite semigroups and $\mathbf{W}$ be the smallest variety that recognizes all languages of $\mathcal{L}(\mathbf{V})$, then $\mathbf{V} = \mathbf{W}$.*

We define some common varieties of semigroups considered later. **Fin** is the variety of all finite semigroups, **A** the variety of all aperiodic semigroups, **G** the variety of all groups. The variety **A** contains the variety **DA** of all semigroups whose $\mathcal{D}$-classes are aperiodic. The variety **G** contains the variety of all solvable groups $\mathbf{G_{solv}}$ and all Abelian groups **Ab**. The semigroups which contains only solvable subgroups are called solvable and are denoted by $\overline{\mathbf{G_{solv}}}$, the semigroups where all subgroups are Abelian are denoted by $\overline{\mathbf{Ab}}$. Further we look at the variety **DS** of semigroups, where all $\mathcal{D}$-classes are are semigroups, and **DO**, where all $\mathcal{D}$-classes are orthogonal. Also to complete the list we denote by $\mathbf{DA} \square \mathbf{G}$ the variety spanned by the block product of **DA** with **G** (please refer to the next section for the definition of the block product).

Everything of this chapter is also considered for the monoid case in [Eil76], but we restrict here to semigroups since the monoid case is equivalent.

2.3.2 Block Product

The block product was introduced in [RT89] to decompose the semigroups in their basic building blocks. We will give here a short definition of the block product and avoid to introduction notations that are not necessary for this thesis. The basic idea is to introduce an equivalent product for semigroups as the wreath product for groups.

U_1	1	0
1	1	0
0	0	0

Figure 2.1: *The semigroup U_1*

The semigroup $S^{T\times T}$ consists of all functions $f : T \times T \to S$, and the pointwise multiplication, which we usual denote by + although it might be noncommutative. On this set we have a left and right action of T: let $t_1, t_2 \in T$, then $t_1 * f$ is defined by $(t_1 * f)(m_1, m_2) = f(m_1 t_1, m_2)$, and $f * t_2$ by $(f * t_2)(m_1, m_2) = f(m_1, t_2 m_2)$ for all $m_1, m_2 \in T$.

Definition 2.9 (Block Product $\square$)**.** We define the block product $S \square T$ as the semigroup on the set $S^{T\times T} \times T$ equipped with the multiplication $(f_1, t_1)(f_2, t_2) = (f_1 * t_2 + t_1 * f_2, t_1 t_2)$.

We let $I = \{1\}$ be the smallest semigroup with the multiplication $1 \cdot 1 = 1$. The semigroup U_1 consists of two elements $0, 1$ and the product is 0 except for $1 \cdot 1 = 1$.

Theorem 2.10 (Decomposition Theorem)**.** *Every finite semigroup S divides a block product $S_1 \square (S_2 \square (\ldots (S_{k-1} \square S_k) \ldots))$, where $S_i = U_1$ or S_i is a simple group for all $i \in 1, \ldots k$. If S is a group then it suffices to use factors S_i that are simple groups, and if S is aperiodic than all factors S_i can be chosen to be U_1.*

The block product $\mathbf{V} \square \mathbf{W}$ of two varieties $\mathbf{V}$ and $\mathbf{W}$ is the smallest variety that contains all semigroups $V \square W$ where $V \in \mathbf{V}$ and $W \in \mathbf{W}$. There is a close connection between the block product of two semigroup varieties and the substitution of the corresponding classes of formulas, as states in [TT05].

Theorem 2.11 (Block Product Principle)**.** *Let Γ be a class of* (FO + MOD)$_2$[<] *sentences and Λ a class of* (FO + MOD)$_2$[<] *formulas with one free variable. If $\mathbf{V}, \mathbf{W}$ are semigroup varieties such that $\mathcal{L}(\Gamma) = \mathcal{L}(\mathbf{V})$ and $\mathcal{P}_1(\Lambda) = \mathcal{P}_1(\mathbf{W})$, then $\mathcal{L}(\Gamma \circ \Lambda) = \mathcal{L}(\mathbf{V} \square \mathbf{W})$.*

2.4 Circuits

A circuit is a directed acyclic graph, the nodes with fan-in zero are called input nodes and the other nodes are called gates. There is a distinguished gate called the output gate. Since we want to recognize a language $L_n \subseteq \Sigma^n$, we allow as input nodes either nodes labeled true or false, or nodes labeled $w_i = \sigma$, where $i = 1, \ldots, n$ and $\sigma \in \Sigma$. We call a family of r-ary binary functions for each $r \in \mathbb{N}$ a gate type. The gates of the circuit with fan-in r are labeled by r-ary binary functions from a gate type. Common examples for gate types are AND, OR or MOD$_p$.

Given a word $w \in \Sigma^n$, the truth value of the gates is computed in the obvious way, and the truth value of the circuit is that of the output gate. Given for each $n \in \mathbb{N}$ a circuit C_n that accepts a word of length n as input, where the binary functions of the gates are only chosen from a finite set of gate types, then the family $(C_n)_{n\in\mathbb{N}}$ accepts a language $L \subseteq \Sigma^+$, where $w \in L$ if $C_{|w|}$ is true for the input w.

Definition 2.12. Let Σ be an alphabet and X be a set of variables and $(C_n)_{n\in\mathbb{N}}$ be a family of circuits. We let $\mathcal{L}^\Sigma((C_n)_{n\in\mathbb{N}})= \{L \mid L \subseteq \Sigma^+$ is recognized by $(C_n)_{n\in\mathbb{N}}\}$ and $\mathcal{L}((C_n)_{n\in\mathbb{N}})= \bigcup_\Sigma \mathcal{L}^\Sigma((C_n)_{n\in\mathbb{N}})$.

For a set of circuit families $\mathbf{C}$ be define $\mathcal{L}(\mathbf{C})$ equivalently.

The depth of a circuit C_n is the length of the longest directed path in it. A family of circuits $C = (C_n)_{n\in\mathbb{N}}$ has constant depth if there is a constant d such that the depth of each circuit is bound by d, and the size of a circuit is the number of gates. In this thesis we are mainly interested in circuits of constant depth, and polynomial or linear size.

The circuit C_n in a family of circuits has in general no relation to any other circuit in the same family. In order to capture the complexity in the difference of the circuits we will introduce a uniformity language in Chapter 4, this definition is quite different from the usual definition of the uniformity languages as in [BIS90], but quite close to newer definitions that allow finer uniformity [BL06]. We will refer to these papers when we define uniformity later, so the reader can compare the notions of uniformity.

We use the normal definition for programs over finite semigroups as in [BT88, Bar89]. Later we will give a definition over typed semigroups (see Definition 4.18) that also covers the finite case. Let $\mathbf{V}$ be a class of semigroups, then we denote by $\pi - \mathbf{V}$, the set of languages recognized by the programs over semigroups of $\mathbf{V}$, also called the program variety of $\mathbf{V}$.

If we look at the previously know connections we see that there are many gaps and the connections work only for certain quantifiers (see Figures 2.2 and 2.3). But still a general connection between all three classes was missing. If we look at the bounded

Circuits	Logic	Algebra	
AC^0	FO[arb]	$\pi - \mathbf{A}$	[GL84, Imm87, BT88]
ACC^0	FO + MOD[arb]	$\pi - \overline{\mathbf{G_{solv}}}$	
NC^1	FO + G[arb]	$\pi - \mathbf{S}$	
FO[<]-uniform AC^0	FO[<]	$\mathbf{A}$	[Sch65, MP71, BL06]
FO[<]-uniform ACC^0	FO + MOD[<]	$\overline{\mathbf{G_{solv}}}$	[STT95, BL06]
	FO + MAJ[<]	$T_<$	[KLR07]
FO[<]-uniform TC^0	FO + MAJ[<, +, $*$]	$T_{<,Sq}$	[BL06, KLR07]
	FO + G[<]	$\mathbf{S}$	

Figure 2.2: *Relation between circuits and logic and algebra*

Circuits	Logic	
DLOGTIME-uniform AC^0	FO[<, +, ∗]	[BIS90]
DLOGTIME-uniform ACC^0	FO + MOD[<, +, ∗]	[BIS90]
DLOGTIME-uniform NC^1	FO + G[<, +, ∗]	[BIS90]

Figure 2.3: *Relation between circuits and logic*

variable cases there were even less results, which gave the intuition that linear circuits correspond to 2 variables in logic ([KLPT06]) again corresponding to the weak block product in algebra ([TW98, ST02, ST03]), see Figure 2.4.

It is known by Barrington [Bar89] that polynomial length programs over the group A_5 characterize the languages in NC^1. In fact any non-solvable group could be used instead of A_5. On the other hand any semigroup not containing only solvable groups, i.e. $\overline{\mathbf{G_{solv}}}$, is in ACC^0 and hence in TC^0. Therefore in order to separate the circuit complexity classes TC^0 and NC^1 by a regular language, the only possibility is to find a nonsolvable group that cannot be recognized by TC^0, where the choice of group negligible.

Circuits	Logic	Algebra	
LC^0	FO_2[arb]		
	FO_2[<]	**DA**	[TW98, ST02]
LCC^0	$(FO + MOD)_2$[arb]		[KLPT06]
	$(FO + MOD)_2$[<]	$\mathbf{DA} \square \mathbf{G_{solv}}$	[ST03]

Figure 2.4: *Relation between linear circuits, logic with two variables and algebra*

2.5 Summary

In this chapter we introduced the basic terms used in logic, algebra and circuit theory.

We recalled the definition of first order formulas with generalized quantifiers, similar to the notion of the monoidal quantifier, we allow that a quantifier has multiple subformulas. Please note that we did not restrict the definitions to the finite/regular case and allow infinite monoidal quantifiers and arbitrary predicate sets.

For a set of quantifiers and a set of predicates we denote by $\mathfrak{Q}[\mathfrak{P}]$ the class of formulas built from quantifiers of $\mathfrak{Q}$ and predicates in $\mathfrak{P}$. Further we consider subclasses where we limit the number of variables to one (or two), written as $\mathfrak{Q}_1[\mathfrak{P}]$ ($\mathfrak{Q}_2[\mathfrak{P}]$).

The usual notations from the algebraic treatment of languages, mainly from semigroup theory were given. We recapitulated the block product, and its connection to

logic by the substitution/block product principle, but we postponed a proof to the next chapter where we present a more general statement.

Given a logic, algebraic or circuit class $\mathbf{V}$ we denote by $\mathcal{L}(\mathbf{V})$ the languages over an alphabet Σ^+ recognized by the class $\mathbf{V}$, and by $\mathcal{P}(\mathbf{V})$ the pointed languages in $\Sigma^+ \otimes X$ recognized by $\mathbf{V}$. Please note that recognizing of a pointed language $L \subseteq \Sigma^+ \otimes X$ is easier than recognizing of a language $L \subseteq (\Sigma \cup 2^X)^+$.

Chapter 3

Typed Semigroups

An algebraic framework that captures the expressiveness of arbitrary classes of logic, like MAJ[<], by morphisms cannot be based on finite structures, since the classes might recognize nonregular languages. Therefore the (finitely) typed (semi)groups introduced in [KLR05] cannot be finite but are infinite (semi)groups with additional structure. Here we use an extended version, having closer connections to logic and circuit complexity.

First we give the basic definitions for typed semigroups and show that they form a category. Then we proceed to prove a theorem stating that there is a one to one correspondence between varieties of languages and varieties of typed semigroups, equivalent to the correspondence theorem of Eilenberg in the regular case. Since we intended to construct an algebraic characterization of the languages for a given class of logic, we devote our attention to the block product of typed semigroups.

3.1 Basics

We motivate the technical details of the typed semigroups by putting it in the context of languages recognition.

When we look at the syntactic semigroup S of a language L, we also have the syntactic morphism

$$\eta : \Sigma^+ \to S,$$

such that $L = \eta^{-1}(A)$ for some set $A \subseteq S$. So we have more algebraic structure than the semigroup S alone. We will define the typed semigroups such that the information about the accepting set $A = \eta(L)$ and the generators $\eta(\Sigma)$ are also represented.

Definition 3.1 (Typed Semigroup)**.** A *typed semigroup* is a triple $(S, \mathfrak{S}, \mathcal{E})$, where S is a finitely generated semigroup, $\mathfrak{S}$ a finite Boolean algebra over S and a finite set $\mathcal{E} \subseteq S$. The elements of $\mathfrak{S}$ are called *types* and the elements of $\mathcal{E}$ are called *units*. We call $(S, \mathfrak{S}, \mathcal{E})$ a *typed monoid* iff S is a monoid.

In this definition $\mathfrak{S}$ will limit the accepting sets, in order to explain this exactly we need the notion of a morphism, so we delay this for a moment.

Since we are later interested in classes closed under inverse length preserving morphisms, but not inverse morphism, we need to formalize some notion of length. In the free semigroup Σ^+, the elements $w \in \Sigma^+$ have a notion of length $|w|$, where length is the number n of generators $w_1, \ldots, w_n \in \Sigma$ such that $w_1 \ldots w_n = w$. For arbitrary semigroups there no equivalent definition of length, since there is no natural set of generators. We will use the units of typed semigroups for a reduced concept. Here we forgo to define a length for all elements of S, but define only a set of units, which should be thought as the elements of length one. Even if this seems to be dissatisfying it allows to define notions of length preserving morphisms through a requirement that units are mapped to units.

> **Finite case:** Clearly the finite semigroups embed in the typed semigroups. Given a semigroup S, we equip it with the discrete typeset and let all elements of S be units, i.e. $(S, 2^S, S)$.

Throughout this chapter we will reveal links to category theory, only to justify some of the definitions in this chapter, but not to gain results. For an introduction to category theory we refer to the book [Mac98], although no knowledge is required in this area. With the following definition of a morphism the typed semigroups form a category:

Definition 3.2 (Typed Morphism)**.** Let $(S, \mathfrak{S}, \mathcal{E})$ and $(S', \mathfrak{S}', \mathcal{E}')$ be two typed semigroups, then a morphism $h : (S, \mathfrak{S}, \mathcal{E}) \to (S', \mathfrak{S}', \mathcal{E}')$ is a triple $(h_S, h_\mathfrak{S}, h_\mathcal{E})$, such that:

- $h_S : S \to S'$ is a morphism of semigroups,
- $h_\mathfrak{S} : \mathfrak{S} \to \mathfrak{S}'$ is a morphism of Boolean algebras,
- $h_\mathcal{E} : \mathcal{E} \to \mathcal{E}'$ is a mapping of sets,
- $\forall \mathfrak{S} \in \mathfrak{S} \;\; h_S(\mathfrak{S}) = h_\mathfrak{S}(\mathfrak{S}) \cap h_S(S)$,
- $\forall u \in \mathcal{E} \;\; h_S(u) = h_\mathcal{E}(u)$.

Because of the compatibility clauses of this definition we can omit the indices of the morphism. A morphism of typed monoids is defined in the same way with the additional requirement that h_S is a morphism of monoids.

> **Finite case:** This is also consistent in the finite case, since any semigroup morphism $S \to T$ induces a typed morphism $(S, 2^S, S) \to (T, 2^T, T)$. So the finite semigroups are a subcategory of the typed semigroups.

We define when a language is recognized by a typed semigroup according to the previous considerations, compatible with the embedding of the finite case.

Definition 3.3. We say that a typed semigroup $(S, \mathfrak{S}, \mathcal{E})$ *recognizes a language* $L \subseteq \Sigma^+$ iff there is a morphism $h : \Sigma^+ \to S$ with $h(\Sigma) \subseteq \mathcal{E}$ and a set $\mathcal{S} \in \mathfrak{S}$, such that $L = h^{-1}(\mathcal{S})$. Similar $(S, \mathfrak{S}, \mathcal{E})$ *recognizes a pointed language* $L \subseteq \Sigma^+ \otimes X$ iff there is a morphism $h : (\Sigma \times X)^+ \to S$ with $h(\Sigma \times \emptyset) \subseteq \mathcal{E}$ and a set $\mathcal{S} \in \mathfrak{S}$, if for a word $w \in \Sigma^+ \otimes X$ we have $w \in L \iff h(w) \in \mathcal{S}$.

Let Σ be an alphabet and X be a set of variables and $(S, \mathfrak{S}, \mathcal{E})$ be a typed semigroup. We let $\mathcal{L}^\Sigma((S, \mathfrak{S}, \mathcal{E}))= \{L \mid L \subseteq \Sigma^+$ is recognized by $(S, \mathfrak{S}, \mathcal{E})\}$ and $\mathcal{L}((S, \mathfrak{S}, \mathcal{E}))= \bigcup_\Sigma \mathcal{L}^\Sigma((S, \mathfrak{S}, \mathcal{E}))$. We let $\mathcal{P}^{\Sigma,X}((S, \mathfrak{S}, \mathcal{E}))= \{L \mid L \subseteq \Sigma^+ \otimes X$ is recognized by $(S, \mathfrak{S}, \mathcal{E})\}$ and $\mathcal{P}((S, \mathfrak{S}, \mathcal{E}))= \bigcup_{\Sigma,X} \mathcal{P}^{\Sigma,X}((S, \mathfrak{S}, \mathcal{E}))$. Finally we denote the set of languages with one free variable by $\mathcal{P}_1((S, \mathfrak{S}, \mathcal{E}))= \bigcup_{\Sigma,|X|=1} \mathcal{P}^{\Sigma,X}((S, \mathfrak{S}, \mathcal{E}))$.

For a set of typed semigroups $\mathbf{S}$ be define $\mathcal{L}(\mathbf{S}), \mathcal{P}(\mathbf{S}), \mathcal{P}_1(\mathbf{S})$ equivalently.

The above definition uses the types to limit the accepting set, i.e. $h(L) \in \mathfrak{S}$ if the morphism h maps to the typed semigroup$(S, \mathfrak{S}, \mathcal{E})$. If we look at the semigroup $(\mathbb{F}_2, \{\emptyset, \{1\}, \mathbb{F}_2 \setminus \{1\}, \mathbb{F}_2\}, G)$, where G is a minimal generator set of $\mathbb{F}_2$. Then this contains the free group $\mathbb{F}_2$ and hence we can map Σ^+ invectively to $\mathbb{F}_2$. So if we would allow arbitrary accepting sets, then the free group $\mathbb{F}_2$ could recognize every language. Through the restrictions with the types the power is restricted pretty much to recognize the Dyck languages or inverse morphism images of them. Using a minimal generator set for the units reduces the power even to the Dyck language with 2 generators, and not arbitrary generators as without units.

In the following we will ease notation by allowing to specify $\mathfrak{S}$ only by its generators. We write for example $(\mathbb{F}_2, 1, G)$ in the example above and as another example $(\mathbb{Z}, 0, \pm 1)$ for $(\mathbb{Z}, \{\{0\}, \mathbb{Z} \setminus \{0\}, \mathbb{Z}, \emptyset\}, \pm 1)$. Please note since we chose to pick a Boolean algebra for the types, they are always closed under union, intersection and negation, hence the finest language sets that we can characterize by typed semigroups have similar closure properties.

In the following we assign to a language $L \subseteq \Sigma^+$, the typed semigroup (Σ^+, L, Σ) and show that the recognition of the language L by $(S, \mathfrak{S}, \mathcal{E})$ is equivalent to the existence of a morphism from (Σ^+, L, Σ) to $(S, \mathfrak{S}, \mathcal{E})$.

Lemma 3.4. *For any language* $L \subseteq \Sigma^+$*, the triple* (Σ^+, L, Σ) *is a typed semigroup and* L *is recognized by* $(S, \mathfrak{S}, \mathcal{E})$ *iff there is a morphism from* (Σ^+, L, Σ) *to* $(S, \mathfrak{S}, \mathcal{E})$*. Also for any language* $L \subseteq \Sigma^+ \otimes X$*, the triple* $((\Sigma \times X)^+, L, \Sigma \times \emptyset)$ *is a typed semigroup and* L *is recognized by* $(S, \mathfrak{S}, \mathcal{E})$ *iff there is a morphism from* $((\Sigma \times X)^+, L, \Sigma \times \emptyset)$ *to* $(S, \mathfrak{S}, \mathcal{E})$*.*

Proof. This follows directly from the definition of recognizability (Definition 3.3). □

For example the typed semigroup $(\mathbb{Z}, \mathbb{Z}^+, \pm 1)$ can recognize the language L where each word contains more a's than b's by $h(a) = +1$, $h(b) = -1$, $L = h^{-1}(\mathbb{Z}^+)$, but for example not the language with an equal number of a's and b's. Using a direct product $(\mathbb{Z}, \mathbb{Z}^+, \pm 1) \times (\mathbb{Z}, \mathbb{Z}^+, \pm 1)$ with $h(a) = (+1, -1)$ and $h(b) = (-1, +1)$, we can recognize the language with the same number of a's and b's with the type $\mathbb{Z}_0^- \times \mathbb{Z}_0^-$.

Please note that we only consider language in Σ^+ here, and not in Σ^*. In the finite case if a language $L \subseteq \Sigma^*$ is recognized by a monoid (or even semigroup) S, then we

B_2	a	b	ab	ba	0
a	0	ab	0	a	0
b	ba	0	b	0	0
ab	a	0	ab	0	0
ba	0	b	0	ba	0
0	0	0	0	0	0

Figure 3.1: *The semigroup B_2*

can add a neutral letter to the language without changing the syntactic morphism. On the other hand there is no morphism from Σ^* onto a semigroup, unless it is a monoid, hence the study of languages in Σ^* leads to coarser classes of languages that can be examined.

In the category of typed semigroups the situation is different: a typed morphism $h : (\Sigma^*, L, \Sigma) \to (S, \mathfrak{S}, \mathcal{E})$ can exist, but we cannot add a neutral letter to the language. This is possible, since $h(\varepsilon)$ might not be a unit, so we cannot add a letter and map it to $h(\varepsilon)$ since all letters are units and thus need to be mapped to units. Since there is not even a subtile difference in the theory of language of Σ^+ and Σ^* in our algebraic theory, we decided to state everything only for languages in Σ^+, since this is closer to the intuition from the finite case.

> **Finite case:** In the finite case the semigroup B_2 (see Figure 3.1) recognizes the language $L_{B_2} \subseteq \{a, b\}^+$ with $L_{B_2} = (ab)^+$ that does not posses a neutral letter. The same language as a subset of $\{a, b\}^*$, needs the monoid B_2^1 to be recognized, but there we have a morphism $h : \{a, b, e\} \to B_2^1$ with $h(e) = 1_{B_2}$ where e is a neutral letter. If we view this in the typed semigroup sense, we have a typed morphism $h : (\{a, b\}^+, L_{B_2}, \{a, b\}) \to (B_2, \{ab\}, \{a, b\})$, that recognizes the language. If we embed the language into $\{a, b\}^*$ we get the modified morphism $h(\{a, b\}^*, L_{B_2}, \{a, b\}) \to (B_2^1, \{ab\}, \{a, b\})$. Here we cannot simply add the letter e to the alphabet and map it to $1 \in B_2^1$ since 1 is not a unit.

We proceed now with the definitions known for (finite) semigroups adopted to typed semigroups, with a special attention to division, that will lead to the an extension of a well known result in the regular case, that the syntactic semigroup is the smallest semigroup under division recognizing the language.

Definition 3.5 (Injective, Surjective, Bijective Typed Morphisms,Typed Subsemigroup and Division)**.** Let $(S, \mathfrak{S}, \mathcal{E}), (S', \mathfrak{S}', \mathcal{E}')$ be two typed semigroups.

- A morphism $h : (S, \mathfrak{S}, \mathcal{E}) \to (S', \mathfrak{S}', \mathcal{E}')$ with $h = (h_S, h_\mathfrak{S}, h_\mathcal{E})$ is *injective/surjective/bijective* iff $h_S, h_\mathfrak{S}$, and $h_\mathcal{E}$ are injective/surjective/bijective.

- $(S', \mathfrak{S}', \mathcal{E}')$ is a *typed subsemigroup* of $(S, \mathfrak{S}, \mathcal{E})$, denoted by the order symbol $(S', \mathfrak{S}', \mathcal{E}') \leq (S, \mathfrak{S}, \mathcal{E})$, iff S' is a subset of S and there is an injective morphism from $(S', \mathfrak{S}', \mathcal{E}')$ to $(S, \mathfrak{S}, \mathcal{E})$.
- A typed semigroup $(S', \mathfrak{S}', \mathcal{E}')$ *divides* a typed semigroup $(S, \mathfrak{S}, \mathcal{E})$ denotes by $(S', \mathfrak{S}', \mathcal{E}') \preceq (S, \mathfrak{S}, \mathcal{E})$ iff $(S', \mathfrak{S}', \mathcal{E}')$ is the morphic image of a typed subsemigroup of $(S, \mathfrak{S}, \mathcal{E})$.

We will show that the definitions are sound with the abstract categorical definitions.

Lemma 3.6. *Let* $(S, \mathfrak{S}, \mathcal{E}), (S', \mathfrak{S}', \mathcal{E}'), (T, \mathfrak{T}, \mathcal{F})$ *be typed semigroups.*

1. *For every typed group the identity mapping is a bijective morphism.*
2. *The application of two (injective/surjective/bijective) is again a (injective,surjective,bijective) morphism.*
3. $\alpha : (S', \mathfrak{S}', \mathcal{E}') \to (T, \mathfrak{T}, \mathcal{F})$ *is an injective morphism iff* $\beta = \beta' \iff \alpha\beta = \alpha\beta'$ *for all morphisms* $\beta, \beta' : (S, \mathfrak{S}, \mathcal{E}) \to (S', \mathfrak{S}', \mathcal{E}')$.
4. $\alpha : (T, \mathfrak{T}, \mathcal{F}) \to (S, \mathfrak{S}, \mathcal{E})$ *is a surjective morphism iff* $\beta = \beta' \iff \beta\alpha = \beta'\alpha$ *for all morphisms* $\beta, \beta' : (S, \mathfrak{S}, \mathcal{E}) \to (S', \mathfrak{S}', \mathcal{E}')$.
5. $\alpha : (S, \mathfrak{S}, \mathcal{E}) \to (S', \mathfrak{S}', \mathcal{E}')$ *is a bijective morphism iff there is a morphism* $\alpha' : (S', \mathfrak{S}', \mathcal{E}') \to (S, \mathfrak{S}, \mathcal{E})$ *with* $\alpha\alpha' = \mathbf{1}_{(S, \mathfrak{S}, \mathcal{E})}$ *and* $\alpha'\alpha = \mathbf{1}_{(S', \mathfrak{S}', \mathcal{E}')}$.

Proof.

1. This is clear.
2. This is also clear.
3. Let α be an injective morphism, and $\beta \neq \beta'$, then there is a $s \in S$ with $\beta(s) \neq \beta'(s)$ and by injectivity of α, we have $\alpha\beta(s) \neq \alpha\beta'(s)$. Assume α is not injective, then there are $s \neq s' \in (S, \mathfrak{S}, \mathcal{E})$ with $\alpha(s) = \alpha(s')$. We let $(S, \mathfrak{S}, \mathcal{E})$ be the subsemigroup of $(S', \mathfrak{S}', \mathcal{E}')$ generated by (s, s'), we define two morphism $\beta((s, s')) = s$ and $\beta'((s, s')) = s'$. Then $\alpha\beta = \alpha\beta'$ but $\beta \neq \beta'$. Note that we can choose types and units in such a way that the morphisms are typed morphisms.
4. This is similar to the previous case.
5. Let $\alpha = (\alpha_S, \alpha_{\mathfrak{S}}, \alpha_{\mathcal{E}})$ be a bijective morphism, then we need to show that $(\alpha_S^{-1}, \alpha_{\mathfrak{S}}^{-1}, \alpha_{\mathcal{E}}^{-1})$ is also a typed morphism. But since α is surjective for every type $\mathcal{S}' \in \mathfrak{S}'$, there is a type $\mathcal{S} \in \mathfrak{S}$ with $\alpha_S(\mathcal{S}) = \alpha_{\mathfrak{S}}(\mathcal{S}) = \mathcal{S}'$ and by injectivity we have $\alpha_S^{-1}(\mathcal{S}') = \alpha_{\mathfrak{S}}^{-1}(\mathcal{S}')$. For the same reason $\alpha_{\mathcal{E}}^{-1}$ coincides with α_S^{-1}.

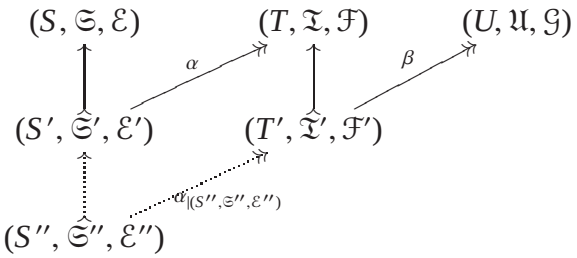
   The reverse direction follows from (3) and (4).

□

The notion of division allows us to compare two semigroups in the finite case. we show that we can also compare typed semigroups.

Lemma 3.7. *Division is a preorder on the class of typed semigroups.*

Proof. It is clear that division is reflexive, hence we need to show that it is transitive, i.e. for three typed semigroups $(S, \mathfrak{S}, \mathcal{E})$,$(T, \mathfrak{T}, \mathcal{F})$,$(U, \mathfrak{U}, \mathcal{G})$: if $(U, \mathfrak{U}, \mathcal{G})$ divides $(T, \mathfrak{T}, \mathcal{F})$ and $(T, \mathfrak{T}, \mathcal{F})$ divides $(S, \mathfrak{S}, \mathcal{E})$, then $(U, \mathfrak{U}, \mathcal{G})$ divides $(S, \mathfrak{S}, \mathcal{E})$. We are in the following situation:

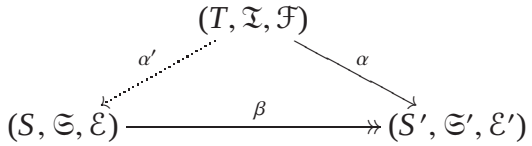


In the diagram above we let $(S'', \mathfrak{S}'', \mathcal{E}'') = \alpha^{-1}((T', \mathfrak{T}', \mathcal{F}'))$, then $\beta\alpha$ maps $(S'', \mathfrak{S}'', \mathcal{E}'')$ subjectively on $(U, \mathfrak{U}, \mathcal{G})$ and is a subgroup of $(S, \mathfrak{S}, \mathcal{E})$, hence $(U, \mathfrak{U}, \mathcal{G})$ divides $(S, \mathfrak{S}, \mathcal{E})$. □

> **Finite case:** In the finite case division is of course a partial order since if S divides T, then $|S| \leq |T|$, and if $|S| = |T|$ division implies isomorphism.

But please note that division in general is not a partial order, as in the finite semigroup case. For example $(\mathbb{F}_2, 1_{\mathbb{F}_2}, \emptyset)$ divides $(\mathbb{F}_3, 1_{\mathbb{F}_2}, \emptyset)$ and $(\mathbb{F}_3, 1_{\mathbb{F}_3}, \emptyset)$ divides $(\mathbb{F}_2, 1_{\mathbb{F}_2}, \emptyset)$, since as monoids $\mathbb{F}_2$ is a submonoid of $\mathbb{F}_3$ and also $\mathbb{F}_3$ is a submonoid of $\mathbb{F}_2$, by the same morphisms they are typed submonoids, but they are not isomorphic.

Lemma 3.8. *Let $(S, \mathfrak{S}, \mathcal{E}), (S', \mathfrak{S}', \mathcal{E}')$ be typed semigroups. If there is a surjective morphism $\beta : (S, \mathfrak{S}, \mathcal{E}) \to (S', \mathfrak{S}', \mathcal{E}')$, then every morphism from the free semigroup $(T, \mathfrak{T}, \mathcal{F})$ to $(S', \mathfrak{S}', \mathcal{E}')$ factors though β. That is for every morphism α, there is a morphism α' such that the following diagram commutes.*



Proof. For every generator t of $(T, \mathfrak{T}, \mathcal{F})$, we define $\alpha'(t) = s_t$, where s_t is any preimage of $\alpha(t)$ under β. Since $(T, \mathfrak{T}, \mathcal{F})$ is free, we can extend this to a morphism and by definition $\alpha(t) = \beta(\alpha'(t))$, hence $\alpha = \beta \circ \alpha'$. □

Two typed semigroups that divides each other do not need to be isomorphic but they recognize the same languages, which we will prove with the help of the typed syntactic semigroup in the next lemma.

Definition 3.9 (Typed Syntactic Semigroup)**.** Let $L \subseteq \Sigma^+$ be a language, then $(\Sigma^+/\sim_L, L/\sim_L, \Sigma/\sim_L)$ is the *typed syntactic semigroup* of L.

For example let $L \subseteq \{a,b\}^+$ be the language with the same number of a's and b's. Then $w \sim_L w'$ iff $\#_a(w) - \#_b(w) = \#_a(w') - \#_b(w')$. We have $\Sigma^+/\sim_L$ is the set of equivalence classes that is isomorphic to the integers by $\eta : \Sigma^+ \to \mathbb{Z}$ by $w \mapsto \#_a(w) - \#_b(w)$. This implies that $L/\sim_L$ is just one equivalence class and $\eta(L) = 0$, and the equivalence classes that contain Σ are mapped by η to $+1$ and -1. Hence we get $\mathrm{syn}(L) = (\mathbb{Z}, \{0\}, \{1,-1\})$.

It is easy to see that the units are always a generator set of a typed syntactic semigroup, i.e. iff $(S, \mathfrak{S}, \mathcal{E})$ is a typed syntactic semigroup then $S = \mathcal{E}^+$. We will continue to show that the typed syntactic semigroup is the unique minimal semigroup that recognizes L.

> **Finite case:** Since division is a partial order in the finite case it is easy to show that the syntactic semigroup is the minimal semigroup by division that recognize the languages.

Lemma 3.10. *Let* $(S, \mathfrak{S}, \mathcal{E})$ *be a typed semigroup, then if* $\mathrm{syn}(L)$ *divides* $(S, \mathfrak{S}, \mathcal{E})$*, then* $(S, \mathfrak{S}, \mathcal{E})$ *recognizes* L*.*

Proof. By definition there is a subsemigroup $(S', \mathfrak{S}', \mathcal{E}')$ of $(S, \mathfrak{S}, \mathcal{E})$, such that there is a surjective morphism α from $(S', \mathfrak{S}', \mathcal{E}')$ to $\mathrm{syn}(L)$. By definition of the syntactic semigroup, there is a morphism $\eta : (\Sigma^+, L, \Sigma) \to \mathrm{syn}(L)$, then by Lemma 3.8 there is a morphism $\alpha' : (\Sigma^+, L, \Sigma) \to (S', \mathfrak{S}', \mathcal{E}')$, with $\alpha \circ \alpha' = \eta$. So $(S', \mathfrak{S}', \mathcal{E}')$ recognizes L and so $(S, \mathfrak{S}, \mathcal{E})$. □

Although we do not have a partial order the syntactic semigroup of L is the unique minimal typed semigroup up to isomorphism that recognizes L.

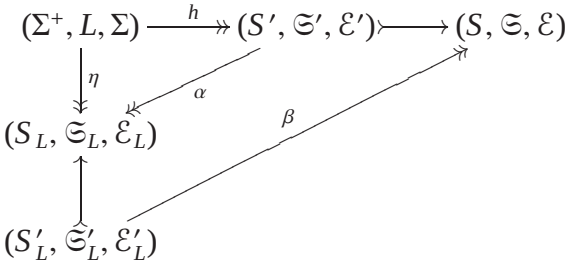
Lemma 3.11. *The typed syntactic semigroup of a language* L *is the minimal semigroup under division that recognizes* L*.*

Proof. Let L be a language recognized by a typed semigroup $(S, \mathfrak{S}, \mathcal{E})$ with a morphism h, and let $(S', \mathfrak{S}', \mathcal{E}')$ be the image of (Σ^+, L, Σ). We show there is a surjective morphism α from $(S', \mathfrak{S}', \mathcal{E}')$ to the typed syntactic semigroup $(S_L, \mathfrak{S}_L, \mathcal{E}_L)$, and hence $(S_L, \mathfrak{S}_L, \mathcal{E}_L)$ divides $(S, \mathfrak{S}, \mathcal{E})$.

$$\begin{array}{ccccc} (\Sigma^+, L, \Sigma) & \xrightarrow{h} & (S', \mathfrak{S}', \mathcal{E}') & \rightarrowtail & (S, \mathfrak{S}, \mathcal{E}) \\ \downarrow \eta & \swarrow \alpha & & & \\ (S_L, \mathfrak{S}_L, \mathcal{E}_L) & & & & \end{array}$$

So we need to show that $\alpha(s) = \eta(h^{-1}(s))$ is well defined. But assume that there are w_1, w_2 with $h(w_1) = h(w_2)$ and $\eta(w_1) \neq \eta(w_2)$, then there are $u, v \in \Sigma^+$ with $uw_1v \in L \iff uw_2v \notin L$, but $h(uw_1v) = h(uw_2v)$ hence L is not recognized by S.

In order to show that is $(S_L, \mathfrak{S}_L, \mathcal{E}_L)$ is the unique minimal typed semigroup we assume $(S, \mathfrak{S}, \mathcal{E})$ divides $(S_L, \mathfrak{S}_L, \mathcal{E}_L)$. Then there is a submonoid $(S'_L, \mathfrak{S}'_L, \mathcal{E}'_L)$ of $(S_L, \mathfrak{S}_L, \mathcal{E}_L)$.



Since Σ generates Σ^+ and h is surjective, we know that $|\mathcal{E}'|, |\mathcal{E}_L| < \infty$, $\mathcal{E}'$ generates S', also S_L is generated by $\mathcal{E}_L$. Also we know $|\mathcal{E}'| \leq |\mathcal{E}| \leq |\mathcal{E}'_L| \leq |\mathcal{E}_L| \leq |\mathcal{E}'|$, so $|\mathcal{E}'| = |\mathcal{E}| = |\mathcal{E}'_L| = |\mathcal{E}_L|$. We get $(S_L, \mathfrak{S}_L, \mathcal{E}_L) \cong (S'_L, \mathfrak{S}'_L, \mathcal{E}'_L)$, and β maps S'_L to the span of $\mathcal{E}$, hence $\mathcal{E}$ generates S, which again implies $(S, \mathfrak{S}, \mathcal{E}) \cong (S', \mathfrak{S}', \mathcal{E}')$.

Now we have a surjective morphism from $(S, \mathfrak{S}, \mathcal{E})$ to $(S_L, \mathfrak{S}_L, \mathcal{E}_L)$ and converse, but this does not imply that there is an isomorphism. But it is clear that $\alpha\beta$ is a permutation of $\mathcal{E}$, hence there is a power of $\alpha\beta$ that is the identity on $\mathcal{E}$. But then this power is also an identity on $(S, \mathfrak{S}, \mathcal{E})$ and we have $(S_L, \mathfrak{S}_L, \mathcal{E}_L) \cong (S, \mathfrak{S}, \mathcal{E})$. □

Contrary to the finite case bilateral division does not imply isomorphy between two typed semigroups, but we can prove a weaker result:

Lemma 3.12. *If $(S, \mathfrak{S}, \mathcal{E})$ and $(T, \mathfrak{T}, \mathcal{F})$ are two typed semigroups that divide each other, then $\mathcal{P}((S, \mathfrak{S}, \mathcal{E})) = \mathcal{P}((T, \mathfrak{T}, \mathcal{F}))$.*

Proof. The lemma is equivalent to statement: the syntactic semigroup of language divides $(S, \mathfrak{S}, \mathcal{E})$ iff it divides $(T, \mathfrak{T}, \mathcal{F})$. This is true since division is a preorder. □

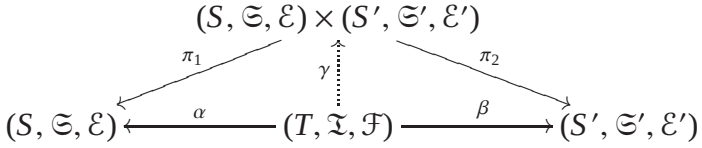
> **Finite case:** The previous lemma is again trivial for finite semigroups since division is a partial order there.

Finally we define the direct product as the last basic operation.

Definition 3.13 (Direct Product)**.** The *direct product* of two semigroups $(S, \mathfrak{S}, \mathcal{E})$, $(S', \mathfrak{S}', \mathcal{E}')$, denoted by $(S, \mathfrak{S}, \mathcal{E}) \times (S', \mathfrak{S}', \mathcal{E}')$, is defined as $(S \times S', \mathfrak{S} \times \mathfrak{S}', \mathcal{E} \times \mathcal{E}')$.

The direct product in the abstract categorical sense is defined as in the following lemma, where we show that the two definitions coincide.

Lemma 3.14. *The direct product of two semigroups $(S, \mathfrak{S}, \mathcal{E})$ and $(S', \mathfrak{S}', \mathcal{E}')$, is the smallest typed semigroup such that for every typed semigroup $(T, \mathfrak{T}, \mathcal{F})$ and all morphisms α, β, there is a morphism γ such that the following diagram commutes.*



Proof. It is clear that for the direct product we can let $\gamma = \alpha \times \beta$ and the diagram commutes. Assume that $(S, \mathfrak{S}, \mathcal{E}) \times (S', \mathfrak{S}', \mathcal{E}')$ does not divide the direct product $(U, \mathfrak{U}, \mathcal{G})$, then we let $(T, \mathfrak{T}, \mathcal{F}) = (S, \mathfrak{S}, \mathcal{E}) \times (S', \mathfrak{S}', \mathcal{E}')$, and $\alpha = \pi_1, \beta = \pi_2$. So the morphism γ cannot be injective, hence there are two elements $(s_1, s_1'), (s_2, s_2')$ that map to the same element in $(U, \mathfrak{U}, \mathcal{G})$. But then both elements are mapped to the same elements under $\pi_1\gamma$ and $\pi_2\gamma$, this implies $s_1 = s_2$ and $s_1' = s_2'$, and hence $(s_1, s_1') = (s_2, s_2')$ a contradiction $\lightning$. □

Finally we show how the direct product can be used to disassemble the Boolean algebra of a typed semigroups:

Lemma 3.15. *Let* $(S, \mathfrak{S}, \mathcal{E})$ *and* $(S, \mathfrak{S}', \mathcal{E})$ *be two typed semigroups, then* $(S, \mathfrak{S} \vee \mathfrak{S}', \mathcal{E})$ *divides* $(S, \mathfrak{S}, \mathcal{E}) \times (S, \mathfrak{S}', \mathcal{E})$.

Proof. We define $h : (S, \mathfrak{S} \times \mathfrak{S}', \mathcal{E}) \to (S, \mathfrak{S}, \mathcal{E}) \times (S, \mathfrak{S}', \mathcal{E})$, with $h(s) = (s, s)$, note that (s, s) is a unit. Then for $\mathcal{S} \in \mathfrak{S}$, $h(\mathcal{S}) = \mathcal{S} \times S \cap h(S)$, and for $\mathcal{S}' \in \mathfrak{S}'$, $h(\mathcal{S}') = S \times \mathcal{S}' \cap h(S)$, hence h is a typed morphism, and is clearly injective. □

3.2 Weakly closed classes

We will now impose certain closure properties on typed semigroups similar as [Eil76], to get a one-to-one relation to varieties of languages. As an intermediate step we show a weaker relation for classes and weakly closed classes of languages and typed semigroups.

It is possible that the Boolean algebra chosen is to coarse to make use of the semigroup. For example if we look at a typed semigroup with the trivial Boolean algebra $(S, \{S, \emptyset\})$, then we can recognize only the trivial languages independent of S. But still this typed semigroup is not the trivial typed semigroup $(\mathbb{I}, \mathbb{I}, \mathbb{I})$ unless S is the trivial semigroup $\mathbb{I}$, so we introduce a concept that avoids this problem.

Definition 3.16 (Reduced Semigroup/Trivial Extension)**.** Let $(S, \mathfrak{S}, \mathcal{E})$ be a typed semigroup such that there is a congruence $\sim$ of S and $\mathfrak{S}$ is coarser than $\sim$. Then we call $(\widetilde{S, \mathfrak{S}, \mathcal{E}}) = (S/\sim, \mathfrak{S}/\sim, \mathcal{E}/\sim)$ the *reduced semigroup*, and $(S, \mathfrak{S}, \mathcal{E})$ a *trivial extension* of $(S/\sim, \mathfrak{S}/\sim, \mathcal{E}/\sim)$.

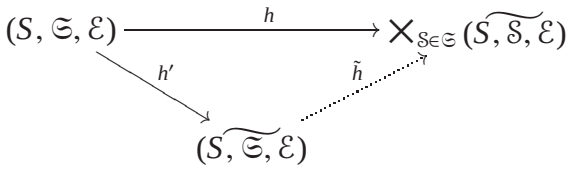
> **Finite case:** There is no counterpart for the concept above in finite semigroups, since every semigroup except the trivial one, allows us to recognize non-trivial languages. But in the case of typed semigroups any semigroup S with a trivial typeset, i.e. only $\emptyset, S$, can only recognize trivial languages.

Some examples of trivial extensions are: Let G be any group and N be a normal subgroup, then (G, N) is a trivial extension of $(G/N, 1)$. Another example is $(\mathbb{Z}, 2\mathbb{Z})$ which is a trivial extension of $(\mathbb{Z}_2, 0)$. Also for each language L, the typed semigroup (Σ^+, L, Σ) is a trivial extension of the syntactic semigroup of L.

Lemma 3.17. *If* $(\widetilde{S, \mathfrak{S}, \mathcal{E}}) = (S', \mathfrak{S}', \mathcal{E}')$*, then* $\mathcal{P}((S, \mathfrak{S}, \mathcal{E})) = \mathcal{P}((S', \mathfrak{S}', \mathcal{E}'))$*. Also* $(\widetilde{S, \mathfrak{S}, \mathcal{E}})$ *divides* $\bigtimes_{\mathcal{S}\in\mathfrak{S}} (\widetilde{S, \mathcal{S}, \mathcal{E}})$*.*

Proof. Let $L \subseteq \Sigma^+$ be a language in $\mathcal{P}((S, \mathfrak{S}, \mathcal{E}))$, then there is a typed morphism $h : (\Sigma \times 2^X)^+ \to (S, \mathfrak{S}, \mathcal{E})$ such that $h(\Sigma \times \emptyset) \subseteq \mathcal{E}$ and $L = h^{-1}(\mathcal{S})$ for an $\mathcal{S} \in \mathfrak{S}$. We know there is a morphism $h' : (S, \mathfrak{S}, \mathcal{E}) \to (S', \mathfrak{S}', \mathcal{E}')$ hence $h'h$ is a morphism that recognizes L with the type $h'(\mathcal{S})$. It follows $\mathcal{P}((S, \mathfrak{S}, \mathcal{E})) = \mathcal{P}((S', \mathfrak{S}', \mathcal{E}'))$, since the other direction is trivial.

Again we know there is a morphism $h'_{\mathcal{S}} : (S, \mathcal{S}, \mathcal{E}) \to (\widetilde{S, \mathcal{S}, \mathcal{E}})$, and a morphism $h' : (S, \mathfrak{S}, \mathcal{E}) \to (\widetilde{S, \mathfrak{S}, \mathcal{E}})$. We define a typed morphism $h : (S, \mathfrak{S}, \mathcal{E}) \to \bigtimes_{\mathcal{S}\in\mathfrak{S}} (\widetilde{S, \mathcal{S}, \mathcal{E}})$, by $h(s) = ((h_{\mathcal{S}}(s))_{\mathcal{S}\in\mathfrak{S}})$.



We need to show that h factors through h', so we have a morphism $\tilde{h}$. Assume that $h'(s_1) = h'(s_2)$, then for all elements $l, r \in S$ and all types $\mathcal{S} \in \mathfrak{S}$ we have $ls_1r \in \mathcal{S} \iff ls_2r \in \mathcal{S}$. But if for a fixed type $\mathcal{S}$ we have for all $l, r \in S$ that $ls_1r \in \mathcal{S} \iff ls_2r \in \mathcal{S}$, then $h'_{\mathcal{S}}(s_1) = h'_{\mathcal{S}}(s_2)$. It follows that $h'(s_1) = h'(s_2)$ implies that $h(s_1) = h(s_2)$, and so there is a morphism $\tilde{h}$. □

Lemma 3.18. *A typed semigroup is the syntactic semigroup of a language iff it is reduced, generated by its units and has 4 or 2 types.*

Proof. For the one direction it is clear that by minimality of the syntactic semigroup it is reduced, generated by its units and has 4 types, or if it is trivial 2 types.

Assume a reduced semigroup has 2 types, then it is the trivial semigroup and hence the syntactic semigroup of Σ^+ and $\emptyset$ iff it has a unit.

Assume a reduced semigroup $(S, \mathfrak{S}, \mathcal{E})$ has 4 types and is generated by its units. We let $h : \mathcal{E}^+ \to S$ be the natural morphism, and $L = h^{-1}(\mathcal{S})$ where $\mathcal{S} \in \mathfrak{S}$ is a nontrivial type. Then $\mathrm{syn}(L) = (S, \mathfrak{S}, \mathcal{E})$, by the definition of the syntactic semigroup. □

> **Finite case:** While it is hard to decide if a finite semigroup is the syntactic semigroup of language, it is easy here since a typed syntactic semigroup can have only a minimal typeset and for a fixed type it is easy to decide if a semigroup is a syntactic semigroup. This compares to the problem in the finite case: given a semigroup and an accepting set, is the accepting set is disjunctive.

Definition 3.19 ((Weakly Closed) Class)**.** A *class* of typed semigroups is a non-empty set of typed semigroups, that is closed under division and trivial extensions. A class of typed semigroups that is closed under direct products is called a *weakly closed class*. If S is a set of typed semigroups then $\mathrm{wc}(S)$ is the smallest weakly closed class containing S.

Lemma 3.20. *Let* $\mathbf{V}$ *be a typed semigroup class, then* $(\Sigma^+, L, \Sigma) \in \mathbf{V}$ *iff* $\mathrm{syn}(L) \in \mathbf{V}$.

Proof. This is clear since (Σ^+, L, Σ) is a trivial extension of $\mathrm{syn}(L)$ by definition and $\mathrm{syn}(L)$ divides (Σ^+, L, Σ) by Lemma 3.11. □

Definition 3.21 ((Weakly Closed) Class of Languages)**.** Let $\mathbf{V}$ be a (weakly closed) class of semigroups, then we call $\mathcal{V} = \mathcal{L}(\mathbf{V})$ a *(weakly closed) class of languages*.

We begin to show that the languages recognized by a (weakly closed) class of typed semigroups have certain closure properties.

Proposition 3.22. *If* $\mathbf{V}$ *is a (weakly closed) class of typed semigroups, then* $\mathcal{V} = \mathcal{L}(\mathbf{V})$ *is closed under inverse length preserving morphisms (and Boolean combinations).*

Proof. If $L_2 \in \mathcal{V}$ then there is a morphism $\eta : \Sigma_2^+ \to \mathrm{syn}(L_2)$, that recognizes L_2. Assume there is a length preserving morphism $h : \Sigma_1^+ \to \Sigma_2^+$ with $h(L_1) = L_2$, then ηh recognizes L_1.

Also for a weakly closed class, if $L_1, L_2 \in \mathcal{V}$ then $\mathrm{syn}(L_1), \mathrm{syn}(L_2) \in \mathbf{V}$, and hence $\mathrm{syn}(L_1) \times \mathrm{syn}(L_2) \in \mathbf{V}$ which can recognize $L_1 \cap L_2$ and $L_2 \cup L_2$. □

> **Finite case:** The previous proposition has no correspondence in the finite semigroup world; since the accepting set is ignored, the languages are always closed under quotients. Finite semigroups do not keep track of the units, hence to characterize a closure under inverse length preserving morphisms but not under inverse morphisms is not possible, and the weakly closed class of languages recognized by a weakly closed class of semigroups is already a variety. This makes it even impossible to characterize the weakly closed class of languages recognized for example by FO[<, mod], which we can describe by finite typed semigroups.

Proposition 3.23. *For two classes* $\mathbf{V} \subseteq \mathbf{W}$ *and* $\mathcal{V} = \mathcal{L}(\mathbf{V}), \mathcal{W} = \mathcal{L}(\mathbf{W})$ *we have* $\mathcal{V} \subseteq \mathcal{W}$.

Proof. This is clear since for every $L \in \mathcal{V}$ we have $\mathrm{syn}(L) \in \mathbf{V} \subseteq \mathbf{W}$ so $L \in \mathcal{W}$. □

The next proposition is extremely important since it ensures that every (weakly closed) class of languages can be characterized by a (weakly closed) class of typed semigroups.

Proposition 3.24. *If* $\mathcal{V}$ *is a set of languages closed under inverse length preserving morphisms (and Boolean combination), then there is a (weakly closed) class of typed semigroups* $\mathbf{V}$ *with* $\mathcal{L}(\mathbf{V}) = \mathcal{V}$.

Proof. We only prove this proposition for weakly closed classes of languages, the other proof is equivalent. Let $\mathbf{V}$ be the smallest weakly closed class that contains all syntactic semigroups of $\mathcal{V}$. Then by definition $\mathcal{V}$ is a subset of the corresponding language variety to $\mathbf{V}$.

Assume L has a syntactic semigroup $(S, \mathfrak{S}, \mathcal{E}) = \mathit{syn}(L)$ in $\mathbf{V}$. Then $(S, \mathfrak{S}, \mathcal{E})$ divides $\bigtimes_i (S_i, \mathfrak{S}_i, \mathcal{E}_i)$, where $(S_i, \mathfrak{S}_i, \mathcal{E}_i) = \mathrm{syn}(L_i)$ for some language $L_i \in \mathcal{V}$. So the semigroup $\bigtimes_i (S_i, \mathfrak{S}_i, \mathcal{E}_i)$ recognizes L, i.e. there is a morphism $h : \Sigma^+ \to \bigtimes_i (S_i, \mathfrak{S}_i, \mathcal{E}_i)$. Also since the syntactic morphism of L_i are surjective we can choose a tuple $(w_i)_i$ of words with $w_i \in \Sigma_i^+$ such that $h(s) = \eta((w_i)_i)$. But then we have a morphism $h' : \Sigma^+ \to \bigtimes \Sigma_i^+$ such that hh' recognizes L, and since $\pi_i h'(L) = L_i$ the language L is the inverse image of $\bigcup_i L_i$ and so $L \in \mathcal{V}$. □

3.3 Varieties

While the one-to-one relation fails for weakly closed classes, we now proceed to define the notion of a variety to get the one-to-one correspondence. We need more than closure under direct product and division, like for the definition of a variety of regular languages, we have additional closure properties.

Let $A \subseteq S$ be a subset of the semigroup S, then for $\lambda, \varrho \in S$, we let $\lambda^{-1} A \varrho^{-1} = \{s \mid \lambda s \varrho \in A\}$. This definition is consistent so far that iff λ and ϱ have inverse elements, then this notion coincides with the normal definition, i.e. $\lambda^{-1} A \varrho - 1 = \{\lambda^{-1} s \varrho^{-1} \mid s \in A\}$.

Definition 3.25 (Shifting)**.** Let $(S, \mathfrak{S}, \mathcal{E})$ be a typed semigroup than $(S, \mathfrak{S}', \mathcal{E})$ is a *shift* of $(S, \mathfrak{S}, \mathcal{E})$, iff there are $\lambda, \varrho \in S$ with $\mathfrak{S}' = \{\lambda^{-1} \mathcal{S} \varrho^{-1} \mid \mathcal{S} \in \mathfrak{S}\}$.

The units help to preserve the length in the mappings, which is not desired in some cases hence we define a way to "ignore" the units. We needed units to characterize (weakly closed) classes of languages algebraically; since varieties of languages are closed under inverse morphisms, the units are not needed in the algebraic characterization of varieties of languages.

Definition 3.26 (Unit Relaxation)**.** Let $(S, \mathfrak{S}, \mathcal{E})$ be a typed semigroup then $(S, \mathfrak{S}, \mathcal{E}')$ for any finite set $\mathcal{E}' \subseteq S$ is a *unit relaxation* of $(S, \mathfrak{S}, \mathcal{E})$.

Definition 3.27 (Variety)**.** A *variety* of weakly closed class of typed semigroups that is closed under shifting and unit relaxation.

In the later chapters we will mostly use varieties, hence our typed semigroups are closed under unit relaxation. Because of this we write $(S, \mathfrak{S})$ for a typed semigroup $(S, \mathfrak{S}, \mathcal{E})$ where the unit set is an arbitrary finite subset of S.

> **Finite case:** A variety of finite semigroups is automatically closed under shifting and unit relaxation; since we embed a finite semigroup S into the typed semigroups as $(S, 2^S, S)$, the semigroup S already posses all possible types and units.

Proposition 3.28. *If* $\mathbf{V}$ *is a variety of typed semigroups, then* $\mathcal{V} = \mathcal{L}(\mathbf{V})$ *is a variety of languages.*

Proof. If $L_1, L_2 \in \mathcal{V}$ then $\mathrm{syn}(L_1), \mathrm{syn}(L_2) \in \mathbf{V}$, and hence $\mathrm{syn}(L_1) \times \mathrm{syn}(L_2) \in \mathbf{V}$ which can recognize $L_1 \cap L_2$ and $L_2 \cup L_2$. Also if $L_2 \in \mathcal{V}$ then there is a morphism $\eta : \Sigma_2^+ \to \mathrm{syn}(L_2)$, that recognizes L_2. Assume there is a (length preserving) morphism $h : \Sigma_1^+ \to \Sigma_2^+$ with $h(L_1) = L_2$, then ηh recognizes L_1. In the case that h is not length preserving we need to replace $\mathrm{syn}(L_2)$ by an appropriate unit relaxation, otherwise we would map unit elements to non-unit elements. Finally if $L \in \mathcal{V}$ then so are $\sigma^{-1}L$ and $L\sigma^{-1}$ since the types of the semigroup variety is closed under shifting. □

Proposition 3.29. *For two varieties* $\mathbf{V} \subseteq \mathbf{W}$ *we have* $\mathcal{V} \subseteq \mathcal{W}$*, where equality occurs only if* $\mathbf{V} = \mathbf{W}$.

Proof. It is clear that $\mathcal{V} \subseteq \mathcal{W}$, so we need to show that equivalence statement for varieties.

Let $(S, \mathfrak{S}, \mathcal{E})$ be a semigroup in $\mathbf{W}$ and assume $\mathcal{V} = \mathcal{W}$. We let $\{\mathcal{S}_i\}_i = \mathfrak{S}$, then $(S, \mathcal{S}_i)$ is a typed semigroup for each type $\mathcal{S}_i \in \mathfrak{S}$. It follows $(S, \mathfrak{S}, \mathcal{E})$ divides $\bigtimes_i (S, \mathcal{S}_i)$ and also $(\widetilde{S, \mathfrak{S}, \mathcal{E}})$ divides $\bigtimes_i (\widetilde{S, \mathcal{S}_i})$, but for each $(\widetilde{S, \mathcal{S}_i})$ there is language in $\mathcal{W}$ with $\mathrm{syn}(L_i) = (\widetilde{S, \mathcal{S}_i})$. The same languages are in $\mathcal{V}$ and hence $(\widetilde{S, \mathcal{S}_i})$ is also in $\mathbf{V}$, and by the closure property also $(\widetilde{S, \mathfrak{S}, \mathcal{E}})$ and with a trivial extension also $(S, \mathfrak{S}, \mathcal{E})$ and hence $\mathbf{V} = \mathbf{W}$. □

Proposition 3.30. *For every variety of languages* $\mathcal{V}$ *there is a corresponding variety of typed semigroups* $\mathbf{V}$*, such that* $\mathcal{V} = \mathcal{L}(\mathbf{V})$.

Proof. Let $\mathbf{V}$ be the smallest variety that contains all syntactic semigroups of $\mathcal{V}$. Then by definition $\mathcal{V}$ is a subset of the corresponding language variety to $\mathbf{V}$.

Assume L has a syntactic semigroup $(S, \mathfrak{S}, \mathcal{E}) = \mathit{syn}(L)$ in $\mathbf{V}$, we need to show that $L \in \mathcal{V}$. Then $(S, \mathfrak{S}, \mathcal{E})$ divides a trivial extension of $\bigtimes_i (S_i, \mathfrak{S}_i)$, where $(S_i, \mathfrak{S}_i, \mathcal{E}_i) = \mathrm{syn}(L_i)$ for some language $L_i \in \mathcal{V}$. So the semigroup $\bigtimes_i (S_i, \mathfrak{S}_i)$ recognizes L, i.e. there is a morphism $h : \Sigma^+ \to \bigtimes_i (S_i, \mathfrak{S}_i)$. Also since the syntactic morphism of L_i are surjective we can choose a tuple $(w_i)_i$ of words with $w_i \in \Sigma_i^+$ such that $h(s) = \eta((w_i)_i)$. But then we have a morphism $h' : \Sigma^+ \to \bigtimes \Sigma_i^+$ such that hh' recognizes L, and since $\pi_i h'(L) = L_i$ the language L is the inverse image of $\bigcup_i L_i$ and so $L \in \mathcal{V}$. □

Now we can state an extension of the Eilenberg Theorem to arbitrary varieties of languages. So we get the same result as in the finite case in both directions.

Theorem 3.31. *Varieties of typed semigroups and varieties of languages are in a one to one correspondence:*

- *Let* $\mathcal{V}$ *a variety of languages and* $\mathbf{V}$ *the smallest variety of typed semigroups that recognizes all languages in* $\mathcal{V}$*, then* $\mathcal{L}(\mathbf{V}) = \mathcal{V}$.
- *Let* $\mathbf{V}$ *be a variety of typed semigroups and* $\mathbf{W}$ *be the smallest variety that recognizes all languages of* $\mathcal{L}(\mathbf{V})$*, then* $\mathbf{V} = \mathbf{W}$.

Proof. By Proposition 3.28 the correspondence of a semigroup variety $\mathbf{V}$ and the languages $\mathcal{L}(\mathbf{V})$ is a relation between varieties. Then by Proposition 3.29 this mapping from the semigroup varieties to the language varieties is injective, and by Proposition 3.30 also surjective. □

Since we are most of the time only interested in an algebraic characterization of some language class, we are mainly interested in the first condition of the previous theorem, that we have an algebraic characterization of a variety of languages.

> **Finite case:** Please note that the Eilenberg Theorem in the finite case follows directly from this theorem, since we showed that the finite semigroup can be embedded in the typed semigroups.

3.4 Block product

In the regular case the block product was defined with much freedom, here we need to make some restrictions, as already pointed out in [KLR05].

The block product is used to model algebraically the nesting of quantifiers in logical formulas. The substitution as defined in the preliminaries allows us to decompose formulas. We will use this decomposition in the reverse way to build complex typed semigroups from simple ones. First we will define an algebraic equivalent of the substitution, and show that we can compute this substitution by the block product as defined below.

We define transductions for typed semigroups, compatible with the definition for finite semigroups as in [TW04].

Definition 3.32 ($(S, \mathfrak{S}, \mathcal{E})$-Transduction)**.** Let $h : ((\Sigma \times X)^+, L, \Sigma \times \emptyset) \to (S, \mathfrak{S}, \mathcal{E})$ be a morphism and $C \subseteq S$ a finite set and Σ' an alphabet. A $(S, \mathfrak{S}, \mathcal{E})$-*transduction* is a map $\tau : \Sigma^+ \otimes X \to (\Sigma')^+ \otimes X$, where

$$\tau((w_1 \dots w_n)_{x_1=i_1,\dots,x_k=i_k}) = (w'_1 \dots w'_n)_{x_1=i_1,\dots,x_k=i_k}.$$

In this map the w'_i depend on w_i and on the type of $h(w_1 \dots w_{i-1} c w_{i+1} \dots w_n)$ for all $c \in C$, i.e. there is a map $\Sigma \times \mathfrak{S}^C \to \Sigma'$. We let τ_h be the corresponding morphism h to the transduction τ.

> **Finite case:** In the finite there exists also the notion of a S-transduction. There it is convenient to use $\Sigma' = S \times \Sigma \times S$ and let $w'_i = (h(w_{<i}), w_i, h(w_{>i}))$. But in our case this is not possible since S is not finite and the construction would be too powerful.

We will now define the block product of $(S, \mathfrak{S}, \mathcal{E})$ with $(S', \mathfrak{S}', \mathcal{E}')$ such that this captures the languages recognized by $(S, \mathfrak{S}, \mathcal{E})$ after applying a $(S', \mathfrak{S}', \mathcal{E}')$ transduction. We need more restrictions in the definition than in the finite case (see Chapter 2.3.2), otherwise the block product would be too powerful.

Definition 3.33 (Block Product $\boxdot$)**.** Let $(S, \mathfrak{S}, \mathcal{E})$, $(S', \mathfrak{S}', \mathcal{E}')$ be two typed semigroups and $C \subseteq S'$ be a finite set, then the *block product* $(S, \mathfrak{S}, \mathcal{E}) \boxdot_C (S', \mathfrak{S}', \mathcal{E}') = (T, \mathfrak{T}, \mathcal{F})$ is a typed semigroup, where T is the finitely generated subsemigroup of $S \square S'$, generated by the elements (f, s), where $f : S'^1 \times S'^1 \to \mathcal{E}$ and $s \in \mathcal{E} \cup C$. We require that $f(b_1, b_2) = f(b_1', b_2')$ if for all $c \in C$ and all $\mathcal{S}' \in \mathfrak{S}'$ we have

$$b_1 c b_2 \in \mathcal{S}' \iff b_1' c b_2' \in \mathcal{S}'.$$

The types $\mathfrak{T}$ are $\{(f, s) \mid f(e, e) \in \mathcal{S}\}$ for all $\mathcal{S} \in \mathfrak{S}$. The units $\mathcal{F}$ are the elements (f, s), where f is a generator functions, hence f maps only to units of $(S, \mathfrak{S}, \mathcal{E})$, and s is a unit.

Note that $\mathcal{E}$, C and $\mathfrak{S}'$ are finite and hence there are only finitely many elements (f, s) generating the subsemigroup. We cannot simply let C be $(S', \mathfrak{S}', \mathcal{E}')$ since the block product is not finitely generated. But since any finite set C will do we will skip to specify C in the future and assume the correct C is picked.

Remark 3.34. The definition of the block product is chosen in such a way that if two elements have the same first component, they are in the same type. We could have defined this similar to the direct product and have allowed Boolean combinations of types on the first and second component, but this would be further apart from the logic. Algebraically this is only a tiny difference since one can simulate the second component by a direct product, i.e. $(A \boxdot B) \times B$ captures this power.

> **Finite case:** In the finite case we can allow all functions in the block product and need no restriction, but in the infinite case we $\mathbb{Z} \square \mathbb{Z}$ already contains the free semigroup with two generators. So we could get an injective mapping of the alphabet into $\mathbb{Z} \square \mathbb{Z}$ and if we have no types recognize all languages. Even with types, if we allow all functions $(\mathbb{Z}, 0, \pm 1) \square (\mathbb{Z}, 0, \pm 1)$ contains the free semigroup with two generators and $(\mathbb{Z}, 0, \pm 1) \square ((\mathbb{Z}, 0, \pm 1) \square (\mathbb{Z}, 0, \pm 1))$ could recognize all languages. The reason is that if we allow all functions, we could choose the characteristic function of the language in the left block product.

Now we show how the type of a word is computed in a block product. Let $(T, \mathfrak{T}, \mathcal{F}) = (S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}')$ be a typed semigroup with a type $\mathcal{T} \in \mathfrak{T}$, then an element t is in $\mathcal{T}$ iff $\pi_1(t)(e, e)$ is in some type $\mathcal{S}$ of $(S, \mathfrak{S}, \mathcal{E})$. We call this type $\pi_1 \mathcal{T}$ that corresponds to $\mathcal{T}$, i.e. $s \in \mathcal{T} \iff \pi_1(s)(e, e) \in \pi_1 \mathcal{T}$.

Lemma 3.35. *Let* $h : \Sigma^+ \to (T, \mathfrak{T}, \mathcal{F}) = (S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}')$*, then* $h(w) \in \mathcal{T} \in \mathfrak{T}$ *iff*

$$\sum_{i=1}^{n} \pi_1(h(w))(\prod_{j<i} \pi_2(h(w)), \prod_{j>i} \pi_2(h(w))) \in \pi_1 \mathcal{T} \in \mathfrak{S}$$

Proof. We do this by computation. $h(w) \in \mathcal{T}$ iff $\pi_1(h(w)) \in \pi_1\mathcal{T}$. We let $h(w_i) = (g_i, z_i)$ where $g_i : S' \times S' \to S$, and $z_i \in S'$. We compute

$$\begin{aligned}
h(w) &= (f_1, g_1)(f_2, g_2)\dots(f_n, g_n) = \\
&= (f_1g_2 + g_1f_2)(f_3, g_3)\dots(f_n, g_n) = \\
&= (f_1g_2g_3 + g_1f_2g_3 + g_1g_2f_3)(f_4, g_4)\dots(f_n, g_n) = \\
&= \Big(f_1(\prod_{j=2}^{n} g_j) + \dots + (\prod_{j<i} g_j) f_i (\prod_{j>i} g_i) + \dots + (\prod_{j=1}^{n-1} g_j) f_n, \prod_{i=1}^{n} g_i\Big) = \\
&= \Big(\sum_{i=1}^{n} (\prod_{j<i} g_j) f_i (\prod_{j>i} g_i), \prod_{i=1}^{n} g_i\Big)
\end{aligned}$$

Since $(g_l f g_r)(e, e) = f(g_l, g_r)$ the result follows. □

We list some facts about block products of typed semigroups, that were similar listed in [TW04] for the case of finite monoids.

Lemma 3.36. *Let* $(S, \mathfrak{S}, \mathcal{E})$, $(S', \mathfrak{S}', \mathcal{E}')$, $(T, \mathfrak{T}, \mathcal{F})$, $(T', \mathfrak{T}', \mathcal{F}')$ $(U, \mathfrak{U}, \mathcal{G})$ *be typed semigroups.*

1. *If* $(S, \mathfrak{S}, \mathcal{E})$ *and* $(S', \mathfrak{S}', \mathcal{E}')$ *are typed monoids, then* $(S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}')$ *is a typed monoid.*
2. *If* $(S, \mathfrak{S}, \mathcal{E})$ *and* $(S', \mathfrak{S}', \mathcal{E}')$ *are typed groups, then* $(S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}')$ *is a typed group.*
3. $((S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}'))^k \preceq (S, \mathfrak{S}, \mathcal{E})^k \boxdot (S', \mathfrak{S}', \mathcal{E}')^k$.
4. *If* $(S, \mathfrak{S}, \mathcal{E})$ *is a typed monoid and divides* $(T, \mathfrak{T}, \mathcal{F})$, *and* $(S', \mathfrak{S}', \mathcal{E}') \preceq (T', \mathfrak{T}', \mathcal{F}')$, *then* $(S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}') \preceq (T, \mathfrak{T}, \mathcal{F}) \boxdot (T', \mathfrak{T}', \mathcal{F}')$.
5. *Let 1 denote the trivial typed monoid. Then* $(S, \mathfrak{S}, \mathcal{E}) \boxdot 1 \cong (S, \mathfrak{S}, \mathcal{E})$, *and* $1 \boxdot (S, \mathfrak{S}, \mathcal{E})$ *is a trivial extension of 1.*
6. $(S, \mathfrak{S}, \mathcal{E}) \preceq (S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}')$.
7. $((S, \mathfrak{S}, \mathcal{E}) \boxdot (T, \mathfrak{T}, \mathcal{F})) \boxdot (U, \mathfrak{U}, \mathcal{G}) \preceq (S, \mathfrak{S}, \mathcal{E}) \boxdot (((T, \mathfrak{T}, \mathcal{F}) \boxdot (U, \mathfrak{U}, \mathcal{G})) \times (U, \mathfrak{U}, \mathcal{G}))$.

Proof. 1. $(\mathbf{e_S}, e_{S'})$ is the one element of $(S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}')$.

2. Let $(f, s') \in (S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}')$, then

$$\begin{aligned}
(s'^{-1} f^{-1} s'^{-1}, s'^{-1})(f, s') &= (s'^{-1} f^{-1} s'^{-1} s' + s'^{-1} f, s'^{-1} s') = \\
&= (s'^{-1} f^{-1} + s'^{-1} f, e_{S'}) = \\
&= (s'^{-1}(f^{-1} + f), e_{S'}) = \\
&= (s'^{-1} \mathbf{e_S}, e_{S'}) = \\
&= (\mathbf{e_S}, \mathbf{e_{S'}})
\end{aligned}$$

and also $(f, s')(s'^{-1} f^{-1} s'^{-1}, s'^{-1}) = (\mathbf{e_S}, e_{S'})$.

3. Let $((f_i, s_i'))_{i\in\{1,\dots,k\}} \in ((S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}'))^k$, where $f_i : (S', \mathfrak{S}', \mathcal{E}') \times (S', \mathfrak{S}', \mathcal{E}') \to (S, \mathfrak{S}, \mathcal{E})$ for $i \in \{1, \dots, k\}$. We let $\tilde{f_i} : ((S', \mathfrak{S}', \mathcal{E}')^k \times (S', \mathfrak{S}', \mathcal{E}')^k) \to (S, \mathfrak{S}, \mathcal{E})$ by $\tilde{f_i}(t_1, t_2) = f_i(\pi_i(t_1), \pi_i(t_2))$. Then $((\tilde{f_i})_i, (s_i')_i) \in (S, \mathfrak{S}, \mathcal{E})^k \boxdot (S', \mathfrak{S}', \mathcal{E}')^k$, and the map $((f_i, s_i'))_i \mapsto ((\tilde{f_i})_i, (s_i')_i)$ is an injective morphism.

4. If $S' < S$ and $T' < T$ then $(f', t') \mapsto (f, t')$ where f is the extension on $T \times T$ by the neutral element of S is an injective morphism. Hence we can assume that that S' is a morphic image of S and T' of T.

   In the special case $T = T'$, we define a map (f, t) maps to (f', t), where $f'(t_1, t_2)$ is the morphic image of $f(t_1, t_2)$. It is clear that this is a surjective morphism.

   Now we assume that $T \neq T'$, then we take the subsemigroup V of $T \times T \to S$, where the functions are constant on the congruence classes of the map $T \to T'$. It is clear that there is a morphism from (f, t) for $f \in V$ to (f', t') since V is has constant functions on a congruence.

5. The morphism $s \mapsto (\mathbf{s}, e)$ and its inverse are bijections, since $\mathbf{s}$ is a function $1 \times 1 \to (S, \mathfrak{S}, \mathcal{E}) \cong (S, \mathfrak{S}, \mathcal{E})$. Also the typed semigroup $1 \boxdot (S, \mathfrak{S}, \mathcal{E})$ has only one type, hence is a trivial extension of 1.

6. The morphism $s \mapsto (\mathbf{s}, e)$ is injective.

7. We show $((S, \mathfrak{S}, \mathcal{E}) \boxdot (T, \mathfrak{T}, \mathcal{F})) \boxdot (U, \mathfrak{U}, \mathcal{G})$ is a typed subsemigroup of $(S, \mathfrak{S}, \mathcal{E}) \boxdot (((T, \mathfrak{T}, \mathcal{F}) \boxdot (U, \mathfrak{U}, \mathcal{G})) \times (U, \mathfrak{U}, \mathcal{G}))$. We need the extra typed semigroup $(U, \mathfrak{U}, \mathcal{G})$ only to get the correct types, not for the computation. Hence we will first show that $(S \square T) \square U$ is a subsemigroup of $S \square (T \square U)$ and even that this map works for the restricted block product version.

   First we describe the elements in the semigroup of $(S \square T) \square U$, by a triple $((f, g), z)$, where $f : U^2 \to (T^2 \to S)$, $g : U^2 \to T$ and $u \in U$. Since the tuple (f, g) describes a map $U^2 \to S \square U$ it is clear that the triple describes exactly the elements of $(S \square T) \square U$.

   Now the elements in the strong block product are in $S \square (T \square U)$ also by triples $(\hat{f}, (\hat{g}, \hat{z}))$, where $\hat{f} : (T \square U)^2 \to S$, $\hat{g} : U^2 \to T$ and $\hat{z} \in U$. Here $(\hat{g}, \hat{z})$ describes an elements of $T \square U$. Already by the functions of these two triples one gets an idea how to map the last to components, but for the first this is a bit tricky.

   Let $((f, g), z) \in (S \square T) \square U$. We define $\hat{f_z} : (T \square U)^2 \to S$ by

$$\hat{f_z}((g_1, z_1), (g_2, z_2)) = f(z_1, z_2)(g_1(e, zz_2), g_2(z_1 z, e)).$$

   Then we have a map $\alpha : (S \square T) \square U \to S \square (T \square U)$ by $((f, g), z) \mapsto (\hat{f_z}, (g, z))$. We need to show this map is a morphism. Before this we define the notion of a left and right action $*$ of $g : U^2 \to T$ on $f : U^2 \to (T^2 \to S)$, where g acts pointwise on f, by the left and right action of T on $T^2 \to S$, i.e.

$(f * g)(z_1, z_2)(b_1, b_2) = (f(z_1, z_2)g(z_1, z_2))(b_1, b_2) = f(z_1, z_2)(b_1, g(z_1, z_2)b_2)$ and $(g * f)(z_1, z_2)(b_1, b_2) = (g(z_1, z_2)f(z_1, z_2))(b_1, b_2) = f(z_1, z_2)(b_1 g(z_1, z_2), b_2)$. So we compute the products:

$((f, g), z)((f', g'), z') = ((fz') * (zg') + (gz') * (zf'), gz' + zg', zz')$.

$(\hat{f_z}, (g, z))(\hat{f_{z'}}, (g', z')) = ((\hat{f_z})(g', z') + (g, z)(\hat{f'_{z'}}), (gz' + zg', zz'))$.

If we show $(\hat{f_z})(g', z') = (\widehat{(fz') * (zg')})_{zz'}$ we are done by symmetry.

We compute

$$\begin{aligned}
&(\hat{f_z})(g', z')((g_1, z_1), (g_2, z_2)) \\
&\quad = (\hat{f_z})((g_1, z_1), (g', z')(g_2, z_2)) \\
&\quad = (\hat{f_z})((g_1, z_1), (g'z_2 + z'g_2, z'z_2)) \\
&\quad = f(z_1, z'z_2)(g_1(e, zz'z_2), (g'z_2 + z'g_2)(z_1 z, e)) \\
&\quad = f(z_1, z'z_2)(g_1(e, zz'z_2), g'(z_1 z, z_2) + g_2(z_1 zz', e))
\end{aligned}$$

We compute

$$\begin{aligned}
&(\widehat{(fz') * (zg')})_{zz'}((g_1, z_1), (g_2, z_2)) \\
&\quad = ((fz') * (zg'))(z_1, z_2)(g_1(e, zz'z_2), g_2(z_1 zz', e)) \\
&\quad = f(z_1, z'z_2)(g_1(e, zz'z_2), (zg')(z_1, z_2) + g_2(z_1 zz', e)) \\
&\quad = f(z_1, z'z_2)(g_1(e, zz'z_2), g'(z_1 z, z_2) + g_2(z_1 zz', e))
\end{aligned}$$

Hence the mapping is a morphism, and it is clearly injective.

Finally we need to show that this maps generating functions of the restricted block product to generator functions. So let f be a generator functions. We have $\hat{f_z}((g_1, z_1), (g_2, z_2)) = v$ iff $f(z_1, z_2)(g_1(e, zz_2), g_2(z_1 z, e)) = v$, which depends on a finite collection of clauses of the form: $z_1 c z_2 \in \mathfrak{U}$ since this is a projection of generator function of the outer block product of $(S \square T) \square U$ and of clauses of the form $(g_1(e, zz_2)bg_2(z_1 z, e)) \in \mathfrak{T}$ since f maps to generator functions of $S \square T$. The last condition can be rewritten to $(g_1, z_1)(\mathbf{b}, z)(g_2, z_2)$ in a type of $T \square U$. But $\hat{f_z}$ is not a generator function since we cannot test the first type of conditions. Hence we need the extra $(U, \mathfrak{U}, \mathcal{G})$, the $((f, g), z)$ is mapped to $(\hat{f_z}, ((g, z), z))$, and now we can define $\hat{f_z}$ equivalently as above and here it is a generator function.

Please not that since the types are only defined on the first component this map gives rise to an embedding of $((S, \mathfrak{S}, \mathcal{E}) \boxdot (T, \mathfrak{T}, \mathcal{F})) \boxdot (U, \mathfrak{U}, \mathcal{G})$ into $(S, \mathfrak{S}, \mathcal{E}) \boxdot ((T, \mathfrak{T}, \mathcal{F}) \boxdot (U, \mathfrak{U}, \mathcal{G})) \times (U, \mathfrak{U}, \mathcal{G})$.

□

The block product of two weakly closed classes is the weakly closed class spanned by the block product of their elements. We get the follow facts:

Lemma 3.37. *Let* $\mathbf{U}, \mathbf{V}, \mathbf{W}$ *be weakly closed classes of typed semigroups.*

- *If* $\mathbf{V}, \mathbf{W} \subseteq \mathbf{G}$*, then* $\mathbf{V} \boxdot \mathbf{W} \subseteq \mathbf{G}$.
- $\mathbf{V}^k \boxdot \mathbf{W}^k = (\mathbf{V} \boxdot \mathbf{W})^k$.
- $(\mathbf{U} \boxdot \mathbf{V}) \boxdot \mathbf{W} \subseteq \mathbf{U} \boxdot ((\mathbf{V} \boxdot \mathbf{W}) \times \mathbf{W})$.

Proof. All these results are directly obtained from the previous lemma. □

Lemma 3.38. *Let* $(S, \mathfrak{S}, \mathcal{E}), (S', \mathfrak{S}', \mathcal{E}')$ *be two typed semigroups, then for every language* L*:* $L \in \mathcal{P}((S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}'))$ *iff there is an* $(S', \mathfrak{S}', \mathcal{E}')$*-transduction* τ *such that* $\tau(L) \in \mathcal{P}((S, \mathfrak{S}, \mathcal{E}))$*.*

Proof. Let L be recognized by h into $(T, \mathfrak{T}, \mathcal{F}) = (S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}')$. Then by Lemma 3.35, we have $w \in L$ iff

$$\sum_{i=1}^{n} \pi_1(h(w))(\prod_{j<i} \pi_2(h(w)), \prod_{j>i} \pi_2(h(w))) \in \pi_1\mathcal{T}.$$

We can rewrite this as $\sum_{i=1}^{n} v_i \in \pi_1\mathcal{T}$, where $v_i \in (S, \mathfrak{S}, \mathcal{E})$, and

$$v_i = \pi_1(h(w))(\prod_{j<i} \pi_2(h(w)), \prod_{j>i} \pi_2(h(w))).$$

But since the set V of all possible v_i is finite we let $\Sigma' = \Sigma \times V$, and the map $w_1 \dots w_n \mapsto (w_1, v_1) \dots (w_n, v_n)$ is a $(S', \mathfrak{S}', \mathcal{E}')$-transduction. Also the product of the v_i is computed in $(S, \mathfrak{S}, \mathcal{E})$, hence $\tau(L) \in \mathcal{P}((S, \mathfrak{S}, \mathcal{E}))$.

If $\tau(L) \in \mathcal{P}((S, \mathfrak{S}, \mathcal{E}))$ where h_τ is the morphism corresponding to τ, then there is a morphism h' to $(S, \mathfrak{S}, \mathcal{E})$ that recognizes $\tau(L)$. Also for $w_1, w_2 \in \Sigma^*$, $\sigma \in \Sigma$, we let $\tau(w_1, \sigma, w_2) = \tau(w_1\sigma w_2)_{|w_1|+1}$. We let $f_\sigma(t_1, t_2) = h'(\tau(h_\tau^{-1}(t_1), \sigma, h_\tau^{-1}(t_2)))$, by definition of τ this is well defined. Now h to $(S, \mathfrak{S}, \mathcal{E}) \boxdot (S', \mathfrak{S}', \mathcal{E}')$ by $\sigma \mapsto (f, h_\tau(\sigma))$ recognizes L. This can be seen by a small computation: Since

$$\sum_{i=1}^{n} f(\prod_{j<i} h_\tau(w), \prod_{j>i} h_\tau(w)) \in \pi_1\mathcal{T}$$

is equal to

$$\sum_{i=1}^{n} h'(\tau(w)_i) \in \pi_1\mathcal{T}$$

by definition of h. □

In the following lemma we will show that for Γ-substitution the is an equivalent $\mathbf{V}$-transduction if $\mathcal{P}(\Gamma) = \mathcal{P}(\mathbf{V})$. We will not have the same map in both cases since the adjoint map of the Γ-substitution maps to a different alphabet than the $\mathbf{V}$-transduction.

Lemma 3.39. *Let Γ be a set of formulas and $\mathbf{V}$ be a weakly closed class of typed semigroups, with $\mathcal{P}(\Gamma) = \mathcal{P}(\mathbf{V})$. Then for any Γ-substitution θ there is a $\mathbf{V}$-transduction τ such that $\vartheta = h\tau$, where h is a length preserving morphism of the resulting alphabets. Also for any $\mathbf{V}$-transduction τ there is a Γ-substitution θ such that $\tau = h\vartheta$, where h is a length preserving morphism of the resulting alphabets.*

Proof. Let σ be an Γ-substitution then there are formulas $\Phi = \{\varphi_1, \ldots, \varphi_k\} \subset \Gamma$ such that σ is a Φ substitution. Since $\mathcal{P}(\Gamma) = \mathcal{P}(\mathbf{V})$ for each φ_i there is a typed semigroup $(S_i, \mathfrak{S}_i, \mathcal{E}_i)$ in $\mathbf{V}$ such that L_{φ_i} is recognized by h_{φ_i} into $(S_i, \mathfrak{S}_i, \mathcal{E}_i)$ by the type $\mathcal{S}_i$. We define a $h = \bigtimes_{i=1}^{k} h_{\varphi_i}$ into $\bigtimes_{i=1}^{k}(S_i, \mathfrak{S}_i, \mathcal{E}_i)$ transduction τ with the constants $C = (h((\sigma, x)))_{\sigma\in\Sigma}$.

Then $\vartheta(w_1 \ldots w_n) = w'_1 \ldots w'_n$ and $w'_i \in 2^\Phi$ is the set

$$\begin{aligned}\{\varphi_i \mid w_{x=i} \models \varphi_i\} &= \{\varphi_i \mid w_{x=i} \in L_{\varphi_i}\} = \\ &= \{\varphi_i \mid h_{\varphi_i}(w) \in \mathcal{S}\} = \\ &= \{\varphi_i \mid \pi_i(h(w)) \in \mathcal{S}\} = \\ &= \{\varphi_i \mid h(w) \in \mathcal{S}_i\} = \\ &= \{\varphi_i \mid \tau(w)_i \subseteq \mathcal{S}_i\}\end{aligned}$$

We let $h : \mathcal{S}^+ \to 2^\Phi$ by $h(\mathcal{S}) = \{\varphi_i \mid \mathcal{S} \subseteq \mathcal{S}_i\}$, then $\vartheta = h\tau$.

The reverse direction is proven equivalent. □

Now we come to the main theorem that allows us to construct typed semigroup classes with the block product that correspond to logic classes.

Theorem 3.40 (Block Product Principle)**.** *Let Γ, Λ be two classes of formulas and $\mathbf{V}, \mathbf{W}$ be two classes of semigroups, such that $\mathcal{P}(\Gamma) = \mathcal{P}(\mathbf{V})$ and $\mathcal{P}(\Lambda) = \mathcal{P}(\mathbf{W})$, then $\mathcal{P}(\Gamma \circ \Lambda) = \mathcal{P}(\mathbf{V} \boxdot \mathbf{W})$. Also if $\mathcal{L}(\Gamma) = \mathcal{L}(\mathbf{V})$ and $\mathcal{P}_1(\Lambda) = \mathcal{P}_1(\mathbf{W})$, then $\mathcal{L}(\Gamma \circ \Lambda) = \mathcal{L}(\mathbf{V} \boxdot \mathbf{W})$.*

Proof. By Lemma 2.3 we know $L \in \mathcal{P}(\Gamma \circ \Lambda)$ iff there is a Λ-substitution θ and $\vartheta(L) \in \mathcal{P}(\Gamma) = \mathcal{P}(\mathbf{V})$. But by Lemma 3.39 this is equivalent to there is a $\mathbf{W}$-transduction τ with $\tau(L) = \vartheta(L)$. And by Lemma 3.38 this is equivalent to $L \in \mathcal{P}(\mathbf{V} \boxdot \mathbf{W})$. □

Definition 3.41 (sbpc,wbpc)**.** We write sbpc($\mathbf{V}$) for the smallest weakly closed class closed under block products from the left, and wbpc($\mathbf{V}$) for the smallest weakly closed class closed under block product from the right.

Proposition 3.42. *If $\mathbf{V}$ is a weakly closed class of semigroups, then*

$$\text{sbpc}(\mathbf{V}) \supseteq \text{wbpc}(\mathbf{V}).$$

Proof. This is a consequence of Lemma 3.36 (7) and (3). □

Since we often consider the $<$ predicate we introduce a new version of the block product that is closer to the block product in the finite case. Later we will see that in the present of the order predicate in the logic class this block product yields a simpler algebraic equivalence.

Definition 3.43 (Block Product $\boxtimes$)**.** Let $(S, \mathfrak{S}, \mathcal{E}), (S', \mathfrak{S}', \mathcal{E}')$ be Let $C \subseteq S'$ be a finite set, then the *block product* $(S, \mathfrak{S}, \mathcal{E}) \boxtimes {}_C(S', \mathfrak{S}', \mathcal{E}')= (T, \mathfrak{T}, \mathcal{F})$ is a typed semigroup, where T is the finitely generated subsemigroup of $(S, \mathfrak{S}, \mathcal{E}) \square S'$, generated by the elements (f, s), where $f : S'^1 \times S'^1 \to \mathcal{E}$ and $s \in \mathcal{E} \cup C$. We require that $f(b_1, b_2) = f(b_1', b_2')$ if for all $c \in C$ and all $\mathcal{S}' \in \mathfrak{S}'$ we have

$$b_1 c b_2 \in \mathcal{S}' \iff b_1' c b_2' \in \mathcal{S}' \wedge b_1 c \in \mathcal{S}' \iff b_1' c \in \mathcal{S}' \wedge c b_2 \in \mathcal{S} \iff c b_2' \in \mathcal{S}.$$

The types $\mathfrak{T}$ are $\{(f, s) \mid f(e, e) \in \mathcal{S}\}$ for all $\mathcal{S} \in \mathfrak{S}$. The units $\mathcal{F}$ are the elements (f, s), where f is a generator functions, hence f maps only to units of $(S, \mathfrak{S}, \mathcal{E})$, and s is a unit.

> **Finite case:** This block product is actually closer to the finite case. Even in the finite case this gives the problem that the block product can be used only for logic classes that contain the order predicate, otherwise it is already there to strong. The advantage in the finite case is that with this definition all functions are possible, and hence the algebra is easier.

The difference in the definition of $\boxdot$ and $\boxtimes$ is that we allow more generator functions. The following lemmas have essentially the same proofs as above.

Lemma 3.44. *Let* $(S, \mathfrak{S}, \mathcal{E})$, $(S', \mathfrak{S}', \mathcal{E}')$, $(T, \mathfrak{T}, \mathcal{F})$, $(T', \mathfrak{T}', \mathcal{F}')$ $(U, \mathfrak{U}, \mathcal{G})$ *be typed semigroups.*

1. *If* $(S, \mathfrak{S}, \mathcal{E})$ *and* $(S', \mathfrak{S}', \mathcal{E}')$ *are typed monoids, then* $(S, \mathfrak{S}, \mathcal{E}) \boxtimes (S', \mathfrak{S}', \mathcal{E}')$ *is a typed monoid.*

2. *If* $(S, \mathfrak{S}, \mathcal{E})$ *and* $(S', \mathfrak{S}', \mathcal{E}')$ *are typed groups, then* $(S, \mathfrak{S}, \mathcal{E}) \boxtimes (S', \mathfrak{S}', \mathcal{E}')$ *is a typed group.*

3. $((S, \mathfrak{S}, \mathcal{E}) \boxtimes (S', \mathfrak{S}', \mathcal{E}'))^k \preceq (S, \mathfrak{S}, \mathcal{E})^k \boxtimes (S', \mathfrak{S}', \mathcal{E}')^k$.

4. *If* $(S, \mathfrak{S}, \mathcal{E})$ *is a typed monoid and divides* $(T, \mathfrak{T}, \mathcal{F})$*, and* $(S', \mathfrak{S}', \mathcal{E}') \preceq (T', \mathfrak{T}', \mathcal{F}')$*, then* $(S, \mathfrak{S}, \mathcal{E}) \boxtimes (S', \mathfrak{S}', \mathcal{E}') \preceq (T, \mathfrak{T}, \mathcal{F}) \boxtimes (T', \mathfrak{T}', \mathcal{F}')$.

5. *Let 1 denote the trivial typed monoid. Then* $(S, \mathfrak{S}, \mathcal{E}) \boxtimes 1 \cong (S, \mathfrak{S}, \mathcal{E})$*, and* $1 \boxtimes (S, \mathfrak{S}, \mathcal{E})$ *is a trivial extension of 1.*

6. $(S, \mathfrak{S}, \mathcal{E}) \preceq (S, \mathfrak{S}, \mathcal{E}) \boxtimes (S', \mathfrak{S}', \mathcal{E}')$.

7. $((S, \mathfrak{S}, \mathcal{E}) \boxtimes (T, \mathfrak{T}, \mathcal{F})) \boxtimes (U, \mathfrak{U}, \mathcal{G}) \preceq (S, \mathfrak{S}, \mathcal{E}) \boxtimes (((T, \mathfrak{T}, \mathcal{F}) \boxtimes (U, \mathfrak{U}, \mathcal{G})) \times (U, \mathfrak{U}, \mathcal{G}))$.

Proof. The proof of this lemma is essentially the same as Lemma 3.36. □

Lemma 3.45. *Let* $\mathbf{U}, \mathbf{V}, \mathbf{W}$ *be weakly closed classes of typed semigroups.*

- *If* $\mathbf{V}, \mathbf{W} \subseteq \mathbf{G}$*, then* $\mathbf{V} \boxtimes \mathbf{W} \subseteq \mathbf{G}$.
- $\mathbf{V}^k \boxtimes \mathbf{W}^k = (\mathbf{V} \boxtimes \mathbf{W})^k$.
- $(\mathbf{U} \boxtimes \mathbf{V}) \boxtimes \mathbf{W} \subseteq \mathbf{U} \boxtimes ((\mathbf{V} \boxtimes \mathbf{W}) \times \mathbf{W})$.

Proof. This follows again directly from the previous lemma. □

Definition 3.46 ($\mathrm{sbpc}_<$,$\mathrm{wbpc}_<$)**.** We write $\mathrm{sbpc}_<(\mathbf{V})$ for the smallest weakly closed class closed under block products from the left, and $\mathrm{wbpc}_<(\mathbf{V})$ for the smallest weakly closed class closed under block product from the right.

Proposition 3.47. *If* $\mathbf{V}$ *is a weakly closed class of semigroups, then*

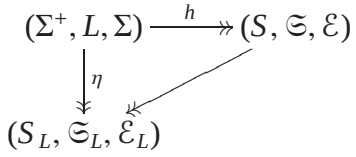
$$\mathrm{sbpc}_<(\mathbf{V}) \supseteq \mathrm{wbpc}_<(\mathbf{V}).$$

Proof. This is a consequence of Lemma 3.44 (7) and (3). □

It is also possible to define a block product principle as above, then we would see that this is equivalent to a substitution where we allow formulas on the prefix and suffix $\varphi^{<x}, \varphi^{>x}$ additionally.

3.5 Summary

In this chapter we gave the definition of the algebraic structure of a typed semigroup (Definition 3.1) and of morphisms between typed semigroups (Definition 3.2) and attained a typed syntactic semigroup (Definition 3.9) similar to the regular case. This allows us to hold on to the usual diagram where a typed semigroup $(S, \mathfrak{S}, \mathcal{E})$ recognizes a language L with syntactic semigroup $(S_L, \mathfrak{S}_L, \mathcal{E}_L)$, since the typed syntactic semigroup is the smallest semigroup recognizing the language L.



We assume here that $(S, \mathfrak{S}, \mathcal{E})$ is the image of h

Analogous to the Correspondence Theorem of Eilenberg Proposition 3.24 states that for every class of languages there is a corresponding class of typed semigroups, even for non-regular languages; for varieties there even is a one-to-one connection, see Theorem 3.31. This showed that the structure of a typed semigroup suffices to describe any language class.

The definition of the block product (Definition 3.33) allows to describe quantifier nesting if we know the typed semigroup corresponding to the quantifier. The proof of this will be given in a new way using the block product principle (Theorem 3.40) even in the case of unbounded variables. Our goal for the next chapter is therefore to describe the semigroups corresponding to quantifiers and predicates.

3.6 Further Research

In the definitions of this chapter we made some technical decisions on how to define the category of typed semigroups. These decisions were mainly made with the goal of establishing close relation to logic and circuit theory while keeping the algebra simple enough for using it concrete calculations, as we will see in the proceeding chapters. Since there might be other needs for typed semigroups we will not conceal some alternative definitions that might suit other needs better.

The first striking restriction is looking only at finitely generated semigroups. The reason is easy to see: Since our alphabet is always finite all semigroups ever considered for recognizing languages are finitely generated, so even if we allowed infinitely generated semigroups we would only examine subsemigroups that are finitely generated. For a more general theory it might be interesting to add infinitely generated semigroups. To arrive at a one to one correspondence as in Theorem 3.31, one needs a closure property stating that: if all finite subsemigroups of $(S, \mathfrak{S}, \mathcal{E})$ are in $\mathbf{V}$ then $(S, \mathfrak{S}, \mathcal{E})$ is in $\mathbf{V}$. But this would mean no modification of the rest of the theory. The question still remains whether this is of any use or just a generalization that obscuring proofs by introducing more technical details instead of giving deeper insight into the kernel of the theory.

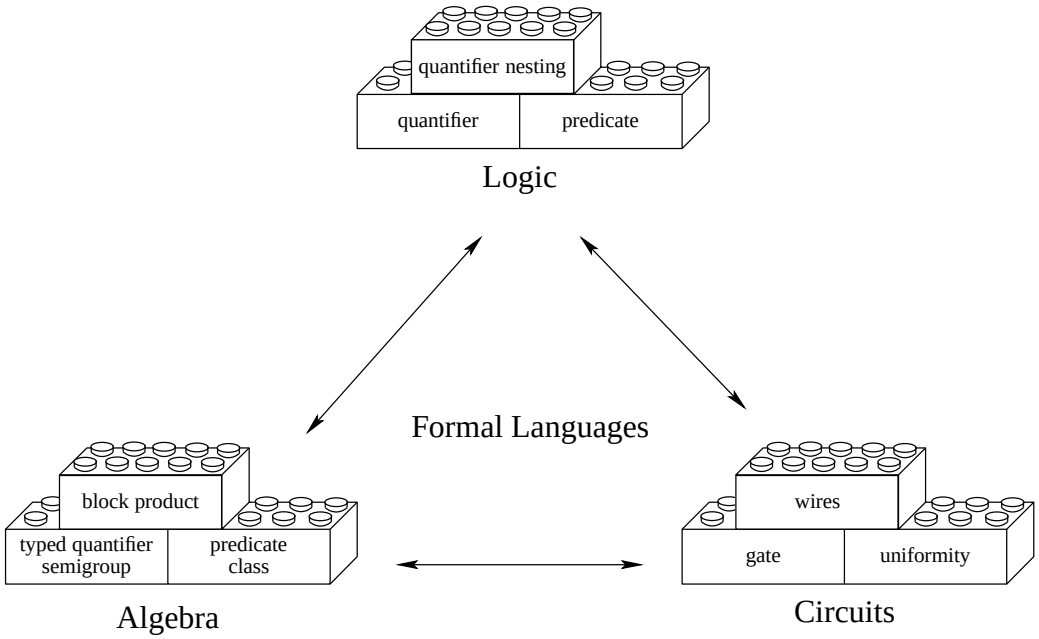
Loosening the restriction of a finite Boolean algebra for the type set and allowing infinite type sets on a semigroup would require a similar closure property as above for one to one correspondence, but otherwise the theory would remain unchanged. Similar considerations could be made allowing infinite direct products, but then we would need to pay close attention to the allowed types on such products. Albeit an extension as such is only useful if it helps giving results or deeper insight into the theory.

Another possible extensions would be to separate the types and units into positive and negative sets, with the goal to describe language classes that are not closed under complement, similar to the positive varieties introduced by Pin [Pin98]. One would have an obvious definition for recognition and with a few technical restriction gain a block product in these settings.

Chapter 4

Connections between Algebra, Logic and Circuits

In this chapter we will exhibit the three-way connection between algebra, logic and circuits. For the finite case the importance of this connection originates from the fact that we do not only have a correspondence between the languages recognized by the classes of all of the three systems, but also a linkage between the structure, i.e. the basic building blocks and the operations used to build more complex classes.



In logic we have quantifiers and predicates, which correspond to gates and uniformity in circuit families. For algebra the differences between quantifiers/gates and predicates/uniformity are not obvious; using the usual constructions we obtain typed semigroups where the set of units is unimportant and semigroups with only one unit.

So the "atomic" algebraic objects correspond more closely to the formulas or circuits of depth one than to either quantifiers or predicates.

A connection of this kind for two variables and majority logic was shown in [BKM07] and is extended here to an unbounded number of variables and arbitrary quantifiers.

4.1 Logic

In this section we present connections between the syntactic structure of logical formulas and a construction of typed semigroups. We start by defining a correspondence between quantifiers and algebra, where we prove connections to algebra for formulas with the equality predicate only. Then we link predicates to algebra and gain a results for formulas with larger predicate sets.

4.1.1 Quantifiers

It is quite natural to describe the computational power of a quantifier by a semigroup. We can also view this as a quantifier that captures the computational power of a semigroup. An example of this are monoidal quantifiers [BCST92], where we have correspondence between $\exists, \forall$ and U_1 or Mod_p and $\mathbb{Z}_p$.

We extend this correspondence to typed semigroups: Given an extended quantifier $Q\ x\ \langle\varphi_1, \ldots, \varphi_k\rangle$ as defined in Section 2.2, we will define a semigroup corresponding to this quantifier. Throughout of this section we will use this *quantifier semigroup* in the block product to obtain a correspondence between logic and algebra.

Definition 4.1 (Typed Quantifier Semigroup)**.** For a quantifier $Q^{(k)}$ we let $\Sigma_Q = \{0,1\}^k$. Then we define the *typed quantifier semigroup* $(S_Q, \mathfrak{S}_Q, \mathcal{E}_Q)$ to be the syntactic semigroup of $L_Q \subseteq (\{0,1\}^c)^+$, where

$$L_Q = \{w \mid w \models Q\ x\ \langle \mathfrak{c}_1^1(x), \mathfrak{c}_1^2(x), \ldots, \mathfrak{c}_1^k(x)\rangle\},$$

where $\mathfrak{c}_1^j(x)$ checks if the j-th entry of the letter at x is a 1, i.e.

$$w_{x=i} \models \mathfrak{c}_1^j(x) \iff \pi_j(w_i) = 1.$$

We denote by h_Q the syntactic morphism and by $\mathcal{S}_Q$ the accepting set $h_Q(L_Q)$.

In order to justify the naming we will examine the languages recognized by the quantifier semigroup in this proposition.

Proposition 4.2. *Let Q be a quantifier and $(S_Q, \mathfrak{S}_Q, \mathcal{E}_Q)$ be the typed quantifier semigroup of Q, then $\mathcal{L}(Q_1) = \mathcal{L}(\mathrm{wc}((S_Q, \mathfrak{S}_Q, \mathcal{E}_Q)))$.*

Proof. Let $L \subseteq \Sigma^+$ be a language in $\mathfrak{L}(Q_1)$. We let $h_Q : \Sigma_Q \to S_Q$ be the morphism as in the definition of the typed quantifier semigroup. The language L can be described by a Boolean combination of formulas or the form $Q\ x\ \langle\varphi_1, \ldots, \varphi_k\rangle$, where the φ_i are Boolean combinations of the $\mathfrak{c}_\sigma(x)$ predicates. Since the subformulas φ_l depend only on the letter at the position of x, we have a tuple $v_\sigma = \{0,1\}^k = \Sigma_Q$ for each $\sigma \in \Sigma$, such that $(v_\sigma)_l = 1$ iff $\sigma_{x=1} \models \varphi_l$. We let $h : \Sigma^+ \to (S_Q, \mathfrak{S}_Q, \mathcal{E}_Q) : \sigma \mapsto h_Q(v_\sigma)$, then $h(w) \in \mathcal{S}_Q$ iff $w \models Q\ x\ \langle\varphi_1, \ldots, \varphi_k\rangle$. For the Boolean combinations of the quantifiers we can use direct product of $(S_Q, \mathfrak{S}_Q, \mathcal{E}_Q)$.

For the other direction assume there is a morphism $h : \Sigma^+ \to (S, \mathfrak{S}, \mathcal{E})$, where $(S, \mathfrak{S}, \mathcal{E}) \in \mathrm{wc}((S_Q, \mathfrak{S}_Q, \mathcal{E}_Q))$. We can assume that $(S, \mathfrak{S}, \mathcal{E})$ is a direct product of $(S_Q, \mathfrak{S}_Q, \mathcal{E}_Q)$. Assume for a moment that $(S, \mathfrak{S}, \mathcal{E}) = (S_Q, \mathfrak{S}_Q, \mathcal{E}_Q)$.

Since $h(\Sigma) \subseteq \mathcal{E}_Q$, we can choose for each $\sigma \in \Sigma$ an letter $v_\sigma \in \Sigma_Q = \{0,1\}^k$ such that $h(\sigma) = h_Q(v_\sigma)$. Then we build formulas $\varphi_1, \ldots, \varphi_k$ that are Boolean combinations of $\mathfrak{c}_\sigma(x)$ predicates, such that $\sigma_{x=1} \models \varphi_l$ iff $(v_\sigma)_l = 1$.

If $(S, \mathfrak{S}, \mathcal{E})$ is a power of $(S_Q, \mathfrak{S}_Q, \mathcal{E}_Q)$, we use Boolean combinations of the formulas constructed for each factor as above. For any type $\mathcal{S} \in \mathfrak{S}$, this yields a depth on formula φ such that $w \models \varphi \iff h(w) \in \mathcal{S}$. □

We just constructed for a given quantifier a typed semigroup such that they recognize the same languages. The reverse, i.e. given a typed semigroup construct a quantifier, is also possible which leads to the definition of a *typed semigroup quantifier* as in the next definition.

Definition 4.3 (Typed Semigroup Quantifier)**.** Given a typed semigroup $(S, \mathfrak{S}, \mathcal{E})$, then for any map $f : \{0,1\}^k \to (S, \mathfrak{S}, \mathcal{E})$ and any type $\mathcal{S} \in \mathfrak{S}$, we let $Q^{f,\mathcal{S}}\ x\ \langle\varphi_1, \ldots, \varphi_k\rangle$ be a *typed semigroup quantifier* of $(S, \mathfrak{S}, \mathcal{E})$, also called a $(S, \mathfrak{S}, \mathcal{E})$-*quantifier*. For a k-tuple of formulas $\varphi_1, \ldots, \varphi_k$, we let $w \models Q^{f,\mathcal{S}}\ x\ \vec{\varphi}$ iff $\prod_{i=1}^{|w|} f(v^{(i)}) \in \mathcal{S}$, where $v^{(i)}$ is a k-tuple such that $v_l^{(i)} = 1$ if $w_{x=i} \models \varphi_l$ and $v_l^{(i)} = 0$ otherwise.

Since the algebra has no clear disjunction between "quantifiers" and "predicates", the typed semigroup quantifier is only a useful counterpart for certain typed semigroups. We will state a property that ensure a close connection between the typed semigroup quantifier and its typed semigroup. Please note that we show equivalence for the typed semigroups which is stronger than the equivalence for the languages they recognize.

Lemma 4.4. *Let $(S, \mathfrak{S}, \mathcal{E})$ be a typed semigroup, with $S = \mathcal{E}^+$. We let $\mathfrak{Q}$ be the set of all $(S, \mathfrak{S}, \mathcal{E})$-quantifiers and $\mathbf{Q}$ be the weakly closed class of all typed quantifier semigroups of $\mathfrak{Q}$, then $\mathbf{Q} = \mathrm{wc}((S, \mathfrak{S}, \mathcal{E}))$.*

Proof. We let $k = \lceil \log |\mathcal{E}| \rceil$ and fix any surjective map $f : \{0,1\}^k \to \mathcal{E}$, then the set of quantifiers $Q^{f,\mathcal{S}}$ for $\mathcal{S} \in \mathfrak{S}$ corresponds to the quantifier semigroup $(S, \mathcal{S}, \mathcal{E})$. By Lemma 3.15 the typed semigroup $(S, \mathfrak{S}, \mathcal{E}) \preceq \bigtimes_{\mathcal{S} \in \mathfrak{S}} (S, \mathcal{S}, \mathcal{E})$.

The other direction is trivial. □

When we lift the correspondence between logic and algebra to formulas of depth greater than one, the block product is involved on the algebraic side. As we have seen in the previous chapter the languages classes characterized by typed semigroups are always closed under inverse length preserving morphisms (Proposition 3.22). Since we will additionally need pointed languages in the inductions below, we extend the definition of length preserving morphisms to pointed languages.

Definition 4.5 (Nearly Length Preserving Morphism)**.** A morphism $\Sigma^+ \otimes X \to \Sigma'^+ \otimes X$ is *nearly length preserving* if $h((\Sigma \times \emptyset)) \subseteq \Sigma' \times \emptyset$.

If $X = \emptyset$, then a nearly length preserving morphism is a length preserving morphism.

The following proposition is the extension of the Proposition 4.2 to pointed languages for certain sets of quantifiers. We will use this proposition as the induction step in a theorem relating formulas with depth greater than one to block products of typed semigroups.

Proposition 4.6. *Let $\mathfrak{Q}$ be a set of quantifiers such that $\mathcal{P}(\mathfrak{Q}_1[=])$ is closed under inverse nearly length preserving morphisms. Let* **Q** *be the weakly closed class of all typed quantifier semigroups corresponding to $\mathfrak{Q}$, then*

$$\mathcal{P}(\mathfrak{Q}_1[=]) = \mathcal{P}(\mathbf{Q})).$$

Proof. Let $L \in \mathcal{P}(\mathfrak{Q}_1[=])$, with $L \subseteq \Sigma^+ \otimes X$. There are multiple possible languages $L' \subseteq (\Sigma \times 2^X)^+$ with $L = L' \cap \Sigma^+ \otimes X$, but in general $L' \notin \mathcal{L}(\mathfrak{Q}_1)$. Since we do not have any predicates except equality, we can replace the equality predicate by the query predicate and obtain a language $L' \in \mathcal{L}(\mathfrak{Q}_1)$ such that $L = L' \cap \Sigma^+ \otimes X$. By Proposition 4.2 we have $L' \in \mathcal{L}(\mathbf{Q})$ and hence $L \in \mathcal{P}(\mathbf{Q})$.

For the other direction we cannot argue in the same way. Because if there is a type morphism recognizing L as pointed language $h : ((\Sigma \times 2^X)^+, L, \Sigma \times \emptyset) \to (S, \mathfrak{S}, \mathcal{E})$ for $(S, \mathfrak{S}, \mathcal{E}) \in \mathbf{Q}$, then there is no reason that $h((\Sigma \times 2^X)) \subseteq \mathcal{E}$, so we cannot use the same morphism to recognize a language L' by this morphism.

We choose a map $\tilde{h} : \Sigma \times 2^X \to \mathcal{E}^+$ with $\tilde{h}((\sigma, \emptyset)) = h((\sigma, \emptyset))$ and otherwise $\tilde{h}((\sigma, \vec{x})) = w$ with $\prod_i w_i = h((\sigma, \vec{x}))$. We extend $\tilde{h}$ to a morphism from $(\Sigma \times 2^X)^+$ to $\mathcal{E}^+$. Then $\tilde{h}$ is nearly length preserving so it suffices to show $\tilde{L} = \tilde{h}(L)$ is in $\mathcal{P}(\mathfrak{Q}_1[=])$. Also note that we have a natural morphism $h' : (\mathcal{E}^+, \tilde{L}, \mathcal{E}) \to (S, \mathfrak{S}, \mathcal{E})$ by $h'(w) = \prod_i w_i$. But now we can apply Lemma 4.2 and see that $\tilde{L} \in \mathcal{L}(\mathfrak{Q}_1)$. But this implies $L \in \mathcal{P}(\mathfrak{Q}_1[=])$. □

In the next theorem we use the fact that all formulas can be constructed from formulas of depth one by the substitution principle (Definition 2.2), and that the typed semigroups we consider can be constructed from the typed quantifier semigroup by the block product (Definition 3.33). Please note that we prove this theorem for pointed languages which is a stronger statement than for languages without free variables.

Theorem 4.7. *Let $\mathfrak{Q}$ be a set of quantifiers such that $\mathcal{P}(\mathfrak{Q}_1[=])$ is closed under inverse nearly length preserving morphisms. We let* **Q** *be the weakly closed class of all typed quantifier semigroups corresponding to $\mathfrak{Q}$, then*

1. $\mathcal{P}(\mathrm{sbpc}(\mathbf{Q})) = \mathcal{P}(\mathfrak{Q}[=])$,

2. $\mathcal{P}(\mathrm{wbpc}(\mathbf{Q})) = \mathcal{P}(\mathfrak{Q}_2[=])$.

Proof. "$\supseteq$" Let φ be a formula in $\mathfrak{Q}[=]$. If φ has depth 1, the inclusion follows in both cases from Proposition 4.6. If φ has depth more than one, we look at the cases of unbounded number of variables first. Using substitution (Definition 2.2) we can decompose φ as a Φ-substitution of φ', where φ is of depth 1 and all formulas of Φ have lower depth than φ. By induction the formulas of Φ are recognized by a typed semigroup $(S, \mathfrak{S}, \mathcal{E}) \in \mathrm{sbpc}(\mathbf{Q})$, and φ' is recognized by Proposition 4.6 by a semigroup of **Q**. Hence by the block product principle (Lemma 3.40) L_φ is recognized by $(S_Q, \mathfrak{S}_Q, \mathcal{E}_Q) \boxdot (S, \mathfrak{S}, \mathcal{E}) \in \mathrm{sbpc}(\mathbf{Q})$.

In the case of two variables we know by decomposition proposition φ is a Φ-substitution of φ', where φ is of lower depth and all formulas of Φ have depth 1. By induction the formula φ' is recognized by a typed semigroup of $(S, \mathfrak{S}, \mathcal{E}) \in \mathrm{wbpc}(\mathbf{Q})$ and the formulas Φ are recognized by Proposition 4.6 by a semigroup of **Q**, hence by the block product principle φ is recognized by $(S, \mathfrak{S}, \mathcal{E}) \boxdot \mathbf{Q} \in \mathrm{wbpc}(\mathbf{Q})$.

"$\subseteq$" For the other direction assume a language is recognized by $(S, \mathfrak{S}, \mathcal{E})$. We are done in both cases if $(S, \mathfrak{S}, \mathcal{E})$ divides a semigroup of **Q** by Proposition 4.6. Otherwise in the case of an unbounded number of variables we can assume that $(S, \mathfrak{S}, \mathcal{E}) = \mathbf{Q} \boxdot (S', \mathfrak{S}', \mathcal{E}')$. By induction we know $\mathcal{P}((S', \mathfrak{S}', \mathcal{E}')) \subseteq \mathcal{P}(\mathfrak{Q}[=])$ so we can apply Theorem 3.40 and get $\mathcal{P}((S, \mathfrak{S}, \mathcal{E})) \subseteq \mathcal{P}(\mathfrak{Q}_1[=] \circ \mathfrak{Q}[=]) = \mathcal{P}(\mathfrak{Q}[=])$. In the case of two variables we can assume that $(S, \mathfrak{S}, \mathcal{E}) = (S', \mathfrak{S}', \mathcal{E}') \boxdot \mathbf{Q}$. By induction we know $\mathcal{P}((S', \mathfrak{S}', \mathcal{E}')) \subseteq \mathcal{P}(\mathfrak{Q}_2[=])$ so we can apply Theorem 3.40 and get $\mathcal{P}((S, \mathfrak{S}, \mathcal{E})) \subseteq \mathcal{P}(\mathfrak{Q}_2[=] \circ \mathfrak{Q}_1[=]) = \mathcal{P}(\mathfrak{Q}_2[=])$.

□

4.1.2 Numerical Predicates

We turn our attention to the sets of numerical predicates used by the logic classes and define an algebraic counterpart. We will use this correspondence to prove theorems as in the previous section but for logic classes with larger predicate sets.

Each predicate $\mathfrak{p}(x_1, \ldots, x_k)$ has a natural connection to its pointed language $L_\mathfrak{p} \subseteq \{a\}^+ \otimes \{x_1, \ldots, x_k\}$. This language can be understood as a language $\hat{L}_\mathfrak{p} \subseteq (\{a\} \times 2^X)^+$ such that $\hat{L}_\mathfrak{p} \cap \{a\}^+ \otimes \{x_1, \ldots, x_k\} = L_\mathfrak{p}$ The choice of the language $\hat{L}_\mathfrak{p}$ is not unique.

When we have a typed semigroup $(S, \mathfrak{S}, \mathcal{E})$ that recognizes $L_\mathfrak{p}$, then there is a morphism $h : (\Sigma \times 2^X)^+ \to (S, \mathfrak{S}, \mathcal{E})$ and a type $\mathcal{S} \in \mathfrak{S}$ such that $L_\mathfrak{p} = h^{-1}(\mathcal{S}) \cap \{a\} \otimes X$. Here $\hat{L}_\mathfrak{p} = h^{-1}(\mathcal{S})$ is a language in $(\Sigma \times 2^X)^+$. This difference will be important in the considerations of this section.

$P_<$	$(\mathbf{1},1)$	$(\mathbf{1},0)$	$(f,1)$	$(f,0)$	0	Sample for a word
$(\mathbf{1},1)$	$(\mathbf{1},1)$	$(\mathbf{1},0)$	$(f,1)$	$(f,0)$	0	ε
$(\mathbf{1},0)$	$(\mathbf{1},0)$	$(\mathbf{1},0)$	0	0	0	w_x
$(f,1)$	$(f,1)$	$(f,0)$	$(f,1)$	$(f,0)$	0	w_y
$(f,0)$	$(f,0)$	$(f,0)$	0	0	0	$w_{x=i,y=j}$ with $i<j$
0	0	0	0	0	0	$w_{x=i,y=j}$ with $i\geq j$

Figure 4.1: *A sample for a typed predicate semigroup for the order predicate*

For any predicate $\mathfrak{p}$ we call a typed semigroup $(S,\mathfrak{S},\mathcal{E})$ that recognizes $L_\mathfrak{p}\subseteq\{a\}^+\otimes X$ a typed predicate semigroup for $\mathfrak{p}$. We will examine the order predicate and construct an explicit typed predicate semigroup for order and even further characterize all predicate classes for the order predicate later in this section. Now we look at the order predicate.

We can give a concrete typed semigroup that is a predicate semigroup for the order predicate. Let $f: U_1\times U_1\to U_1$ be a function with $f(s_1,s_2)=0$ iff $s_1=0$ and $f(s_1,s_2)=1$ otherwise. We let $P_<=\{(\mathbf{1},1),(\mathbf{1},0),(f,1),(f,0),(\mathbf{0},0)\}$, then $(P_<,(\mathbf{0},0),(\mathbf{1},1))$ is an order predicate monoid (see Figure 4.1).

We will continue to characterize sets of predicates.

Definition 4.8 (Predicate Class Of Typed Semigroups)**.** Let $\mathfrak{P}$ be a set of predicates. A *predicate class* for $\mathfrak{P}$ is a class of typed semigroup $\mathcal{P}$ such that: For a predicate $\mathfrak{p}\in\mathfrak{P}$, the language $L_\mathfrak{p}\in\mathcal{P}(\mathcal{P})$ iff $\mathfrak{p}\in\mathfrak{P}$.

This definition does not assure that there is any set of predicates such that a predicate class exists. The following proposition fills this gap, showing that for most predicate sets usually considered such a predicate class exists.

Proposition 4.9. *If $\mathfrak{P}$ is a set of predicates, such that $\mathcal{P}(\mathfrak{P})$ is closed under inverse nearly length preserving inverse morphisms, then there is a predicate class for $\mathfrak{P}$.*

Proof. Let $P\in\mathfrak{P}$. We let $L_P\subseteq\{a\}^+\otimes X\subseteq(\{a\}\times 2^X)^+$ be the language corresponding to the predicate P, and $(S_P,\mathfrak{S}_P,\mathcal{E}_P)=((\{a\}\times 2^X)^+,L_P,\{a\})$. We need to show that if for a predicate P' the language $L_{P'}$ is recognized by $(S_P,\mathfrak{S}_P,\mathcal{E}_P)$, then $L_{P'}$ is a nearly length preserving inverse morphism of L_P, but since $L_{P'}$ is recognized by $(S_P,\mathfrak{S}_P,\mathcal{E}_P)$ we have a morphism $((\{a\}\times 2^{X'}),L_{P'},\{a\}\times\emptyset)\to(S_P,\mathfrak{S}_P,\mathcal{E}_P)=((\{a\}\times 2^X)^+,L_P,\{a\})$, and hence $P'\in\mathfrak{P}$, since $\mathfrak{P}$ is closed under nearly length preserving morphisms. □

We will now characterize all typed predicate semigroups for the order predicate. The following lemma states that any monoid where the one is the only unit can express at most order.

Lemma 4.10. *Let $(S,\mathfrak{S},\mathcal{E})$ be a typed monoid and $\mathcal{E}=\{1\}$, then*

$$\mathcal{P}((S,\mathfrak{S},\mathcal{E}))\subseteq\mathcal{P}(\mathrm{wc}(P_<,(\mathbf{0},0),(\mathbf{1},1))).$$

Proof. Let L_P be recognized by a morphism h to $(S, \mathfrak{S}, \mathcal{E})$ where $\mathcal{E} = \{1\}$, then: $h(w_{\vec{x}}) = h(w'_{\vec{x}})$ if the order or all free variables is the same, since the morphism maps to the neutral element at all other positions. Hence the predicate P is a Boolean combination of the order predicate and can be recognized by a morphism to a direct product of an order monoid. □

Definition 4.11 (Order Predicate Monoid)**.** Any typed non-commutative monoid $(S, \mathfrak{S}, \mathcal{E})$ where 1_S is the only unit, i.e. $\mathcal{E} = \{1_S\}$, is an *order predicate monoid*. In the following we let $\mathbf{P}_<$ be set of all *order predicate monoids*.

We begin to show the correspondence between logic and algebra starting with the formulas of depth one.

Lemma 4.12. *Let $\mathfrak{P}$ be a set of predicates, such that $\mathcal{P}(\mathfrak{P})$ is closed under inverse nearly length preserving morphisms,* **P** *be the corresponding predicate class, $\mathfrak{Q}$ be a set of quantifiers and* **Q** *the set of typed quantifier semigroups for $\mathfrak{Q}$, then*

$$\mathcal{L}(\mathfrak{Q}_1[\mathfrak{P}]) = \mathcal{L}(\mathbf{Q} \boxdot \mathbf{P}).$$

Proof. It is easy to see that a $\mathfrak{Q}_1[\mathfrak{P}]$ formula is a $\mathfrak{P}$-substitution of a $\mathfrak{Q}_1$ formula. Since $\mathcal{L}(\mathfrak{Q}_1) = \mathcal{L}((S_Q, \mathfrak{S}_Q, \mathcal{E}_Q))$ and $\mathcal{P}(\mathfrak{P}) = \mathcal{P}((S_P, \mathfrak{S}_P, \mathcal{E}_P))$, we have $\mathcal{L}(\mathfrak{Q}_1[\mathfrak{P}]) = \mathcal{L}((S_Q, \mathfrak{S}_Q, \mathcal{E}_Q) \boxdot (S_P, \mathfrak{S}_P, \mathcal{E}_P))$ by the block product principle (Lemma 3.40). □

Again we need pointed languages in the inductive step later so we prove the following lemma for formulas of depth one and pointed languages.

Lemma 4.13. *Let $\mathfrak{P}$ be a set of predicates, such that $\mathcal{P}(\mathfrak{P})$ is closed under inverse nearly length preserving morphisms,* **P** *be the corresponding predicate class, $\mathfrak{Q}$ be a set of quantifiers, such that $\mathcal{P}(\mathfrak{Q})$ is closed under inverse nearly length preserving morphism, and* **Q** *the set of typed quantifier semigroups for $\mathfrak{Q}$, then*

$$\mathcal{P}(\mathfrak{Q}_1[\mathfrak{P}]) = \mathcal{P}(\mathbf{Q} \boxdot \mathbf{P}).$$

Proof. Again a $\mathfrak{Q}_1[\mathfrak{P}]$ formula is a $\mathfrak{P}$-substitution of a $\mathfrak{Q}_1$ formula. Since $\mathcal{P}(\mathfrak{Q}_1) = \mathcal{P}(\mathbf{Q})$ and $\mathcal{P}(\mathfrak{P}) = \mathcal{P}(\mathbf{P})$, we have $\mathcal{P}(\mathfrak{Q}_1[\mathfrak{P}]) = \mathcal{P}(\mathbf{Q} \boxdot \mathbf{P})$ by the block product principle (Lemma 3.40). □

We will proceed to use the block product principle (Lemma 3.40) as in the previous chapter to show the correspondence for arbitrary logic classes.

Theorem 4.14. *Let $\mathfrak{P}$ be a set of predicates, such that $\mathcal{P}(\mathfrak{P})$ is closed under inverse nearly length preserving morphisms,* **P** *be the corresponding predicate class, $\mathfrak{Q}$ be a set of quantifiers, such that $\mathcal{P}(\mathfrak{Q})$ is closed under inverse nearly length preserving morphism, and* **Q** *the set of typed quantifier semigroups for $\mathfrak{Q}$, then*

- $\mathcal{P}(\mathrm{sbpc}(\mathbf{Q} \cup \mathbf{P})) = \mathcal{P}(\mathfrak{Q}[\mathfrak{P}])$,

P_{succ}	$(f,1)$	$(\mathbf{1},0)$	$(f_<,0)$	$(f_>,0)$	$(f_{<>},0)$	0	Sample for a word
$(f,1)$	$(f,1)$	$(f_>,0)$	$(f_{<>},0)$	$(f_>,0)$	$(f_{<>},0)$	0	$(a,\emptyset)^+$
$(\mathbf{1},0)$	$(f_>,0)$	$(\mathbf{1},0)$	$(f_<,0)$	0	0	0	$(a,x)^+$
$(f_<,0)$	$(f_<,0)$	0	0	0	0	0	$(a,\emptyset)^+(a,x)^+$
$(f_>,0)$	$(f_{<>},0)$	$(f_>,0)$	$(f_{<>},0)$	0	0	0	$(a,x)^+(a,\emptyset)^+$
$(f_{<>},0)$	$(f_{<>},0)$	0	0	0	0	0	$(a,\emptyset)^+(a,x)^+(a,\emptyset)^+$
0	0	0	0	0	0	0	$(a,x)^+(a,\emptyset)^+(a,x)^+$

Figure 4.2: *A sample for a typed predicate semigroup for the* succ *predicate*

- $\mathcal{P}(\text{wbpc}(\mathbf{Q} \boxdot \mathbf{P})) = \mathcal{P}(\mathfrak{Q}_2[\mathfrak{P}])$.

Proof. "$\supseteq$" Let φ be a formula in $\mathfrak{Q}[\mathfrak{P}]$. If φ has depth 1, the result follows in both cases from Proposition 4.13. If φ has depth more than one, we look at the cases of unbounded number of variables first. By the decomposition proposition φ is a Φ-substitution of φ', where φ is of depth 1 and all formulas of Φ have lower depth. By induction the formulas of Φ are recognized by a typed semigroup $(S,\mathfrak{S},\mathcal{E}) \in \text{sbpc}(\mathbf{Q} \cup \mathbf{P})$, and φ' is recognized by Proposition 4.13 by a semigroup of $(\mathbf{Q} \boxdot \mathbf{P})$. Hence by the block product principle (Lemma 3.40) φ is recognized by $(\mathbf{Q} \boxdot \mathbf{P}) \boxdot (S,\mathfrak{S},\mathcal{E})$, which divides $\mathbf{Q} \boxdot (\mathbf{P} \boxdot (S,\mathfrak{S},\mathcal{E})) \in \text{sbpc}(\mathbf{Q} \cup \mathbf{P})$.

In the case of two variables we know by decomposition proposition φ is a Φ-substitution of φ', where φ is of lower depth and all formulas of Φ have depth 1. By induction the formula φ' is recognized by a typed semigroup of $(S,\mathfrak{S},\mathcal{E}) \in \text{wbpc}(\mathbf{Q} \boxdot \mathbf{P})$ and the formulas Φ are recognized by Proposition 4.13 by a power of $\mathbf{Q} \boxdot \mathbf{P}$, hence by the block product principle φ is recognized by

$$(S,\mathfrak{S},\mathcal{E}) \boxdot (\mathbf{Q} \boxdot \mathbf{P}) \in \text{wbpc}(\mathbf{Q} \boxdot \mathbf{P}).$$

"$\subseteq$" For the other direction assume a language is recognized by $(S,\mathfrak{S},\mathcal{E})$. We are done in both cases if $(S,\mathfrak{S},\mathcal{E})$ divides a semigroup of $\mathbf{Q} \boxdot \mathbf{P}$ by Proposition 4.13. Otherwise in the case of an unbounded number of variables we can assume that $(S,\mathfrak{S},\mathcal{E}) = (\mathbf{Q} \boxdot \mathbf{P}) \boxdot (S',\mathfrak{S}',\mathcal{E}')$. By induction we know $\mathcal{P}((S',\mathfrak{S}',\mathcal{E}')) \subseteq \mathcal{P}(\mathfrak{Q}[\mathfrak{P}])$ so we can apply Theorem 3.40 and get $\mathcal{P}((S,\mathfrak{S},\mathcal{E})) \subseteq \mathcal{P}(\mathfrak{Q}_1[\mathfrak{P}] \circ \mathfrak{Q}[\mathfrak{P}]) = \mathcal{P}(\mathfrak{Q}[\mathfrak{P}])$. In the case of two variables we can assume that $(S,\mathfrak{S},\mathcal{E}) = (S',\mathfrak{S}',\mathcal{E}') \boxdot (\mathbf{Q} \boxdot \mathbf{P})$. By induction we know $\mathcal{P}((S',\mathfrak{S}',\mathcal{E}')) \subseteq \mathcal{P}(\mathfrak{Q}_2[\mathfrak{P}])$ so we can apply Theorem 3.40 and get $\mathcal{P}((S,\mathfrak{S},\mathcal{E})) \subseteq \mathcal{P}(\mathfrak{Q}_2[\mathfrak{P}] \circ \mathfrak{Q}_1[\mathfrak{P}]) = \mathcal{P}(\mathfrak{Q}_2[\mathfrak{P}])$. □

We end this section by examining the closure properties of the languages recognized by logic classes depending on the predicates used. For the languages $\mathcal{L}(\mathfrak{Q}[\mathfrak{P}])$ we can determine certain closure properties depending on closure properties of the predicates. For this reason we give a definition for the succ predicate.

Definition 4.15 (Successor Predicate Monoid)**.** Let P_{succ} be the submonoid of $U_1 \square U_1$ generated by $(f,1)$,$(\mathbf{1},0)$, where $f : U_1 \times U_1 \to U_1$ with $f(s_1,s_2) = 0$ if $s_1 = s_2 = 0$

and $f(s_1, s_2) = 1$ otherwise. Then the *typed successor monoid* is $(P_{succ}, (\mathbf{1}, 0), (f, 1))$, we let $\mathbf{P}_{\mathbf{succ}}$ be the smallest weakly closed class of typed semigroup that contains this monoid. We let $\mathbf{P}_{<,\mathbf{succ}} = \mathbf{P}_{<} \times \mathbf{P}_{\mathbf{succ}}$.

$\mathbf{P}_{\mathbf{succ}}$ computes the the predicate that is true iff the distance between x and y is 1, i.e. $x = y+1 \vee y = x+1$. Please note that $(f, 1)$ of the previous definition is idempotent.

The following remark is inspired by to [Str02, theorem 3].

Remark 4.16. If $\mathfrak{Q}$ is a set of quantifiers, such that $\mathcal{P}(\mathfrak{Q}_1)$ is closed under inverse morphisms, and $\mathfrak{P}$ is a set of predicates, then

- If for each predicate in $\mathfrak{P}$ there is a predicate semigroup such that the unit is the identity, then the language class $\mathcal{L}(\mathfrak{Q}[\mathfrak{P}])$ is closed under erasing morphisms.
- If for each predicate in $\mathfrak{P}$ there is a predicate semigroup where the unit is an idempotent, then the language class $\mathcal{L}(\mathfrak{Q}[\mathfrak{P}])$ is closed under non-erasing morphisms.
- If for each predicate in $\mathfrak{P}$ the predicate semigroup is a group and closed under shifting, then the language class $\mathcal{L}(\mathfrak{Q}[\mathfrak{P}])$ is closed under length multiple morphisms.

Note that in the previous remark that the predicate semigroups where the unit is the identity correspond to the predicates that are Boolean combinations of the order predicate (Lemma 4.10), and by definition of P_{succ} the identity of the typed successor semigroup is an idempotent. For the third case of the previous remark we have the set of modulo predicates that fulfill the conditions.

4.2 Circuits

We will progress in a similar with circuits as in the logic section. A logical formula and constant depth circuits are highly equivalent as expressed in [BL06] or for linear size circuits in [BKM07]. We will use this similarity to structure the proofs in a similar fashion.

4.2.1 Gates

Usually circuits are studied only for AND, OR and MOD gate types. We will examine here arbitrary gate types as defined in Section 2.4. In order to allow arbitrary gate types we need a way to describe the function they compute. Following the algebraic approach we use typed semigroups for this purpose. We will define in a natural way a language that characterizes the gate type.

Definition 4.17 (Typed Gate Semigroup)**.** Let G be a gate type, i.e. a family of functions $f_G^{(r)} : \{0,1\}^r \to \{0,1\}$ for $r \in \mathbb{N}$. We let $f_G : \{0,1\}^+ \to \{0,1\}$ be a

function so that $f_G(w) = 1$ if the value of a gate of type G with $|w|$ inputs that are assigned the truth values $w_1, \ldots, w_n$ is true, i.e. $f_G(w) = f_G^{(r)}(w_1, \ldots, w_r)$. We let $L_G = \{w \in \{0,1\}^+ \mid f_G(w) = 1\}$, then $(S_G, \mathfrak{S}_G, \mathcal{E}_G) = \mathrm{syn}(L_G)$ is the *typed gate semigroup* and h_G the syntactic morphism.

Since the input size of a gate is not fixed to the input length and the wiring can be non-uniform we cannot expect to have a correspondence between the languages recognized by a circuit family with a single gate and the languages recognized by its typed gate semigroup via morphisms. We introduce programs over typed semigroups to close the gap.

Definition 4.18 (Program)**.** Let $L \subseteq \Sigma^+$ be a language and $(S, \mathfrak{S}, \mathcal{E})$ be a typed semigroup. An *instruction* is a pair (i, f), where i is a natural number and f is map from Σ to $\mathcal{E}$, where there are $s_1, s_2 \in \mathcal{E}$ and $\sigma' \in \Sigma$ such that

$$f(\sigma) = \begin{cases} s_1 & \text{if } \sigma = \sigma', \\ s_2 & \text{otherwise.} \end{cases}$$

A *program* P_n over $(S, \mathfrak{S}, \mathcal{E})$ is a sequence of instructions $(i_1, f_1) \ldots (i_l, f_l)$, where the indices $i_j \leq n$. Then P_n defines a map $w \mapsto P_n(w)$ by $P_n(w) = \prod_{j=1}^{l} f_j(w_{i_j})$. The language L is recognized by a family of programs $(P_n)_{n\in\mathbb{N}}$ if there is a type $\mathcal{S} \in \mathfrak{S}$ such that $w \in L$ with $|w| = n$ iff $P_n(w) \in \mathcal{S}$.

For a set $\mathbf{S}$ of semigroups we denote by $\mathcal{L}(\pi - \mathbf{S})$ the set of languages recognized by programs over a semigroup of $\mathbf{S}$.

Please note that if we would not restrict the instructions for the programs of all lengths to map to a finite set, there underlying semigroup would have few information of the complexity of the recognized language. For example the semigroup $(\mathbb{Z}, \mathbb{Z}^+) \boxdot (\mathbb{Z}, \mathbb{Z}^+)$ would already recognize all language by programs of linear length.

Lemma 4.19. *A language L is recognized by a depth one circuit with linear/polynomial wires and the gate type G, iff L is recognized by a linear/polynomial length family of programs over the typed gate semigroup $(S_G, \mathfrak{S}_G, \mathcal{E}_G)$ of G.*

Proof. Let L be recognized by a circuit of depth one, then the circuit consists of one gate. We let $(i_1, \sigma_1), \ldots, (i_l, \sigma_l)$ be the wiring of the circuit for a word of length n, i.e. the $j-th$ input of the gate queries if the i_j-th letter is σ_j.

So we can define a program $(i_1, f_1), \ldots, (i_l, f_l)$ by $f_j(\sigma) = h_G(1)$ iff $\sigma = \sigma_j$ and $f_j(\sigma) = h_G(0)$ otherwise. By definition of the gate semigroup this family of program recognizes L.

The other direction is equivalent. □

In the following theorem we extend the previous lemma to constant depth circuits. Compared to the logic case it emerges that we need only the weak block product closure in the following theorem. The reason for this is simply the fact that the programs can have polynomial size, and hence we multiply a polynomial number of elements, so compared to the length of the program the circuit has linear size.

Theorem 4.20. *Let G be a gate type where the gate semigroup* $(S_G, \mathfrak{S}_G, \mathcal{E}_G)$ *is a group. The following three statements are equivalent:*

- *A language L is recognized by a constant depth circuit of polynomial size and polynomial wires with G gates.*
- *A language L is recognized by a polynomial length program over a typed semigroup of* $\mathrm{wbpc}((S_G, \mathfrak{S}_G, \mathcal{E}_G))$.
- *A language L is recognized by a polynomial length program over a typed semigroup of* $\mathrm{sbpc}((S_G, \mathfrak{S}_G, \mathcal{E}_G))$.

Proof. We will only show that the first statement implies the second. By Proposition 3.42 the second statement implies the third. The construction of a circuit for the program of the third statement is a straight-forward induction due to Lemma 4.19.

Let d be the depth of the circuit, than we can assume that there is a polynomial $p(n)$ and for each word length n, the circuit is a complete $p(n)$-tree. So all gates except the gates at the lowest level, i.e. the gates that access the input, form a circuit over the output of the bottom level gates. We do induction on the depth of the circuit.

If the circuit has depth one we can apply Lemma 4.19, otherwise we know by induction that the language computed by the gates which are not bottom level gates can also be computed by a program π to a semigroup of $(S, \mathfrak{S}, \mathcal{E}) \in \mathrm{wbpc}((S_G, \mathfrak{S}_G, \mathcal{E}_G))$.

In the program π a command is of the form (i, f), where i is the number of the bottom level gate and $f : \{0,1\} \to (S, \mathfrak{S}, \mathcal{E})$ is a map.

Let $(S', \mathfrak{S}', \mathcal{E}') = (S, \mathfrak{S}, \mathcal{E}) \boxdot (S_G, \mathfrak{S}_G, \mathcal{E}_G)$ and let π'_i be a program to $(S_G, \mathfrak{S}_G, \mathcal{E}_G)$ that computes the bottom level gate i, we can natural change this into a program to $(S', \mathfrak{S}', \mathcal{E}')$ by the embedding $(S_G, \mathfrak{S}_G, \mathcal{E}_G) \to (S, \mathfrak{S}, \mathcal{E}) \boxdot (S_G, \mathfrak{S}_G, \mathcal{E}_G) : g \mapsto (\mathbf{1}_S, g)$.

We create a program by replacing every command (i, f) of π by a sequence of commands $\pi'_i f' \pi'^{-1}_i$ where $f' = (f'', 1_{S_G})$ and $f'' : (S_G, \mathfrak{S}_G, \mathcal{E}_G) \times (S_G, \mathfrak{S}_G, \mathcal{E}_G) \to (S, \mathfrak{S}, \mathcal{E})$ with $f''(m_1, m_2) = f(1)$ iff $m_1 m_2 \in A$ for the type A corresponding to the program π'_i and $f''(m_1, m_2) = f(0)$ otherwise.

Please note that $\pi'_i f' \pi'^{-1}_i$ multiplies out to $(f(1), 1_{S_G})$ if the i-th bottom gate is true and $(f(0), 1_{S_G})$ otherwise. Hence the new program computes the correct language. □

For majority logic and threshold circuits we get:

Corollary 4.21. *A language L is recognized by a* $TC0$ *circuit family iff it recognized by a program over* $\mathrm{wbpc}((\mathbb{Z}, \mathbb{Z}^+, \pm 1))$.

4.2.2 Uniformity

In the previous section we used programs to describe the languages recognized by circuit families. In order to overcome this limitation and obtain a description by morphisms, we introduce a uniformity language. All circuits considered in the following will have a uniformity languages, which does not imply that the circuit

cannot be highly non-uniform unless we have a uniform description of the uniformity language.

We will define the uniformity language as a pointed language, this has the advantage that we do not need to worry about the encoding of variable positions, but have a clean way to describe the circuits. Uniformity languages that encode the variable positions always get a difference between the uniformity of the circuit and the complexity of the uniformity language, depending on the encoding. So the results in the following are basically the same for any definition of a uniformity language, but there might be a slight variation in the exact complexity.

Definition 4.22 (Uniformity Language)**.** Assume we have a circuit with n inputs. We label the gates by tuples of the numbers 1 to n, where the first entry is bounded by a constant l. We assume that the input length n is always greater or equal to l. The gate numbers $(k, i, 1, \ldots, 1, 1)$ for $k = 1, \ldots, |\Sigma|$, $i = 1, \ldots, n$ are reserved as the input gates that query if there is a letter σ_k at position i, and the gate label $(l, n, \ldots, n)$ is reserved for the output gate.

We let $\mathrm{Conn}(\vec{x}, \vec{y})$ be a predicate, such that it is true if the gate $\vec{x}$ has the gate $\vec{y}$ input. The order of the inputs is given by the following order

$$(1, 1, \ldots, 1), (2, 1, \ldots, 1), \ldots, (l, 1, \ldots, 1), (1, 2, \ldots, 1), \ldots, (l, n, \ldots, 1), \ldots, (l, n, \ldots, n).$$

If the label of a gate is $(x_1, x_2, \ldots, x_k)$, then x_1 determines the gate type, i.e. there is a map from $\{1, \ldots, l\}$ to the gate types including the input gate types and the output gate type.

The uniformity language is the predicate language L_{Conn}. A circuit family is Conn-uniform if Conn describes the uniformity language. Also an Conn-uniform circuit family is X-uniform if X recognizes L_{Conn}, where X is a class of formulas or a class of typed semigroups.

We always assume that the input length n is greater than the largest constant used in the first component of the tuples used for the labeling.

Again for induction purposes we need to define when a circuit recognizes a pointed language. In logic this is handled by free variables, hence the formula has no need to use the query predicate $\mathfrak{c}$ to find the position of the extra information in the input. For circuits we do not have free variables, but we can describe circuits with multiple output gates.

Definition 4.23. A family of circuits $(C_n)_n$ recognizes a language $L \subseteq \Sigma^+ \otimes X$ if there is a vector $\vec{c}$ such that the gate labeled $(\vec{c}, \vec{i})$ outputs 1 for input w iff $w_{\vec{x}=\vec{i}} \in L$.

Definition 4.24. Let Σ be an alphabet and X be a set of variables and $(C_n)_{n\in\mathbb{N}}$ be a family of circuits. We let $\mathcal{P}^{\Sigma,X}((C_n)_{n\in\mathbb{N}}) = \{L \mid L \subseteq \Sigma^+ \otimes X \text{ is recognized by } (C_n)_{n\in\mathbb{N}}\}$ and $\mathcal{P}((C_n)_{n\in\mathbb{N}}) = \bigcup_{\Sigma,X} \mathcal{P}^{\Sigma,X}((C_n)_{n\in\mathbb{N}})$. Finally we denote the set of languages with one free variable by $\mathcal{P}_1((C_n)_{n\in\mathbb{N}}) = \bigcup_{\Sigma,|X|=1} \mathcal{P}^{\Sigma,X}((C_n)_{n\in\mathbb{N}})$.

Definition 4.25. For a set of predicates $\mathfrak{P}$ and a set of gate types $\mathfrak{Q}$, we denote by $\mathfrak{P}$-uniform $\mathfrak{Q}C^0$ for the class of all families of $\mathfrak{P}$-uniform circuits with $\mathfrak{Q}$ gates, with polynomial size and constant depth. If we restrict the fan-in of each gate to linear size we write $\mathfrak{P}$-uniform $\mathfrak{Q}C^0_{lin}$. If we further restrict the size of the circuit to linear size we write $\mathfrak{P}$-uniform $L\mathfrak{Q}C^0$ for the corresponding class.

For a set of circuit families **C** be define $\mathcal{P}(\mathbf{C}), \mathcal{P}_1(\mathbf{C})$ equivalently.

Please note that a circuit with a single non-commutative gate that has all positions as input, cannot be realized by a uniformity language in depth 1, as the order of the inputs is determined by the labels of the gates. So in this case one would add an extra layer of gates $(k+1,1),\ldots,(k+1,n)$ such that $(k+1,i)$ is connected to $(1,n-i+1)$ and the gates with label $(k+1,\cdot)$ compute the identity, then we can connect the single non-commutative gate to the gates $(k+1,1),\ldots,(k+1,n)$.

So in this section when we talk about depth, it is the depth of the uniform circuit even if the uniformity language is highly non-uniform.

Lemma 4.26. *Let $\mathfrak{P}$ be a set of predicates, such that $\mathcal{P}(\mathfrak{P})$ is closed under inverse nearly length preserving morphisms,* **P** *be the corresponding predicate class, $\mathfrak{Q}$ be a set of quantifiers, and* **Q** *the set of typed quantifier semigroups for $\mathfrak{Q}$, where* **Q** *is closed under unit relaxation, then*

$$\mathcal{L}(\mathbf{P}\text{-}\textit{uniform } \mathfrak{Q}C^0_{lin} \textit{ of depth one}) = \mathcal{L}(\mathbf{Q} \boxdot \mathbf{P}).$$

Proof. Since this circuit has only one gate and this is labeled $(n,n,\ldots,n)$, we let $C'_l(j) = C((n,n,\ldots,n),(l,j,1,\ldots,1))$. It is clear that each C'_l can be recognized by a morphism $h_{C'_l}$ to $(S_P,\mathfrak{S}_P,\mathcal{E}_P)$. The circuit itself can have at most $|\Sigma|\cdot n$ wires, since there are not other possible connections. We can take the direct product $(S_P,\mathfrak{S}_P,\mathcal{E}_P)^\Sigma$ can get a morphism $h_{C'}$ and a type $\mathcal{S}_l$ such that $h(w_{y=j})\in\mathcal{S}_l$ iff $w_{y=j}\models C'_l(y)$.

Let $\Sigma=\{\sigma_1,\ldots,\sigma_m\}$ in the same way as the letters are numbered by the uniformity language. We will construct functions that add a true input to the gate iff there is a connection from the output gate to the input gate at position y querying the letter l. We let $f^1_\sigma : (S_P,\mathfrak{S}_P,\mathcal{E}_P)^\Sigma\times(S_P,\mathfrak{S}_P,\mathcal{E}_P)^\Sigma\to(S_G,\mathfrak{S}_G,\mathcal{E}_G)$ by $f^1_\sigma(p_1,p_2)=h_G(1)$ iff $p_1h_{C'}((a,x_l))p_2\in\mathcal{S}_{C'_l}$ and $f^1_\sigma(p_1,p_2)=1$ otherwise. Note that 1 exists since $\mathcal{E}_G=\mathcal{E}^*_G$. Similar we let $f^1_\sigma : (S_P,\mathfrak{S}_P,\mathcal{E}_P)^\Sigma\times(S_P,\mathfrak{S}_P,\mathcal{E}_P)^\Sigma\to(S_G,\mathfrak{S}_G,\mathcal{E}_G)$ by $f^1_\sigma(p_1,p_2)=h_G(0)$ iff $p_1h_{C'}((a,x_l))p_2\in\mathcal{S}_{C'_l}$ and $f^1_\sigma(p_1,p_2)=1$ otherwise.

Now at each position there is at most one letter so we let $f_{\sigma_l}=\prod_{i=1}^{l-1}f^0_{\sigma_i}f^1_{\sigma_l}\prod_{i=l+1}^{n}f^0_{\sigma_i}$. Also we let $h(\sigma_l)=(f_{\sigma_l},h_{C'}(a))$, then this morphism simulates with each letter the wires from this position to the output gate and add the correct number of true and false inputs. Hence there is a type $\mathcal{T}$ such that $L=h^{-1}(\mathcal{T})$. □

Lemma 4.27. *Let $\mathfrak{P}$ be a set of predicates, such that $\mathcal{P}(\mathfrak{P})$ is closed under inverse nearly length preserving morphisms,* **P** *be the corresponding predicate class, $\mathfrak{Q}$ be a set of quantifiers, and* **Q** *the set of typed quantifier semigroups for $\mathfrak{Q}$, where* **Q** *is closed under unit relaxation, then*

$$\mathcal{P}(\mathbf{P}\text{-}\textit{uniform } \mathfrak{Q}C^0_{lin} \textit{ of depth one}) = \mathcal{P}(\mathbf{Q} \boxdot \mathbf{P}).$$

Proof. Given a circuit C in X, that recognizes a language $L \subseteq \Sigma^+ \otimes \{x_1, \ldots, x_k\}$, then C consists of n^k gates, the output gates, each has at most linear fan-out, since there are only $|\Sigma| \cdot n$ possible predecessors. The construction of the previous lemma works.

For the other direction let $h : \Sigma^+ \otimes \{x_1, \ldots, x_k\} \to (S_G, \mathfrak{S}_G, \mathcal{E}_G) \boxdot (S_P, \mathfrak{S}_P, \mathcal{E}_P)$, that recognizes a language L. We let $h((\sigma, x)) = (f_{\sigma,x}, p_x)$, then by Lemma 3.35 we know $h(w_{\vec{x}=\vec{i}}) \in \mathcal{S}$, iff

$$\prod_{i=1}^{n} f_{w_i}(\pi_2(h(w_{<i})), \pi_2(h(w_{>i}))).$$

So we connect the i-th input of the gate so that to the input gates such that it input f_{w_i} to the gate. □

Definition 4.28 (Input Gate Substitution)**.** Let $(C_n)_n$ and $(C'_n)_n$ be a circuit families, where the gates (C_n) are labeled $(1, \vec{x})$ to $(k, \vec{x})$ and $(C'_n)_n$ are labeled $(1, \vec{x})$ to $(k', \vec{x})$. If the input gates of $(C_n)_n$ have values 1 to m, then we relabel the gates of $(C_n)_n$ by $(i, \vec{x}) \mapsto (k' - m + 1, \vec{x})$. Then $(C_n \cup C'_n)$ is the *input gate substitution* of $(C_n)_n$ by $(C'_n)_n$. We write $(C_n)_n \circ (C'_n)_n$ for the resulting circuit.

Any polynomial size circuit with linear fan-in gates can be build by input gate substitution of depth one circuits, if we start with the output gate and successifly add the predecessor gates. If we have a linear size circuit we can inductively build the circuit bottom up.

Lemma 4.29. *Let X, Y be a classes of circuits, and $\mathbf{V}, \mathbf{W}$ be classes of typed semigroups. If $\mathcal{P}(X) = \mathcal{P}(\mathbf{V})$ and $\mathcal{P}(Y) = \mathcal{P}(\mathbf{W})$ then $\mathcal{P}(X \circ Y) = \mathcal{P}(\mathbf{V} \boxdot \mathbf{W})$.*

Proof. This is equivalent to the proof of Theorem 3.40. □

Theorem 4.30. *Let $\mathfrak{P}$ be a set of predicates, such that $\mathcal{P}(\mathfrak{P})$ is closed under inverse nearly length preserving morphisms, $\mathbf{P}$ be the corresponding predicate class, $\mathfrak{Q}$ be a set of quantifiers, and $\mathbf{Q}$ the set of typed quantifier semigroups for $\mathfrak{Q}$, where $\mathbf{Q}$ is closed under unit relaxation, then*

- $\mathcal{P}(\text{sbpc}(\mathbf{Q} \cup \mathbf{P})) = \mathcal{P}(\mathfrak{P} - \textit{uniform } \mathfrak{Q}C^0_{lin})$,
- $\mathcal{P}(\text{wbpc}(\mathbf{Q} \boxdot \mathbf{P})) = \mathcal{P}(\mathfrak{P} - \textit{uniform } L\mathfrak{Q}C^0)$.

Proof. This proof is very similar to Theorem 4.14. We do this by induction on the depth of the circuit. For depth one circuit this is proven in Lemma 4.27.

For the first equation, if the depth is greater than one, than the circuit is a gate substitution of a depth one circuit by a circuit of smaller depth. The depth one circuit can be recognized by $\mathbf{Q} \boxdot \mathbf{P}$ by Lemma 4.27, and the circuit of smaller depth by a typed semigroup of $\text{sbpc}(\mathbf{Q} \cup \mathbf{P})$, hence the circuit can be recognized by $\text{sbpc}(\mathbf{Q} \cup \mathbf{P})$.

For the other direction we let $(S, \mathfrak{S}, \mathcal{E}) \in \text{sbpc}(\mathbf{Q} \cup \mathbf{P})$. Then we can assume $(S, \mathfrak{S}, \mathcal{E}) \preceq (\mathbf{Q} \boxdot \mathbf{P}) \boxdot (S', \mathfrak{S}', \mathcal{E}')$, where $(S', \mathfrak{S}', \mathcal{E}') \in \text{sbpc}(\mathbf{Q} \cup \mathbf{P})$. By Lemma 4.27 we know $\mathcal{P}(\mathbf{Q} \boxdot \mathbf{P}) \subseteq \mathcal{P}(\mathfrak{P} - \text{uniform } \mathfrak{Q}C^0_{lin} \text{ of depth one})$ and

by induction also $\mathcal{P}((S', \mathfrak{S}', \mathcal{E}')) \subseteq \mathcal{P}(\mathfrak{P}$ – uniform $\mathfrak{Q}C^0_{lin})$ so by Lemma 4.29 we have $\mathcal{P}((S, \mathfrak{S}, \mathcal{E})) \in \mathcal{P}(\mathfrak{P}$ – uniform $\mathfrak{Q}C^0_{lin}$ of depth one $\circ$ $\mathfrak{P}$ – uniform $\mathfrak{Q}C^0_{lin}) = \mathcal{P}(\mathfrak{P}$ – uniform $\mathfrak{Q}C^0_{lin})$.

Forthe second equation, if the depth is greater than one, than the circuit is a gate substitution of a smaller depth circuit by a circuit of depth one. The depth one circuit can be recognized by $\mathbf{Q} \boxdot \mathbf{P}$ by Lemma 4.27, and the circuit of smaller depth by a typed semigroup of wbpc($\mathbf{Q} \cup \mathbf{P}$), hence the circuit can be recognized by wbpc($\mathbf{Q} \cup \mathbf{P}$).

For the other direction we let $(S, \mathfrak{S}, \mathcal{E}) \in$ wbpc($\mathbf{Q} \cup \mathbf{P}$). Then we can assume $(S, \mathfrak{S}, \mathcal{E}) \preceq (\mathbf{Q} \boxdot \mathbf{P}) \boxdot (S', \mathfrak{S}', \mathcal{E}')$, where $(S', \mathfrak{S}', \mathcal{E}') \in$ sbpc($\mathbf{Q} \cup \mathbf{P}$). By Lemma 4.27 we know $\mathcal{P}(\mathbf{Q} \boxdot \mathbf{P}) \subseteq \mathcal{P}(\mathfrak{P}$ – uniform $\mathrm{L}\mathfrak{Q}\mathrm{C}^0$ of depth one) and by induction also $\mathcal{P}((S', \mathfrak{S}', \mathcal{E}')) \subseteq \mathcal{P}(\mathfrak{P}$ – uniform $\mathrm{L}\mathfrak{Q}\mathrm{C}^0)$ so by Lemma 4.29 we have $\mathcal{P}((S, \mathfrak{S}, \mathcal{E})) \in \mathcal{P}(\mathfrak{P}$ – uniform $\mathrm{L}\mathfrak{Q}\mathrm{C}^0$ $\circ$ $\mathfrak{P}$ – uniform $\mathrm{L}\mathfrak{Q}\mathrm{C}^0$ of depth one) $= \mathcal{P}(\mathfrak{P}$ – uniform $\mathrm{L}\mathfrak{Q}\mathrm{C}^0)$.

□

4.3 Logic-Algebra-Circuits

Now we can state a threeway connection between logic algebra and circuits.

Theorem 4.31. *Let $\mathfrak{P}$ be a set of predicates, such that $\mathcal{P}(\mathfrak{P})$ is closed under inverse nearly length preserving morphisms, $\mathbf{P}$ be the corresponding predicate class, $\mathfrak{Q}$ be a set of quantifiers, and $\mathbf{Q}$ the set of typed quantifier semigroups for $\mathfrak{Q}$, where $\mathbf{Q}$ is closed under unit relaxation, then*

1. *$L \in \mathfrak{P}$-uniform $\mathfrak{Q}C^0_{lin}$,*
2. *$L \in \mathcal{L}(\mathfrak{Q}[\mathfrak{P}])$,*
3. *$L \in \mathcal{L}(\mathrm{sbpc}(\mathbf{Q} \cup \mathbf{P}))$.*

Remark 4.32. The previous theorem could also be stated starting from an algebra: if we start with a weakly closed class of typed semigroups $\mathbf{P}$, and a weakly closed class of typed semigroups $\mathbf{Q}$ closed under unit relaxation, and let $\mathfrak{P}$ be the set of predicates recognizable by $\mathbf{P}$, and $\mathfrak{Q}$ the set of typed semigroup quantifiers for $\mathbf{Q}$, the previous theorem holds.

Similar starting from circuit theory: if we let $\mathfrak{P}$ be a set of predicates, such that $\mathcal{P}(\mathfrak{P})$ is closed under inverse nearly length preserving morphisms, $\mathbf{P}$ be the corresponding predicate class, $\mathfrak{Q}$ be a set of gate types, and $\mathbf{Q}$ the corresponding set of typed semigroups, where $\mathbf{Q}$ is closed under unit relaxation, then the previous theorem holds.

For a bounded number of variables we get:

Theorem 4.33. *Let $\mathfrak{P}$ be a set of predicates, such that $\mathcal{P}(\mathfrak{P})$ is closed under inverse nearly length preserving morphisms, $\mathbf{P}$ be the corresponding predicate class, $\mathfrak{Q}$ be a*

set of quantifiers, and $\mathbf{Q}$ *the set of typed quantifier semigroups for* $\mathfrak{Q}$*, where* $\mathbf{Q}$ *is closed under unit relaxation, then*

1. $L \in \mathfrak{P}$*-uniform* $L\mathfrak{Q}C^0$,
2. $L \in \mathfrak{L}(\mathfrak{Q}_2[\mathfrak{P}])$,
3. $L \in \mathfrak{L}(\mathrm{wbpc}(\mathbf{Q} \boxdot \mathbf{P}))$.

Remark 4.34. The statement of the previous Remark 4.32 also holds for this theorem.

Since these theorems are stated in a very general way they have many implications. For the consequences in the usual logic and circuit classes considered see Figures 4.3 and 4.4 for the polynomial and linear case.

Circuits	Logic	Algebra (via morphism)
$CC^0[q]$	(MOD_q)[arb]	$\mathrm{sbpc}(\mathbb{Z}_q \cup \mathbf{P}_{\mathbf{arb-un}})$
AC^0	FO[arb]	$\mathrm{sbpc}(U_1 \cup \mathbf{P}_{\mathbf{arb-un}})$
$ACC^0[q]$	$(FO+MOD_q)$[arb]	$\mathrm{sbpc}(U_1 \cup \mathbb{Z}_q \cup \mathbf{P}_{\mathbf{arb-un}})$
TC^0	MAJ[arb]	$\mathrm{sbpc}((\mathbb{Z},\mathbb{Z}^+) \cup \mathbf{P}_{\mathbf{arb-un}})$
NC^1	FO + G[arb]	$\mathrm{sbpc}(\mathbf{Fin} \cup \mathbf{P}_{\mathbf{arb-un}})$
FO[<]-uniform $CC^0[q]$	(MOD_q)[<]	$\mathrm{sbpc}(\mathbb{Z}_q \cup \mathbf{P}_{<}) = \mathrm{sbpc}_<(Z_q)$
FO[<]-uniform AC^0	FO[<]	$\mathrm{sbpc}(U_1 \cup \mathbf{P}_{<}) = \mathrm{sbpc}_<(U_1)$
FO[<]-uniform $ACC^0[q]$	$(FO+MOD_q)$[<]	$\mathrm{sbpc}(U_1 \cup \mathbb{Z}_q \cup \mathbf{P}_{<}) =$ $= \mathrm{sbpc}_<(U_1 \cup \mathbb{Z}_q)$
FO[<]-uniform TC^0	MAJ[<]	$\mathrm{sbpc}((\mathbb{Z},\mathbb{Z}^+) \cup \mathbf{P}_{<}) = \mathrm{sbpc}_<(\mathbb{Z})$

Figure 4.3: *Relations between circuits, logic and typed semigroups (polynomial case)*

Circuits	Logic	Algebra (via morphism)
lin-$CC^0[q]$	$(MOD_q)_2$[arb]	$\mathrm{wbpc}(\mathbb{Z}_q \boxdot \mathrm{wc}(\mathbf{P}_{\mathbf{arb-un}}))$
LC^0	FO_2[arb]	$\mathrm{wbpc}(U_1 \boxdot \mathrm{wc}(\mathbf{P}_{\mathbf{arb-un}}))$
$LCC^0[q]$	$(FO+MOD_q)_2$[arb]	$\mathrm{wbpc}((U_1 \times \mathbb{Z}_q) \boxdot \mathrm{wc}(\mathbf{P}_{\mathbf{arb-un}}))$
LTC^0	$\widehat{MAJ}_2$[arb]	$\mathrm{wbpc}((\mathbb{Z},\mathbb{Z}^+) \boxdot \mathrm{wc}(\mathbf{P}_{\mathbf{arb-un}}))$
FO[<]-uniform lin-$CC^0[q]$	$(MOD_q)_2$[<]	$\mathrm{wbpc}(\mathbb{Z}_q \boxdot \mathrm{wc}(\mathbf{P}_{<})) = \mathrm{wbpc}_<(\mathbb{Z}_q)$
FO[<]-uniform LC^0	FO_2[<]	$\mathrm{wbpc}(U_1 \boxdot \mathrm{wc}(\mathbf{P}_{<})) = \mathrm{wbpc}_<(U_1)$
FO[<]-uniform $LCC^0[q]$	$(FO+MOD_q)_2$[<]	$\mathrm{wbpc}((U_1 \times \mathbb{Z}_q) \boxdot \mathrm{wc}(\mathbf{P}_{<})) =$ $= \mathrm{wbpc}_<(U_1 \times Z_q)$
FO[<]-uniform LTC^0	$\widehat{MAJ}_2$[<]	$\mathrm{wbpc}((\mathbb{Z},\mathbb{Z}^+) \boxdot \mathrm{wc}(\mathbf{P}_{<}))$

Figure 4.4: *Relations between circuits, logic and typed semigroups (linear case)*

4.4 Summary

In this chapter we consistently used the block product principle (Theorem 3.40) to prove one of our main results: a tight connection between logic, algebra and circuits (Theorems 4.31, 4.33).

For this purpose we needed to define the basic building blocks. For logic we defined a typed quantifier semigroup (Definition 4.1) for each quantifier, and conversely a typed semigroup quantifier (Definition 4.3) for each typed semigroup (where the units generate the typed semigroup). For the predicates used in logic in Definition 4.8 we defined an algebraic equivalent and showed that for any predicate set with minimal closure properties, there is an algebraic equivalent.

For circuits we proceeded in a similar way and defined a typed gate semigroup (Definition 4.17) showing that circuits are equivalent to programs over weakly blocked typed gate semigroups. Striving to characterize the circuits by morphisms into the typed semigroups and not by programs we introduced an uniformity language (Definition 4.22). We then used the block product principle adopting it to circuits (Theorem 4.29) and gave a morphic characterization of circuit families.

4.5 Further Research

Here we will survey some other natural restrictions for logic and observe their effect on the algebraic correspondence. One obvious restriction is bounding the quantifier depth to some constant d. Since by Lemma 4.13 we have a correspondence to depth one formulas this results in a typed semigroup class starting with the building blocks $(S_Q, \mathfrak{S}_Q, \mathcal{E}_Q) \boxdot (S_P, \mathfrak{S}_P, \mathcal{E}_P)$ and limiting the block depth to d.

Another restriction already studied is restricting the number of variables to some constant r. While the classes of logic FO[<], FO + MOD[<] containing only regular languages are known to have a bound of three variables, for other classes of logic it is still unknown if there is a bound for the variables needed even when considering only regular languages.

We will show that $\mathrm{MAJ}_2[<] \cap \mathrm{REG} \subsetneq \mathrm{MAJ}_3[<] \cap \mathrm{REG}$, and we know that $\mathrm{MAJ}_4[<]$ contains all languages in $\mathrm{FO}+\mathrm{MOD}[<]$. The question is thus whether $\mathrm{MAJ}_3[<] \cap \mathrm{REG}$ is a proper subset of $\subsetneq \mathrm{MAJ}_4[<] \cap \mathrm{REG}$ or whether they are equal, and analogously $\mathrm{MAJ}_4[<] \cap \mathrm{REG} \subsetneq \mathrm{MAJ}_5[<] \cap \mathrm{REG}$, the latter already implying that $\mathrm{TC}^0 = \mathrm{NC}^1$. So proving the number of variables in majority logic to be a true hierarchy can be expected to be a profound task.

An algebraic characterization of logic bounded to r variables could also be useful. A restriction in logic to r variables translates to block products of $(S_Q, \mathfrak{S}_Q, \mathcal{E}_Q) \boxdot (S_P, \mathfrak{S}_P, \mathcal{E}_P)$, where the constants c of the block product have the property that $\pi_2^r(c) = 1$. It is easy to see that this is equivalent to restricting formulas such that every bound variable occurs only $r-1$ quantifiers inside the quantifier that it is bound by, which is equivalent to the power of r variables. Observe that for $r = 1$ the block

product can be replaced by the direct product and for $r = 2$ the block product can be transformed to its weak version (Proposition 3.42).

This algebraic characterization on the other hand corresponds to circuit families with at most n^{r-1} gates, and regular language always have circuits of size $n^{1+\epsilon}$ [CFL85]. This construction however requires adequate predicates on the side of logic, and therefore does not answer any of the hierarchy questions mentioned above. The eligibility of such a characterization of course depends on its future usefulness in helping to find proofs.

Chapter 5

Majority Logic

In this chapter we take a close look at majority logic. It is known the majority quantifier can simulate the counting quantifier [Lan04] and the counting quantifier can simulate the majority quantifier. There are many other quantifiers that have similar properties in this chapter. We consider different quantifiers and predicates and compare their power in recognizing languages. Since the cases of an unbounded number of variables and two variables have to be differentiated we examine the two cases separately. For the two variable case we will see that the majority quantifier is not as robust as desired. We therefore introduce the extended majority quantifier.

As it turns out the majority quantifier with an arbitrary set of predicates including the order predicate is a variety. For the two variables case we show the extended majority quantifier with the order predicate is also a variety. Moreover we present a very basic algebraic characterization for these varieties.

5.1 Several Counting Quantifiers

We will now recall the definition of some counting quantifiers. According to the definition in the preliminaries we call a quantifier a normal quantifier if it quantifies over one subformula and an extended quantifier if it quantifiers over more than one formula. We will examined the normal and the extended version of these quantifiers separately, where the extended quantifier have not been used before in most cases.

First we define the finite counting quantifier $\exists^{=c}\ x\ \varphi$ for a constant $c \in \mathbb{N}$ by $w \models \exists^{=c}\ x\ \varphi$ iff $|\{i \mid w_{x=i} \models \varphi\}| = c$. It is known that this quantifier has not much extended power over the existential quantifier. We have also a threshold version of this quantifier $\exists^{>c}\ x\ \varphi$ requires that there are more than c positions i for x, such that $w_{x=i} \models \varphi$. The extended versions of these quantifiers coincide with the usual quantifiers because we are always closed under Boolean combinations.

Similar we can replace the constant c by a function $f : \mathbb{N} \to \mathbb{N}$, such that the value depends on the length of the input word. This yields the quantifiers $\exists^{=f}$ and $\exists^{>f}$.

$\exists^{=c}\ x\ \varphi$	constant counting quantifier	exactly c positions
$\exists^{>c}\ x\ \varphi$	constant threshold quantifier	more than c positions
$\exists^{=f}\ x\ \varphi$	threshold counting quantifier	exactly $f(\|w\|)$ positions
$\exists^{>f}\ x\ \varphi$	threshold quantifier $f(n)$	more than $f(\|w\|)$ positions
Maj $x\ \varphi$	majority quantifier	more than half positions
$\exists^{=y}\ x\ \varphi$	counting quantifier	exactly y positions
$\exists^{>y}\ x\ \varphi$	threshold quantifier	more than y positions
$\exists^{\in S}\ x\ \varphi$	The number of positions is in the set $S \subseteq \mathbb{N}$	
$\exists^{\in S_n}\ x\ \varphi$	The number of positions is in the set $S_n \subseteq \mathbb{N}$	

Figure 5.1: *List of several counting quantifiers*

Since f is not bounded these quantifiers do not coincide in general with the extended quantifiers, hence we define $\widehat{\exists}^{=f}$ and $\widehat{\exists}^{>f}$ as the corresponding logic classes with the extended versions of these quantifiers. The majority quantifier Maj $x\ \varphi$ is a special case of this for $f(n) = \lfloor n/2 \rfloor$. We define the logic classes MAJ and $\widehat{\text{MAJ}}$ for the normal and extended version of this quantifier.

Instead of counting up to a number we can count the number of positions and ask whether they are in a given set $S \subseteq \mathbb{N}$ or not. These quantifiers correspond to the natural numbers in algebra and have the quantifier semigroup $(\mathbb{N}, S, \{0,1\})$; a typical example of such a quantifier is the square quantifier, i.e. S is the set of squares $\mathbb{S}$. We get a quantifier $\exists^{\in S}\ x\ \varphi$, defined by $w \models \exists^{\in S}\ x\ \varphi$ iff $|\{i \mid w_{x=i} \models \varphi\}| \in S$. For the extended version we let $\exists^{\in S}\ x\ \langle \varphi_1, \ldots, \varphi_k \rangle$ be defined by $w \models \exists^{\in S}\ x\ \langle \varphi_1, \ldots, \varphi_k \rangle$ iff $|\{(i,l) \mid w_{x=i} \models \varphi_l\}| \in S$. This defines the logic classes $\exists^{\in S}$ and $\widehat{\exists}^{\in S}$.

The quantifier above can also be defined for a set that depends on the input length of w. With an equivalent definition we get the logic classes $\exists^{\in S_n}$ and $\widehat{\exists}^{\in S_n}$.

The counting quantifier $\exists^{=y}\ x\ \varphi$, defined by $w_{y=j} \models \exists^{=y}\ x\ \varphi$ iff $|\{i \mid w_{x=i,y=j} \models \varphi\}| = j$, counts if there are exactly the value of y many positions for x where φ is true. Since this quantifier requires a variable y that is not bound by the quantifier itself, we always add the regular existential quantifier $\exists$ and obtain the logic class $\exists + \exists^{=y}$. The extended version of this quantifier allows multiple subformulas $\varphi_1, \ldots, \varphi_k$: $\exists^{=y}\ x\ \langle \varphi_1, \ldots, \varphi_k \rangle$, defined by $w_{y=j} \models \exists^{=y}\ x\ \langle \varphi_1, \ldots, \varphi_k \rangle$ iff $|\{(i,l) \mid w_{x=i,y=j} \models \varphi_l\}| = j$. This quantifier together with the existential quantifier forms the logic class $\exists + \widehat{\exists}^{=y}$. Also because of the quantifier make use of two variables this quantifier will not be useful if the number of variables if bound to two.

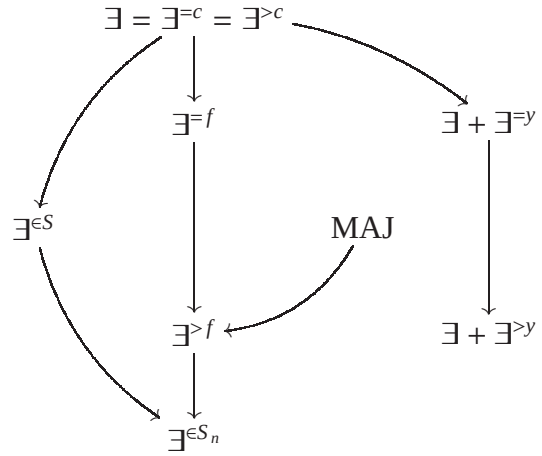
Similar we have a threshold quantifier $\exists^{>y}\ x\ \varphi$, equivalent to the quantifier above that requires more than the value of y positions for x such that φ is true. The corresponding logic class is $\exists + \exists^{>y}$ and with the extended quantifier $\exists + \widehat{\exists}^{>y}$.

We defined quite a lot counting quantifiers, so we list them in the following Figure 5.1.

5.1.1 Unbounded number of variables

We now compare the power of the previous introduced quantifier in the presence of different predicate sets. In the case of using the equality predicate solely, we have only trivial inclusions.

Proposition 5.1. *For the predicate set* $\{=\}$ *and an unbounded number of variables, the following inclusions hold, where* $A \to B$ *means that A can be simulated by B:*



The extended quantifiers can simular the normal quantifiers and the same relations among themselves as in the normal case. Additionally the majority quantifier can simulate the existential quantifier.

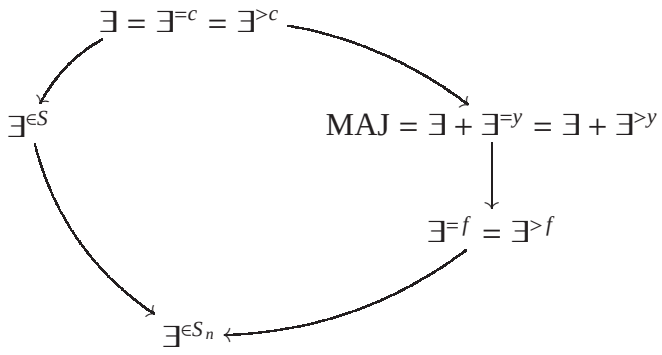
Proof. We show only one inclusion, the others are trivial or similar.

$$\exists^{=c}\ x\ \varphi(x) = \exists^{>c-1}\ x\ \varphi(x) \wedge \neg\exists^{>c}\ x\ \varphi(x).$$

□

If we allow the order predicate the picture becomes more simple. We can split the set of quantifiers in uniform quantifiers which are either first order or counting quantifiers and the non-uniform quantifiers which split in three classes.

Proposition 5.2. *For the predicate set* $\{<\}$ *and an unbounded number of variables, the following inclusions hold, where* $A \to B$ *means that A can be simulated by B:*



The extended version and the normal version of the quantifier are always equally powerful.

Proof.

$$\exists^{>c}\ x\ \varphi(x) = \exists^{=1}\ y\ (\exists^{=c+1}\ x\ x \leq y \wedge \varphi(x)) \wedge \neg(\exists^{=c+1}\ x\ x < y \wedge \varphi(x)).$$

By [Lan04, Corollary 3.3] we know $\exists$ can be simulated by MAJ[<], as well as counting. On the other hand there is a formula $\varphi(x)$ in $\exists + \exists^{>y}$ that is true if $i = \lceil n/2 \rceil$, where i is the value of x. □

If we also use the unary arbitrary predicates the non-uniform quantifiers can be simulated by the uniform quantifiers, which gives a clear picture.

Proposition 5.3. *For the predicate set $\{<, un-arb\}$ and an unbounded number of variables, the following inclusions hold, where $A \to B$ means that A can be simulated by B:*

$$\begin{array}{c} \exists = \exists^{=c} = \exists^{>c} \\ \downarrow \\ \exists^{\in S} \\ \downarrow \\ \mathrm{MAJ} = \exists^{=f} = \exists^{>f} = \exists + \exists^{=y} = \exists + \exists^{>y} = \exists^{\in S_n} \end{array}$$

The extended version and the normal version of the quantifier are always equally powerful.

Proof. We choose a unary predicate $\mathfrak{p}$ that is true for values in S, then

$$\exists^{\in S}\ x\ \varphi(x) = \exists\ y\ \mathfrak{p}(y) \wedge \exists^{=y}\ x\ \varphi(x).$$

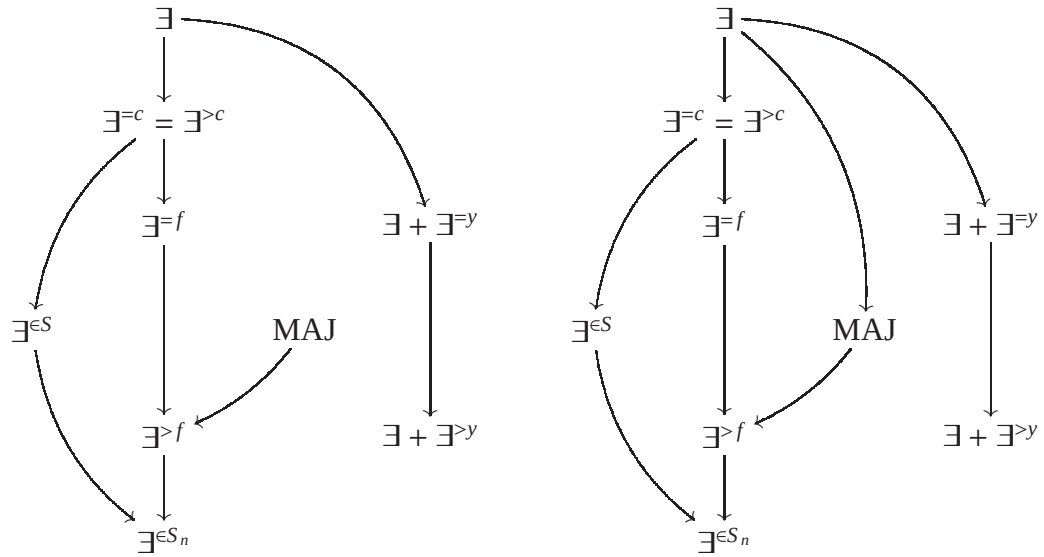
□

5.1.2 Two variable case

We continue to examine the case of two variables again for different predicate sets. Naturally, it turns out that there are less equivalences than in the unbounded number of variables case.

Proposition 5.4. *For the predicate set $\{=\}$ and two variables, where $A \to B$ means*

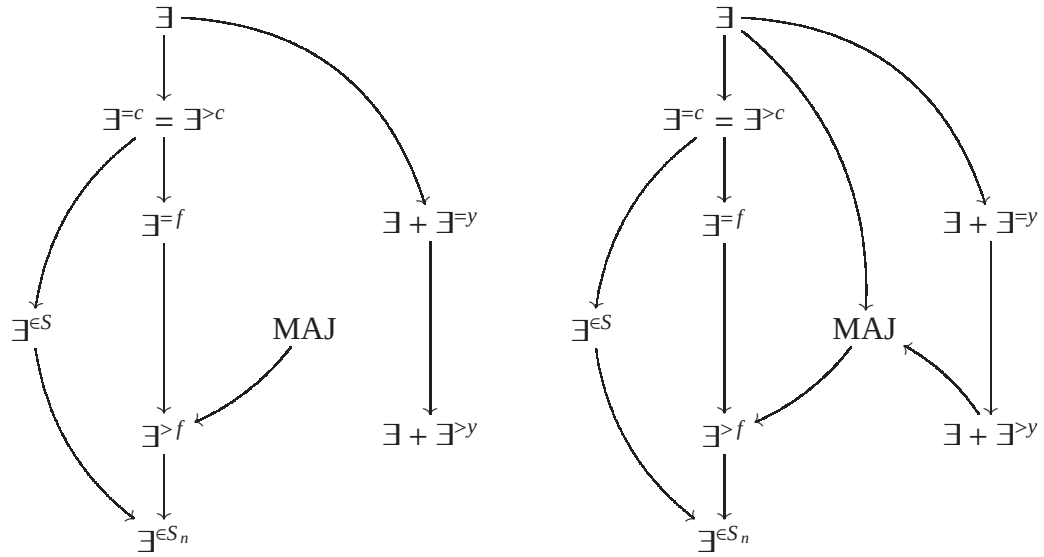
that A can be simulated by B:



Proof. □

Taking a look at the order predicate we obtain different pictures in the case of the normal and the extended quantifiers, in the extended case the majority quantifier can simulate most other quantifiers.

Proposition 5.5. *For the predicate set* $\{<\}$ *and two variables, where* $A \to B$ *means that A can be simulated by B:*



Proof. □

As we see in the last diagram, if we are interested in the uniform quantifiers, the extended majority quantifier can simulate all other uniform quantifiers. In the

following we always include the order predicate when we use the majority quantifier in the unbounded case and the extended majority quantifier in the two variables case, which seems to be the most interesting case.

5.2 Several Predicate Sets

In this section we examine the power of the majority quantifier in terms of the numerical predicates that can be simulated.

Lemma 5.6. *The following equalities hold:*

- $\mathrm{MAJ}[<] = \mathrm{MAJ}[<, +]$
- $\mathrm{MAJ}[<, \mathrm{square}] = \mathrm{MAJ}[<, +, *]$

Proof. This is proven in [Lan04, Theorem 4.1]. □

On the other hand we have some nonexpressibility results.

Lemma 5.7. *We have the following inequalities in majority logic:*

- $\mathrm{MAJ}[+] \neq \mathrm{MAJ}[<, +]$
- $\mathrm{MAJ}[<, +] \neq \mathrm{MAJ}[<, +, *]$

Proof. The first inequality is proven in [Lan04], and the second is a consequence of [Ruh99] and independently of [LMSV01]. □

5.3 Varieties

In this section we show that for all sets of predicates $\mathfrak{P}$ the languages expressed by the majority logic $\mathrm{MAJ}[<, \mathfrak{P}]$ form a variety.

This is very different compared to less powerful logic classes that cannot count. For example the modulo predicates are closed under inverse morphisms, but the languages recognized by formulas of $\mathrm{FO}[<, \mathrm{mod}]$ are not closed under inverse morphisms. We can for example recognize the language L of all words of even length in $\{a, b\}^+$ by a formula of $\mathrm{FO}[<, \mathrm{mod}]$. The morphism $h : \{a, b\}^+ \to \{a, b\}^+$ that maps $a \mapsto a$ and $b \mapsto bb$, can be used to define $L' = h^{-1}(L)$. Then L' is the language of an even number of a's, that is not recognized by $\mathrm{FO}[<, \mathrm{mod}]$. So $\mathrm{FO}[<, \mathrm{mod}]$ is not closed under inverse morphisms and hence not a variety.

For logic classes that can count the situation is different as seen in the following theorem:

Theorem 5.8. *For every set of predicates $\mathfrak{P}$ closed under inverse morphisms and containing order the class $\mathfrak{L}(\mathrm{MAJ}[\mathfrak{P}])$ is a variety of languages.*

Proof. Please note that $\mathcal{L}(\mathrm{MAJ}[\mathfrak{P}]) = \mathcal{L}(\widehat{\mathrm{MAJ}}[\mathfrak{P}])$ since the predicates contain order. Hence we may use the extended majority quantifier in our proof.

We need to show the following closure properties:

1. The languages in MAJ[$\mathfrak{P}$] are closed under Boolean operations.
2. If $L \in \mathrm{MAJ}[\mathfrak{P}]$, $L \subseteq \Sigma^+$ and $h : (\Sigma')^+ \to \Sigma^+$ is a morphism then $L' = h^{-1}(L) \subseteq (\Sigma')^+$ is in MAJ[$\mathfrak{P}$].
3. If $L \in \mathrm{MAJ}[\mathfrak{P}]$, $L \subseteq \Sigma^+$ then $u^{-1}Lv^{-1} \in \mathrm{MAJ}[\mathfrak{P}]$, where $u^{-1}Lv^{-1} = \{w \mid uwv \in L\}$.

(1) Let $L_1, L_2 \in \mathcal{L}(\mathrm{MAJ}[\mathfrak{P}])$, then there are formulas $\varphi_1, \varphi_2 \in \mathrm{MAJ}[\mathfrak{P}]$ such that $L_{\varphi_1} = L_1$ and $L_{\varphi_2} = L_2$. It is easy to see that $L_1 \cap L_2 = L_{\varphi_1 \wedge \varphi_2}$, $L_1 \cup L_2 = L_{\varphi_1 \vee \varphi_2}$ and $\overline{L_1} = L_{\neg\varphi_1}$, hence $\mathcal{L}(\mathrm{MAJ}[\mathfrak{P}])$ is closed under Boolean operations.
(2) Let $L \in \mathcal{L}(\mathrm{MAJ}[\mathfrak{P}])$, then there is a formula φ such that $L_\varphi = L$. Given a morphism $h : \Sigma'^+ \to \Sigma^+$, we need to show that $L' = h^{-1}(L)$ is in $\mathcal{L}(\mathrm{MAJ}[\mathfrak{P}])$. Let $c = \max_{\sigma' \in \Sigma'} |h(\sigma')|$ be maximal length a letter is mapped to.

We will use induction over the subformulas of φ to proof this. Since the subformulas contain free variables we extend the morphism in a special way to free variables:

Given a word $w'_{x_1=i_1,\ldots,x_k=i_k} \in \Sigma'^+ \otimes \{x_1, \ldots, x_k\}$, and a vector $v \in \{0, \ldots, c-1\}^k$, we let

$$h_v(w'_{x_1=i_1,\ldots,x_k=i_k}) = h(w')_{x_1=|h(w'_{<i_1})|+v_1,\ldots,x_k=|h(w'_{<i_k})|+v_k}.$$

So we map the variable x_j pointing to the letter w_{i_j} to the position under the morphism, where v_i is the offset since $h(w'_{i_j})$ might be a word (not only a single letter). Please note that this construction maps a sentence to a sentence (not a set of sentences).

We call v the offset vector. It is easy to see that there are formulas δ_v in MAJ[$\mathfrak{P}$], such that $w'_{x_1=i_1,\ldots,x_k=i_k} \models \delta_v$ if $v_j < |h(i_j)|$ for all j. We call an offset vector v valid for $w'_{x_1,\ldots,x_k}$ iff $w'_{x_1,\ldots,x_k} \models \delta$.

We will define recursively a map that maps any formula ψ with free variables $X_1, \ldots, X_k$ to a set of formulas ψ'_v with $v \in \{0, \ldots, c-1\}^k$, such that for each word $w_{x_1,\ldots x_k} \in \Sigma'^+ \otimes \{x_1, \ldots, x_k\}$ and valid offset vector v:

$$w_{x_1=i_1,\ldots,x_k=i_k} \models \psi'_v$$

iff

$$h_v(w_{x_1=i_1,\ldots,x_k=i_k}) \models \psi.$$

(i) Assume $\psi = \mathfrak{c}_\sigma(x_1)$. We let $\psi'_v = \bigvee_{\sigma' \in \Sigma', h(\sigma')_{v_1} = \sigma} \mathfrak{c}_{\sigma'}(x_1)$.
(ii) Assume $\psi = \mathfrak{p}(x_1, \ldots, x_k)$. Since $\mathfrak{p} \in \mathfrak{P}$ and $\mathfrak{P}$ are closed under inverse morphisms there are predicates $\mathfrak{p}'_{v'}$ for $v' \in \{0, \ldots, c-1\}^k$ such that $\mathfrak{p}'_{v'}(y_1, \ldots, y_k)$ is true iff $\mathfrak{p}(y_1 \cdot c + v'_1, \ldots, y_1 \cdot c + v'_k)$ is true. We cannot use $\mathfrak{p}'$ directly, so we let

$$\psi'_v = \bigvee_{v'} \exists\, y_1 \cdots \exists\, y_k \quad y_1 \cdot c + v_1 = |h(w_{<x_1})| + v'_1 \wedge \cdots \wedge y_k \cdot c + v_k = |h(w_{<x_k})| + v'_k \wedge \mathfrak{p}'_{v'}(y_1, \ldots, y_k)$$

It is easy to see that the arithmetic operations can be performed by an extended majority quantifier using the order predicate.
(iii) Assume $\psi = \text{Maj } x \langle \psi^1, \ldots, \psi^m \rangle$. Since the majority quantifier bounds a variable, we need to ensure that only valid offset vectors are considered when evaluating the subformulas ψ_j. Please note that the offset vector is valid depending on the position of x.

We define $\text{Maj}^{\delta_v}\ x \langle \psi^l \rangle_l$ to equal Maj $x \langle \psi'^l \wedge \delta_v, \psi'^l \vee \neg\delta_v \rangle_l$. Then if for the position of x the offset vector is invalid $\psi'_l \wedge \delta_v$ evaluates to true and $\psi'_l \vee \neg\delta_v$ evaluated to false, hence this position is ignored by the majority quantifier. In the case that the offset vector is valid both terms evaluate to ψ'_l.

Now we can define ψ'_v. We let $\psi'_v = \text{Maj}^{\delta_v}\ x \langle {\psi'}^1_{v,b}, \ldots, {\psi'}^m_{v,b} \rangle_{b=0,\ldots,c-1}$.
(3) This construction is similar to the construction of (2). We use an vector $v \in \{0, 1, 2\}$, depending wether we are at the first, a middle or at the last position and construct to each formula ψ three versions $\psi'_0, \psi'_1, \psi'_2$. □

The proof is also valid for two variables, if we have no predicates:

Theorem 5.9. *The class* $\mathfrak{L}(\widehat{\text{MAJ}}_2[<])$ *is a variety.*

Actually the results follows for any logic class that can count where the quantifiers and predicates are closed under inverse morphism. It is much easier to check if a quantifier or a predicate is closed under inverse morphisms than to check this for the whole class of logic formulas.

Remark 5.10. The languages in the logic class MAJ + $\mathfrak{Q}[\mathfrak{P}]$ form a variety, if the inverse morphic closure of the quantifiers $\mathfrak{Q}$ and the predicates $\mathfrak{P}$ is in MAJ + $\mathfrak{Q}[\mathfrak{P}]$.

Also we get a consequence that somehow reduces hopes to prove Crane Beach results for majority logic.

Corollary 5.11. $L \in \text{MAJ}[\mathfrak{P}]$ *then L with a neutral letter is also in MAJ*$[\mathfrak{P}]$.

Proof. This follows from the proof of Theorem 5.8. □

This gives an alternative proof of theorem 6.3(b) in [BIL+05].

5.4 Algebraic Characterization

With the relative block product we can give a cleaner characterization for majority logic.

Theorem 5.12. *The following are equations are true:*

- $\text{sbpc}_<((\mathbb{Z}, \mathbb{Z}^+)) = \text{MAJ}[<]$,
- $\text{sbpc}_<((\mathbb{Z}, \{\mathbb{Z}^+, \mathbb{S}\})) = \text{MAJ}[<, +, *]$

Proof. It clear that the $\subseteq$ relation is true in both cases since the relative majority quantifier can be simulated by the majority quantifier ([Lan04]). For the other direction we need to show that we can express $<$, by Lemma 4.10 any non-commutative monoid suffices, hence $\mathbb{Z} \boxtimes \mathbb{Z}$, recognizes the order predicate and by Theorem 4.14 we have $\mathrm{sbpc}_<((\mathbb{Z}, \mathbb{Z}^+)) \supseteq \mathrm{MAJ}[<]$.

For the second equation we need to show that we can express the square predicate. But it is easy to see that $(\mathbb{Z}, \mathbb{Z}^+) \boxtimes (\mathbb{Z}, \mathbb{S})$ can recognize this predicate and again by Theorem 4.14 we have $\mathrm{sbpc}_<((\mathbb{Z}, \mathbb{Z}^+, \mathbb{S})) \supseteq \mathrm{MAJ}[<, +, *]$ □

This gives us the following result in circuit theory. Please note that we use morphism into semigroups here to describe the languages recognizable by circuit family and not programs as usual.

Corollary 5.13. $\mathrm{DLOGTIME} - \mathrm{TC}^0 = \mathrm{MAJ}[<, +, *] = \mathrm{sbpc}_<(\mathbb{Z}, \{\mathbb{Z}^+, \mathbb{S}\}, \mathbb{Z})$.

5.5 Summary

We gave an overview of majority logic with a strong focus on expressiveness and algebra. Starting with several different quantifiers (see Figure 5.1) of similar expressive power, we presented their relative power in the presence of different predicate sets.

We discovered that for an unbound number of variables the majority quantifier in the presence of the order predicate is as strong as any "uniform" linear counting quantifier, and in the presence of arbitrary predicates it is as strong as any of the linear counting quantifier examined here. In the case of two variables there seem to be differences for the normal quantifiers, even in the presence of arbitrary predicates. Together with the order predicate the extended majority quantifier has an excelling power relative to all "uniform" quantifiers.

We therefore concluded we can concentrate our attention to the majority quantifier for the case of an unbounded number of variables, and to the extended majority quantifier for the case of two variables. For these two cases we examined which predicates can be simulated by others and which cannot.

We then showed that contrary to other classes of logic, the majority quantifier with most of the commonly considered predicate sets forms a variety, and that this also holds for the extended majority quantifier with the order predicate in the two variables case. This finally led to a very basic algebraic characterization of these varieties, which is interesting in its own right.

5.6 Further Research

In this chapter many questions remain open. We left unsettled many questions of the inequalities of majority logic with the different predicate sets considered. Using typed semigroups we can construct an algebraic characterization for each of these classes of

logic and in this way try to give an algebraic proof for some of the inequalities. Also, we touch on the Crane Beach conjecture or on its generalization the Uniformity Duality Property of [MTV08], the latter conjecturing MAJ[arb]∩CFL = MAJ[<]∩CFL, which implies MAJ[arb] ∩ REG = MAJ[<] ∩ REG. If this conjecture proves correct it might be the reason for it being hard to show that L_{A_5} is in MAJ[<] or the contrary.

Chapter 6

Regular languages in $\widehat{\mathrm{MAJ}}_2[<]$

The goal of this chapter is finding an upper and a lower bound for the finite syntactic semigroups of the regular languages in $\widehat{\mathrm{MAJ}}_2[<]$. In order to achieve this aim we use typed semigroups and the characterization of $\widehat{\mathrm{MAJ}}_2[<]$ of the previous chapter.

We devote the first section to a geometric interpretation of $\widehat{\mathrm{MAJ}}_2[<]$ formulas to gain insight into the algebraic proof techniques used in the following section. We then apply these proof techniques to find an upper bound. The lower bound is proven by a construction of $\widehat{\mathrm{MAJ}}_2[<]$ formulas for the finite semigroups contained in this bound.

6.1 Geometry

In what follows we consider the infinite grid graph on $\mathbb{N} \times \mathbb{N}$ as in Figure 6.1. The vertices of the graph are all points in $\mathbb{N} \times \mathbb{N}$ and there is an edge between two points (x_1, y_1) and (x_2, y_2) iff their distance is one, i.e. $|x_1 - x_2| + |y_1 - y_2| = 1$. There is one distinguished vertex $(0, 0)$ that we call the origin of the graph. For two points we write $(x_1, y_1) \leq (x_2, y_2)$ if $x_1 \leq x_2 \wedge y_1 \leq y_2$.

We call a path $((x_0, y_0), \ldots, (x_n, y_n))$ in this graph positive iff $(x_{i-1}, y_{i-1}) \leq (x_i, y_i)$

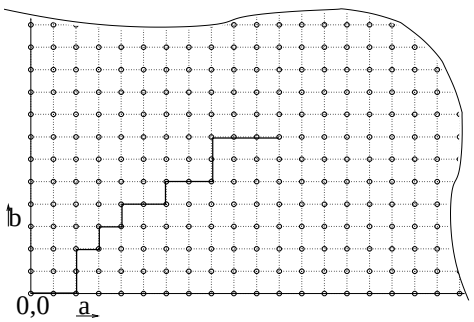


Figure 6.1: *The path for the word aabbababaabaabbaaa*

for all $i = 1, \ldots, n$. There is an obvious correspondence between all (non trivial) positive paths starting in the origin and the words in $\{a, b\}^+$, defined by the following correspondence: $w_1 \ldots w_n$ corresponds to the path $((x_0, y_0), \ldots, (x_n, y_n))$ iff $w_i = a \iff x_i - x_{i-1} = 1$ and $w_i = b \iff y_i - y_{i-1} = 1$. Informally speaking the path makes a step to the right for each a, and a step upwards for each b, see Figure 6.1 for an example.

The observation above is only essential because we limit our attention to words with a fixed Parikh vector $\#(w) = (n_a, n_b) \in \mathbb{N} \times \mathbb{N}$. Then the words w with $\#(w) = (n_a, n_b)$ correspond to the positive paths between the origin and the vertex (n_a, n_b). So we can focus to the subgraph of all points $(x, y) \leq (n_a, n_b)$.

For the set of words with fixed Parikh vector we will now show that $\widehat{\text{MAJ}}_1[<]$ formulas with one free variable can be interpreted as half planes when we embed the graph in the Euclidean plane $\mathbb{R} \times \mathbb{R}$. Similar to the correspondence above a word $w_{y=j}$ corresponds to the path of w with a marker on the point (x_j, y_j). Below we will show that all information about a word $w_{y=j}$ inferable from a majority formula depends on the point (x_j, y_j), or more specifically it depends on in which set of the half planes the point is for a fixed set of half planes.

We will demonstrate the geometric interpretation on an example. Let φ be the formula Maj x $((\mathfrak{c}_a(x) \wedge x \leq y) \vee (\mathfrak{c}_b(x) \wedge x > y))$, then $w_{y=j} \models \varphi$ iff $\#_a(w_{\leq j}) + \#_b(w_{>j}) > \#_a(w_{>j}) + \#_b(w_{\leq j}))$. This can be reformulated as

$$\begin{aligned}
&\#_a(w_{\leq j}) + \#_b(w_{>j}) - \#_a(w_{>j}) - \#_b(w_{\leq j}) > 0 \iff \\
&\iff \#_a(w_{\leq j}) - (n_a - \#_a(w_{\leq j})) + (n_b - \#_b(w_{\leq j})) - \#_b(w_{\leq j}) > 0 \iff \\
&\iff 2\#_a(w_{\leq j}) - 2\#_b(w_{\leq j}) - n_a + n_b > 0
\end{aligned}$$

The last equation describes a half plane. Given a word w and considering the path $(x_0, y_0), \ldots, (x_n, y_n)$ corresponding to w, then $w_{y=j} \models \varphi$ iff (x_j, y_j) is part of the half plane corresponding to φ. This is illustrated by an example in Figure 6.2.

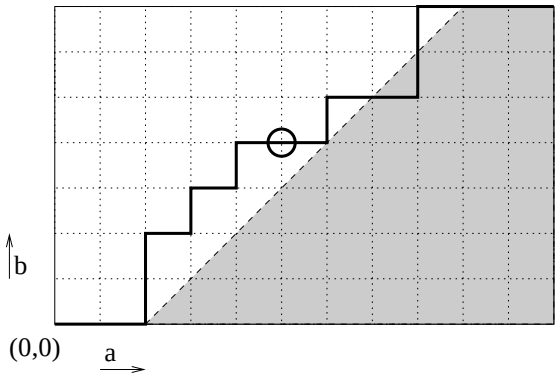


Figure 6.2: *The path for the word aabbababăabaabbaaa, where ă marks the position of the free variable y and half plane corresponding to the formula $\varphi = \text{Maj } x\ ((\mathfrak{c}_a(x) \wedge x \leq y) \vee (\mathfrak{c}_b(x) \wedge x > y))$.*

Below we will show that for each formula of $\widehat{\text{MAJ}}_1[<]$ we can find a set of

corresponding half planes corresponding to them. To this purpose we need to analyze all formulas of depth one.

Let φ be a formula Maj $x\ \langle \psi_1, \ldots, \psi_k \rangle$ with one free variable y. The subformulas ψ_l are Boolean combinations of $x < y, x = y, x > y$ and $\mathfrak{c}_a(x), \mathfrak{c}_b(x), \mathfrak{c}_a(y), \mathfrak{c}_b(y)$. We let $\mathfrak{c}_{ab}(x) = \mathfrak{c}_a(x) \vee \mathfrak{c}_b(x)$ and $\mathfrak{c}_\emptyset(x) = \bot$, and let B be the set $\{\emptyset, \{a\}, \{b\}, \{a, b\}\}$. Then every subformula ψ_l can be written in the form:

$$\mathfrak{c}_a(y) \wedge \left((x < y \wedge \mathfrak{c}_{B^{(l)}_{a,<}}) \vee (x = y \wedge \mathfrak{c}_{B^{(l)}_{a,=}}) \vee (x > y \wedge \mathfrak{c}_{B^{(l)}_{a,>}})\right) \vee$$
$$\vee \mathfrak{c}_b(y) \wedge \left((x < y \wedge \mathfrak{c}_{B^{(l)}_{b,<}}) \vee (x = y \wedge \mathfrak{c}_{B^{(l)}_{b,=}}) \vee (x > y \wedge \mathfrak{c}_{B^{(l)}_{b,>}})\right),$$

where $B^{(l)}_{a,<}, B^{(l)}_{a,=}, B^{(l)}_{a,>}, B^{(l)}_{b,<}, B^{(l)}_{b,=}, B^{(l)}_{b,>} \in B$. Please note that for every word $w_{x=i,y=j}$ exactly one of the clauses $x?y \wedge C_{B^{(l)}_{a,?}}$ is true.

Given a fixed word $w_{y=j}$ then $w_{y=j} \models \varphi$ iff

$$\sum_{i=1}^{n} |\{l \mid w_{x=i,y=j} \models \psi_l\}| > kn/2 \iff \sum_{l=1}^{k} \{i \mid w_{x=i,y=j} \models \psi_l\}| > kn/2$$

Let $\#_{B'}(w) = \sum_{b \in B'} \#_b(w)$ for $B' \in B$, then for a fixed l the size of $\{i \mid w_{x=i,y=j} \models \psi_l\}$, equals

$$\begin{aligned}
&\#_{B^{(l)}_{w_j,<}}(w_{<j}) + \#_{B^{(l)}_{w_j,=}}(w_j) + \#_{B^{(l)}_{w_j,>}}(w_{>j}) = \\
&= \#_{B^{(l)}_{w_j,<}}(w_{<j}) + \#_{B^{(l)}_{w_j,=}}(w_j) + n_{B^{(l)}_{w_j,>}} - \#_{B^{(l)}_{w_j,>}}(w_{<j}) - \#_{B^{(l)}_{w_j,>}}(w_j) = \\
&= \left(\#_{B^{(l)}_{w_j,<}}(w_{<j}) - \#_{B^{(l)}_{w_j,>}}(w_{<j})\right) + \left(\#_{B^{(l)}_{w_j,=}}(w_j) + n_{B^{(l)}_{w_j,>}} - \#_{B^{(l)}_{w_j,>}}(w_j)\right)
\end{aligned}$$

The inequality above can now be reformulated as:

$$\sum_{l=1}^{k} \left(\#_{B^{(l)}_{w_j,<}}(w_{<j}) - \#_{B^{(l)}_{w_j,>}}(w_{<j})\right) + \sum_{l=1}^{k} \left(\#_{B^{(l)}_{w_j,=}}(w_j) + n_{B^{(l)}_{w_j,>}} - \#_{B^{(l)}_{w_j,>}}(w_j)\right) > kn/2$$

The idea is to create two half planes, one for the case $w_j = a$ and one for the case $w_j = b$. We already fixed the Parikh vector of the word; if the letter w_j is also fixed the term on the right is constant, and the sum on the left is a linear combination of $\#_a(w_{<j})$ and $\#_b(w_{<j})$. Hence we obtain two half planes, one for $w_j = a$ and one for $w_j = b$, such that if for two words $w_{y=j}$ and $w'_{y=j'}$ with $\#(w) = \#(w') = (n_a, n_b)$ and $w_j = w'_{j'}$ and the marker on the path of $w_{y=j}$ being contained in the same set of half planes as the marker of the path of $w'_{y=j'}$ then $w_{y=j} \models \varphi \iff w'_{y=j'} \models \varphi$, i.e. φ cannot distinguish between $w_{y=j}$ and $w'_{y=j'}$.

Since every formula of depth one is a Boolean combination of formulas of the form of φ, if we pick any formula of depth one we can use this method to get a set of half planes with the above properties. It is important to note that the exact paths of w and w' have no influence, only the position of the marker has.

Thus every formula corresponds to a constant number of half planes for every Parikh vector (n_a, n_b), but the rectangle grows with (n_a, n_b). We show that for a fixed number of half planes, for an arbitrarily large rectangle being cut by these half planes there is always an arbitrarily large rectangle inside with all points contained in exactly the same set of half planes.

Lemma 6.1. *Given numbers c, n'_a, n'_b, and an rectangle of size at least $(2^c \cdot n'_a, 2^c \cdot n'_b)$, for every set of c half planes we can find a rectangle inside of size (n'_a, n'_b) where all points of this rectangle are contained in exactly the same set of half planes.*

Proof. We split the interval into four quadrants, each of size $(2^{c-1}n'_a, 2^{c-1}n'_b)$, and pick a half plane. Since a line can only intersect three quadrants at most, there is one quadrant that is either completely contained in the half plane or completely outside. By induction over the number of half planes we get the result. □

The basic idea to characterize the languages $\mathcal{L}(\widehat{\mathrm{MAJ}}_2[<])$ is the following: Given a formula φ' recognizing a language L, we proceed as above and find for every Parikh vector (n_a, n_b) a rectangle uncut by the half planes corresponding to the subformulas $\varphi_1, \ldots, \varphi_k$ of depth one in φ'. We can assume that the formulas have a free variable named y. So for each Parikh vector (n_a, n_b) we have two words $p_{(n_a,n_b)}$ and $s_{(n_a,n_b)}$ such that $p_{(n_a,n_b)}$ corresponds to a path from the origin to the lower left vertex of the rectangle and $s_{(n_a,n_b)}$ corresponds to a path from the top right vertex to the vertex (n_a, n_b).See Figure 6.3 for an example.

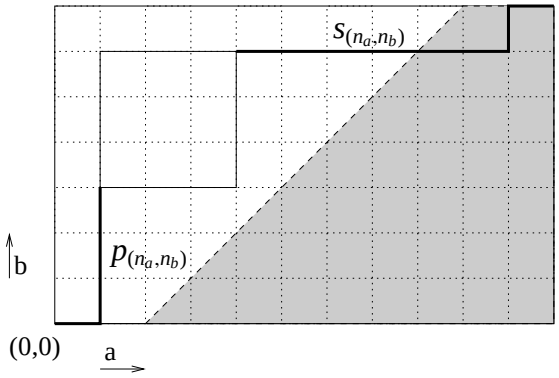


Figure 6.3: *Here we choose the rectangle as drawn in the top left corner. The prefix we choose is abbb and the suffix is aaaaaaba. The formula $\varphi = \mathrm{Maj}\ x\ ((\mathfrak{c}_a(x) \wedge x \leq y) \vee (\mathfrak{c}_b(x) \wedge x > y))$ has no word $w_{y=j}$ as model if abbb is a prefix of w and aaaaaba is a suffix and y points the a letter between the prefix and the suffix, i.e. $4 \leq j \leq 10$. Hence φ is constant on the inner rectangle. Please not that we have some freedom in choosing the prefix and suffix for the same rectangle.*

If we now choose two words w, w' with $\#(w) = \#(w') = (n_a, n_b)$ and $p_{(n_a,n_b)}$ is a prefix of w and w' and $s_{(n_a,n_b)}$ is a suffix of w and w', then $w_{y=j} \models \varphi_l \iff w'_{y=j'} \models \varphi_l$ for all $l = 1, \ldots, k$ and $j = 1, \ldots, n$, $j' = 1, \ldots, n$. Hence all subformulas $\varphi_1, \ldots, \varphi_k$ are

constant except for the prefix and suffix. We can thus define a new language that has fixed prefixes and suffixes depending on the Parikh vector of the input, such that this language is recognized by a formula of a quantifier depth less than the quantifier depth of φ'.

This idea has two drawbacks, firstly the technical formalism we will deal with in the next section, and secondly the need to choose the right prefixes and suffixes. Since we modify the language and create a new language with fixed prefixes and suffixes it is important that the prefixes and suffixes added do not change the complexity of the language. The language $\Sigma^* bb \Sigma^*$ for example with a fixed prefix of bb becomes trivial. The best choice is to pick words that behave neutral with respect to the language as prefixes and suffixes, but this is not always possible. So once we are done with the formalism we will carefully choose prefixes and suffixes in Section 6.3.

6.2 Non-uniform morphisms

The algebraic tool to deal with prefixes and suffixes that depend on the Parikh vector of the word will be the non-uniform morphisms. These are morphisms that can be shifted in a non-uniform way, depending on the number of letters in the words, i.e. the Parikh vector.

Definition 6.2 (Non-Uniform Morphism)**.** Let $(S, \mathfrak{S}, \mathcal{E})$ be a typed semigroup. A *non-uniform morphism* into $(S, \mathfrak{S}, \mathcal{E})$ is a triple (h, λ, ϱ), where h is a semigroup morphism $h : \Sigma^+ \to S$ with $h(\Sigma) \subseteq \mathcal{E}$ and λ, ϱ are mappings $\lambda, \varrho : \mathbb{N}^\Sigma \to S$. A non-uniform morphism (h, λ, ϱ) *recognizes a language* $L \subseteq \Sigma^+$ iff there is a type $\mathcal{S} \in \mathfrak{S}$ such that $w \in L \iff \lambda(\#(w))h(w)\varrho(\#(w)) \in \mathcal{S}$.

In the following we will show some methods of modifying a non-uniform morphism, according to the changes we needed in the previous section. First as one of the basics we show that the composition of two non-uniform morphism is again a non-uniform morphism. Since we defined the notion of a non-uniform morphism only on the free semigroups, the image of the first non-uniform morphism will be a subset of the free semigroup Σ'^+.

Lemma 6.3. *Assume there is a non-uniform morphisms between to alphabets, that is* (h, λ, ϱ) *is a typed morphism from* Σ^+ *to* (Σ'^+, L', Σ) *that recognizes* L*. If there is a non-uniform morphism* (h', λ', ϱ') *to* $(S, \mathfrak{S}, \mathcal{E})$ *that recognizes* L'*, then the non-uniform morphism* $\Big(h' \circ h, (\lambda' \circ (\#(\lambda) + \#h + \#(\varrho)))(h' \circ \lambda), (h' \circ \varrho)(\varrho' \circ (\#(\lambda) + \#h + \#(\varrho)))\Big)$ *recognizes the language* L*.*

Proof. We prove this by computation.

Let $w \in \Sigma^+$, and let $w' = \lambda(\#(w))h(w)\varrho(\#(w))$ then:

$$\begin{aligned}
&\Big(\lambda' \circ (\#(\lambda) + \#h + \#(\varrho))\Big)\Big(\#(w)\Big)\Big(h' \circ \lambda\Big)\Big(\#(w)\Big) \\
&\qquad\qquad \Big(h' \circ h\Big)\Big(w\Big)\Big(h' \circ \varrho\Big)\Big(\#(w)\Big)\Big(\varrho' \circ (\#(\lambda) + \#h + \#(\varrho))\Big)\Big(\#(w)\Big) = \\
&= \lambda'(\#(w'))h'(\lambda(\#(w)))h'(h(w))h'(\varrho(\#(w)))\varrho'(\#w(w')) = \\
&= \lambda'(\#(w'))h'(\lambda(\#(w))h(w)\varrho(\#(w)))\varrho'(\#w(w')) = \\
&= \lambda'(\#(w'))h'(w')\varrho'(\#w(w'))
\end{aligned}$$

Since $w \in L$ iff $w' \in L'$ and (h', λ', ϱ') recognizes the language L', this completes the proof. □

We need to be able to extend the prefixes and suffixes chosen. Thus we show: Given a non-uniform morphism, we can fix a prefix or suffix of the word depending on the number of letters in the word, and still recognize the result with the same typed semigroup.

We recall a definition from the preliminaries that we need in the following. For a morphism $h : \Sigma^+ \to S$, we let $\#h : \mathbb{N}^\Sigma \to \mathbb{N}^S$ be the map that maps v to $\#(h(w))$, where $\#(w) = v$. This definition is sound since h is a morphism and so the map is independent of the word w chosen.

Definition 6.4 (Prefix/Suffix Extension)**.** We call a pair (λ', ϱ') with $\lambda' : \mathbb{N}^\Sigma \to \Sigma^+$ and $\varrho' : \mathbb{N}^\Sigma \to \Sigma^+$, where $v - \#(\lambda'(v)) - \#(\varrho'(v))$ is unbounded for all $v \in \mathbb{N}^\Sigma$, a *prefix/suffix extension*. Given a non-uniform morphism (h, λ, ϱ) we say, the prefix extension (λ', ϱ') leads to a non-uniform morphism $(h, \tilde{\lambda}, \tilde{\varrho})$, where $\tilde{\lambda}(v) = \lambda(v+\#(\lambda'(v))+\#(\varrho'(v)))h(\lambda'(v))$ and $\tilde{\varrho}(v) = h(\varrho'(v))\varrho(v + \#(\lambda'(v)) + \#(\varrho'(v)))$.

When we extend the prefixes and suffixes the language recognized changes. We define a cut of language by a prefix/suffix extension and then prove that this gives the corresponding change of the language.

Definition 6.5 (Prefix/Suffix Cut $\lambda'^{-1}L\varrho'^{-1}$)**.** A *prefix/suffix cut* of a language L by a prefix/suffix extension is the language: $\lambda'^{-1}L\varrho'^{-1} = \{w \mid \lambda'(\#(w))w\varrho'(\#(w)) \in L\}$.

We now prove the compatibility of the algebraic definition of a prefix/suffix extension with the prefix/suffix cut of the language.

Lemma 6.6. *If (h, λ, ϱ) recognizes a language L, then the prefix/suffix extension (λ', ϱ') of (h, λ, ϱ), results in a non-uniform morphism $(h, \tilde{\lambda}, \tilde{\varrho})$ that recognizes $\lambda'^{-1}L\varrho'^{-1}$.*

Proof. Let $w \in \Sigma^+$, then $w \in \lambda'^{-1}L\varrho'^{-1}L$ iff $\lambda'(\#(w))w\varrho'(\#(w)) \in L$. Since (h, λ, ϱ) recognizes L, there is a type A such that $\lambda(\#(w'))h(w')\varrho(\#(w')) \in A$ if $w' \in L$. If we set $w' = \lambda'(\#(w))w\varrho'(\#(w))$, then we get

$$\begin{aligned}&\lambda(\#(\lambda'(\#(w))w\varrho'(\#(w))))h(\lambda'(\#(w))w\varrho'(\#(w)))\varrho(\lambda'(\#(w))w\varrho'(\#(w))) =\\ =&(\lambda(\#(w)+\#(\lambda'(\#(w)))+\#(\varrho'(\#(w))))h(\lambda'(\#(w))))h(w)\cdot\\ &\qquad\cdot(h(\varrho'(\#(w)))\varrho(\#(w)+\#(\lambda'(\#(w)))+\#(\varrho'(\#(w))))) =\\ =&\tilde{\lambda}(\#(w))h(w)\tilde{\varrho}(\#(w)))\in A\end{aligned}$$

iff $\lambda'(\#(w))w\varrho'(\#(w)) \in L$ which is the same as $w \in \lambda'^{-1}L\varrho'^{-1}$. □

The next lemma shows that the languages recognized by non-uniform morphisms are closed under inverse length preserving morphisms and under shifting, i.e. the syntactic semigroups are closed under division and shifting.

Lemma 6.7. *If (h, λ, ϱ) from Σ^+ to $(S, \mathfrak{S}, \mathcal{E})$ recognizes a regular language L, a semigroup T divides the syntactic semigroup of L and a subset $B \subseteq T$, then there is a non-uniform morphism (h', λ', ϱ') to a semigroup of $\mathrm{wc}((S, \mathfrak{S}, \mathcal{E}))$, that recognizes a language L' with syntactic semigroup $(\widetilde{T, \mathfrak{T}, \mathcal{F}})$ for some units $\mathcal{F}$.*

Proof. First we prove the lemma in the special case that $T = S$.

We let $(w_{l_1}, w_{r_1}), \ldots, (w_{l_k}, w_{r_k})$ we the set of tuples of words in Σ^+ such that this set maps surjective on $\mathrm{syn}(L) \times \mathrm{syn}(L)$ by $\eta \times \eta$. We define $(h', \lambda', \varrho') : \Sigma^+ \to (S, \mathfrak{S}, \mathcal{E})^k$ where the π_i of the non-uniform morphism is the prefix/suffix extension by w_{l_i}, w_{r_i}.

Since $\mathrm{syn}(L)$ is the syntactic semigroup of L, with $L = \eta^{-1}(A)$, for every pair $s, s' \in \mathrm{syn}(L)$, there are elements $l, r \in \mathrm{syn}(L)$ such that $lsr \in A \iff ls'r \notin A$. Hence for every elements $s \in \mathrm{syn}(L)$ there is type A' of $(S, \mathfrak{S}, \mathcal{E})^k$ such that $h^{-1}(A') = \eta^{-1}(s)$. Also since the types form a Boolean algebra, there is type for every subset.

Now we prove the lemma in the common case. Let $T' \subseteq \mathrm{syn}(L)$ such that T is a morphic image of T' under $\tilde{h}$. We can assume that $B = \tilde{h}(T' \cap A)$, then we are in the following situation:

$$\begin{array}{ccccc}(\Sigma^+, L, \Sigma) & \xrightarrow{(h,\lambda,\varrho)} & (S', \mathfrak{S}', \mathcal{E}') & \longrightarrow & (S, \mathfrak{S}, \mathcal{E})\\ \downarrow\eta & \swarrow\varphi & & & \\ (S_L, \mathfrak{S}_L, \mathcal{E}_L) & \longleftarrow & (T', B') & \xrightarrow{\tilde{h}} & (T, B)\end{array}$$

We pick generators Σ' such that $\Sigma'^+ = \eta^{-1}(\tilde{h}^{-1}(T'))$, and let $L' = \Sigma'^+ \cap L$. For all $w' \in \Sigma'^+$ we have $w \in L'$ iff $\tilde{h}(\eta(w)) \in B$, hence $\mathrm{syn}(L') \preceq T$ and $L' = \eta'^{-1}(\tilde{h}^{-1}(T'))$, hence $(T, \mathfrak{T}, \mathcal{F})$ is a trivial extension of the syntactic semigroup for $\mathcal{F} = \tilde{h}(\eta(\Sigma'))$. Also this language is still recognized by $(S, \mathfrak{S}, \mathcal{E})$. □

In the following we will show we can reduce the block depth of the non-uniform morphism (h, λ, ϱ), where $\pi_2(h(\Sigma^+))$ is commutative, recognizing a language L by extending the prefix and suffix by λ', ϱ', and the so constructed morphism recognizes $\lambda'^{-1}L\varrho'^{-1}$.

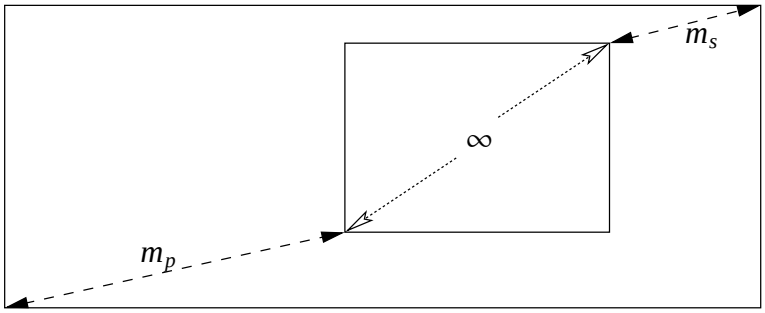


Figure 6.4: *m_p, m_s of Reduction Lemma*

Since all languages except group, will have a smaller syntactic semigroup when adding any prefix/suffix, we have some condition that enables us to control which prefix/suffix applied in the construction. Assume we have fixed the Parikh vector of the word recognized, then we want to be able to ensure a certain minimal number of letters in the prefix and suffix. We will have to functions m_p, m_s that describe the minimum letters in the prefix/suffix as shown in the next picture. The only condition on the functions m_p, m_s is that the number of letters that are non fixed is unbounded.

Then after our construction, we will have to functions t_p, t_s, and can choose any prefix/suffix extension with that has more letters in the prefix/suffix yielding a morphism of lower depth.

The following lemma is the main statement of this section and will be a key tool to prove the main theorems of this and the next chapter.

Lemma 6.8 (Reduction Lemma)**.** *Let $\mathbf{T}'$ be a weakly closed class of typed semigroups. Let L be recognized by a non-uniform morphism (h, λ, ϱ) into a semigroup of the weakly closed class $\mathbf{T} = \mathbf{T}' \boxdot (\mathbf{Z} \boxdot \mathbf{P}_<)$ and let $m_p, m_s : \mathbb{N}^\Sigma \to \mathbb{N}^\Sigma$ be two mappings, such that $\|y - m_p(y) - m_s(y)\|_0$ is unbounded for $y \in \mathbb{N}^\Sigma$. There are mappings $t_p, t_s : \mathbb{N}^\Sigma \to \mathbb{N}^\Sigma$ such that for all prefix/suffix extensions λ', ϱ' with $\#(\lambda') \geq t_p$ and $\#(\varrho') \geq t_s$ there is a non-uniform morphism $(\tilde{h}, \tilde{\lambda}, \tilde{\varrho})$ into a semigroup of the weakly closed class $\mathbf{T}'$ that recognizes $\lambda'^{-1} L \varrho'^{-1}$.*

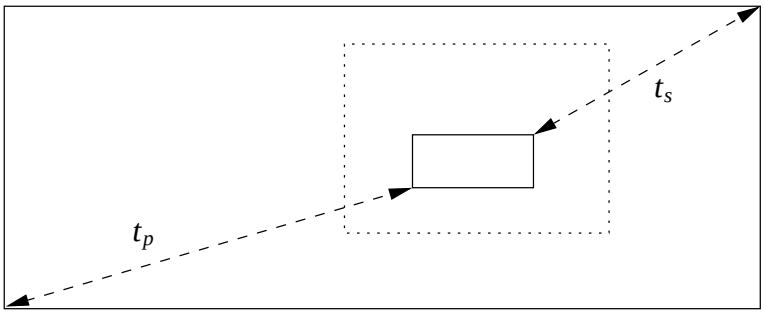


Figure 6.5: *t_p, t_s of Reduction Lemma*

Proof. Let $w \in \Sigma^+$ and $h(w_i) = (f_{w_i}, g_{w_i}, e)$. Then $w \in L \iff \lambda(\#(w))h(w)\varrho(\#(w)) \in \mathcal{S}$ for some type $\mathcal{S}$ which is equivalent to

$$\begin{aligned} & f_\lambda(e, \prod_j (g_j, e)(g_\varrho, p_\varrho))\cdot \\ \prod_i & f_{w_i}((g_\lambda, p_\lambda) \prod_{j<i}(g_{w_j}, e), \prod_{j>i}(g_{w_j}, e)(g_\varrho, p_\varrho))\cdot \\ & f_\varrho((g_\lambda, p_\lambda) \prod_j (g_j, e), e) \in \pi_1 \mathcal{S} \end{aligned}$$

by Lemma 3.35. We will show that for certain fixed prefixes/suffixes the value of the function f_σ is constant on all words. Thus we can then replace f_σ by its only value and get a morphism to semigroup of $\mathbf{T}'$.

Since e is the unit and hence the identity of $\mathbf{P}_<$, and $\mathbb{Z}$ is commutative, the subsemigroup $\pi_2(h(\Sigma^*))$ is commutative. Pick a fixed position i, then

$$f_{w_i}((g_\lambda, p_\lambda) \cdot \prod_{j<i}(g_{w_j}, e), \prod_{j>i}(g_{w_j}, e) \cdot (g_\varrho, p_\varrho))$$

depends on clauses of the form:

$$(g_\lambda, p_\lambda) \cdot \prod_{j<i}(g_{w_j}, e) \cdot (g_f, p_f) \cdot \prod_{j>i}(g_{w_j}, e) \cdot (g_\varrho, p_\varrho) \in \mathcal{S}'$$

for a type $\mathcal{S}'$ of the semigroup $\pi_2(\Sigma^+) \in \mathbf{Z} \boxdot \mathbf{P}_<$ and an element $(g_f, p_f) \in \mathbb{Z} \boxdot \mathbf{P}_<$. This is again by Lemma 3.35 equivalent to

$$g_\lambda(e, p_f \cdot p_\varrho) + \sum_{j<i} g_{w_j}(p_\lambda, p_f \cdot p_\varrho) + g_f(p_\lambda, p_\varrho) + \sum_{j>i} g_{w_j}(p_\lambda \cdot p_f, p_\varrho) + g_\varrho(p_\lambda \cdot p_f, e) > 0.$$

Let $\sigma_< = g_\sigma(p_\lambda, p_f \cdot p_\varrho)$, $\sigma_> = g_\sigma(p_\lambda \cdot p_f, p_\varrho)$, $\gamma_* = g_\lambda(e, p_f \cdot p_\varrho) + g_f(p_\lambda, p_\varrho) + g_\varrho(p_\lambda \cdot p_f, e)$ then we can rewrite the expression above as:

$$\sum_{\sigma \in \Sigma} \#_\sigma(w_{<i})\sigma_< + \sum_{\sigma \in \Sigma} \#_\sigma(w_{>i})\sigma_> + \gamma_* > 0$$

If we look at all words with the same Parikh vector as w with at least $m_p(\#w)_\sigma$ letters σ in the prefix and $m_s(\#w)_\sigma$ letters σ in the suffix, we can estimate this sum by $v_{\min}, v_{\max}$ if we check all possibilities for letters in the prefix and suffix, i.e.:

$$v_{\min} = \sum_{\sigma \in \Sigma} (\#_\sigma(w) - m_p(\#(w))_\sigma - m_s(\#(w))_\sigma) \min\{\sigma_<, \sigma_>\} + m_p(\#w)_\sigma \cdot \sigma_< + m_s(\#w)_\sigma \cdot \sigma_>,$$

$$v_{\max} = \sum_{\sigma \in \Sigma} (\#_\sigma(w) - m_p(\#(w))_\sigma - m_s(\#(w))_\sigma) \max\{\sigma_<, \sigma_>\} + m_p(\#w)_\sigma \cdot \sigma_< + m_s(\#w)_\sigma \cdot \sigma_>.$$

Assume that $\nu_{\min} + \nu_{\max} > 0$, then we we let

$$t_p(\#(w))_\sigma = m_p(\#(w))_\sigma + \lceil \#(w)_\sigma / 2 \rceil$$

iff $\sigma_< > \sigma_>$ and

$$t_p(\#(w))_\sigma = m_p(\#(w))_\sigma$$

otherwise. Also we let

$$t_s(\#(w))_\sigma = m_s(\#(w))_\sigma + \lceil \#(w)_\sigma / 2 \rceil$$

iff $\sigma_\geq > \sigma_<$ and

$$t_s(\#(w))_\sigma = m_s(\#(w))_\sigma$$

otherwise. In this way we assign about half of the letter to the prefix or suffix so the sum of the assigned letters is maximal.

We process equivalently if $\nu_{\min} + \nu_{\max} \leq 0$. In this way we assign about half of the letter to the prefix or suffix so the sum of the assigned letters is minimal.

No matter how the remaining letters between t_p and t_s are chosen the sum will always be positive (or non-positive) since $(\nu_{\min} + \nu_{\max})/2 > 0$ (≤ 0). Since we have finite many letter σ, there is a finite number of f_σ with a finite numbers of clauses. These clauses depend on the constants (g_f, p_f) of f_σ like above and for the finite set of clauses we can apply induction and fix about $(1 - 2^{-c}) \cdot n$ letters where c is the total number of clauses. Since $\|y - m_p(y) - m_s(y)\|_0$ is unbounded, also $\|y - t_p(y) - t_s(y)\|_0 \approx 2^{-c} \cdot \|y - m_p - m_s(y)\|_0$ is unbounded.

Hence if we choose any prefix/suffix extension (λ', ϱ') of (h, λ, ϱ), with $\#\lambda' \geq t_p$ and $\#\varrho' \geq t_s$, then the functions f_σ are constant in all evaluations that appear. Hence we create a new morphism $\tilde{h}$ that maps σ to the value of this evaluation, also the functions f_λ and f_ϱ are constant in all evaluations since $\pi_2(h(\Sigma^+))$ is commutative, hence we can also define $\tilde{\lambda}$, $\tilde{\varrho}$, that map a Parikh vector to the the only value f_λ or f_ϱ evaluates to. The resulting typed morphism recognizes then $\lambda'^{-1} L \varrho'^{-1}$, by Lemma 6.6. □

6.3 Application

In this section we use the reduction lemma, to show that certain (types of) languages cannot be recognized by a $\mathrm{M}\widehat{\mathrm{A}}\mathrm{J}_2[<]$ formula. The essential task in the following lemmas and theorems is to choose the suitable prefix/suffix extension that does not reduce the complexity of the language recognized to much. Please recall the situation of reduction Lemma 6.8, that we some freedom in choosing the prefix and suffix extensions.

We start by showing that we cannot recognize nonabelian groups which is easy since any prefix and suffix can be extended to a neutral word, which eases the use of reduction lemma.

Lemma 6.9. $\mathrm{M}\widehat{\mathrm{A}}\mathrm{J}_2[<] \subseteq \overline{\mathbf{Ab}}$

S_{bb}	a	b	ab	ba	0
a	a	ab	ab	a	0
b	ba	0	b	0	0
ab	a	0	ab	0	0
ba	ba	b	b	ba	0

Figure 6.6: *The semigroup S_{bb}*

Proof. Assume there is a $\widehat{\mathrm{MAJ}}_2[<]$ formula that recognizes a language with a syntactic semigroup that is divided by a non-Abelian group. Then there is morphism into wbpc($\mathbf{Z} \boxdot \mathbf{P}_<$) by Theorem 4.14 that recognizes a language L with syntactic semigroup that is a non-Abelian group by Lemma 6.7. Pick a non-uniform morphism of minimal block depth that recognizes L. We can use Lemma 6.8 with $m_p = m_s = \mathbf{e}$ and for any prefix/suffix extension we get a non-uniform morphism of smaller block depth that still has a syntactic semigroup that is divided by a non-Abelian group, a contradiction, since a formula of the depth 0 depends only on the input length, in our case the Parikh vector of the word. □

We will now turn our attention to the variety $\mathbf{DA} \square \mathbf{G}$ and show that $\mathcal{L}(\widehat{\mathrm{MAJ}}_2[<]) \subseteq \mathcal{L}(\mathbf{DA} \square \mathbf{G})$. We use the fact that the variety $\mathbf{DA} \square \mathbf{G}$ can be characterized by an equation to show that every semigroup outside of $\mathbf{DA} \square \mathbf{G}$ is divided by a fixed semigroup.

This fixed semigroup is syntactic semigroup of the language $L_{bb} = \Sigma^* bb \Sigma^*$ (see Figure 6.6).

Theorem 6.10. *$S \notin \mathbf{DA} \square \mathbf{G} = \mathbf{DA} * \mathbf{G}$ iff S_{bb} divides S.*

Proof. If $S \notin \mathbf{DA} \square \mathbf{G}$ then there are idempotents e, f in a D-class such that ef is in the same D-class but not idempotent. Hence the is an elements s such that $efs = e$ and an element t such that $tef = f$, it follows $fs = te$. We define a morphism $\{a, b\}^+ \to S$, with $a \mapsto fs$ and $b \mapsto ef$. Let $L = h^{-1}(h(\{a, b, ab, ba\}))$.

If we let $S' = h(\Sigma^+)$ then S' is a subsemigroup of S. Also the elements e, f, ef, fs are in the same D-class. If we show that these are the only elements in this D-class we are done.

Let $w = \Sigma^* bb \Sigma^*$ then $h(w) \in S'(efef)S'$. By computation one gets that

$$S'(efef)S' = \{(ef)^{i+1}, e(fe)^i, (fe)^i f, (fe)^i \mid i \geq 1\},$$

hence $w \notin L$. Let $w = [b]a^+(ba^+)^+[b]$, since $h(a) = fs$ is idempotent and $h(ba) = effs = e$ is idempotent we get $h(w) \in h(a) \cup h(ab) \cup h(ba) \cup h(bab)$, and since $h(bab) = (effs)ef = eef = ef = h(b)$, the result follows. □

Next we show that L_{bb} cannot be recognized by $\widehat{\mathrm{MAJ}}_2[<]$ and hence by the previous theorem $\mathcal{L}(\widehat{\mathrm{MAJ}}_2[<])$ is contained in $\mathcal{L}(\mathbf{DA} \square \mathbf{G})$.

Lemma 6.11. $L_{bb} \notin \widehat{\mathrm{MAJ}}_2[<]$

Proof. Assume there is a $\widehat{\mathrm{MAJ}}_2[<]$ formula that recognizes the language L_{bb}. We let $m_p(v) = m_s(v) = (v_b, 1)$ for a vector $v = (v_a, v_b)$, this ensures that when we apply Lemma 6.8 then $(t_p)_a \geq (t_p)_b$ and also $(t_s)_a \geq (t_p)_a$. Hence we can choose a prefix of the form $a^*(ba)^+$ and a suffix of the form $a^*(ab)^+$. But $w \in L$ iff $a^*(ba)^+wa^*(ab)^+ \subseteq L$, so we can recognize L_{bb} also in lower depth $\lightning$. □

Finally adding up the results of the section we get.

Theorem 6.12. $\widehat{\mathrm{MAJ}}_2[<] \subseteq \mathbf{DA} \square \mathbf{G} \cap \overline{\mathbf{Ab}}$

Proof. This follows from the previous lemmas and the fact that $\widehat{\mathrm{MAJ}}_2[<]$ is a variety (Theorem 5.9). □

The other language of special interest to prove containment in certain varieties is L_{B_2}. Bit L_{B_2} is contained in $\widehat{\mathrm{MAJ}}_2[<]$ hence $\widehat{\mathrm{MAJ}}_2[<]$ can recognize languages outside of **DS**. Within **DS** we get an exact characterization of the syntactic semigroups of $\mathfrak{L}(\widehat{\mathrm{MAJ}}_2[<])$ in the next section.

6.4 Lower Bound

For the lower bound we look again at the geometric intuition and observe that the class of languages where we cannot apply reduction lemma, has a clear geometric picture. Informally speaking these languages have only very limited freedom for the words of the language, that is there a parallel half planes of constant distance that leave only small corridors. This is shown for the language L_{B_2} in Figure 6.7.

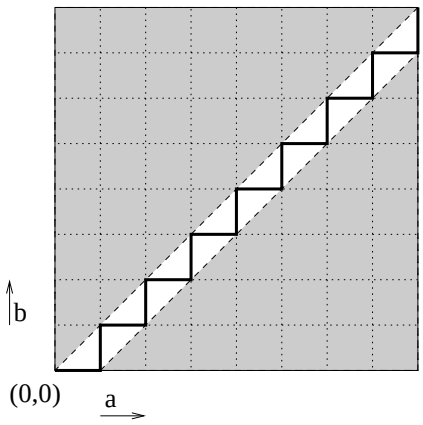


Figure 6.7: *Small corridor for L_{B_2}*

With this intuition we define a variety that contains language of the form described above.

Definition 6.13 (Generalized Bicycle Variety)**.** Let $\mathcal{B}_n$ be the languages for different set of parenthesis of all Dyck words with depth at most n. We let **E** be the variety generated by all $\mathcal{B}_n$.

We give a construction that shows that we can recognize all iterated **E**-transduction of $\overline{\mathbf{Ab}}$

Theorem 6.14. $\mathcal{L}(\mathrm{M}\widehat{\mathrm{A}}\mathrm{J}_2[<]) \supseteq \mathcal{L}((\cdots((\overline{\mathbf{Ab}} \square \mathbf{E})) \square \mathbf{E} \dots) \square \mathbf{E}$

Proof. If we need to show that $\mathcal{L}(\overline{\mathbf{Ab}}) \subseteq \mathcal{L}(\mathrm{M}\widehat{\mathrm{A}}\mathrm{J}_2[<])$ and that $\mathcal{P}_1(\mathbf{E}) \subseteq \mathcal{P}_1(\mathrm{M}\widehat{\mathrm{A}}\mathrm{J}_2[<])$, we can apply the Block Product Principle 3.40 and we are done.

For the first equation it suffices that we can compute $\mathcal{L}(\mathbb{Z}_q)$ for every $q \in \mathbb{Z}^+$. Let $L = \{w \mid |w| \equiv 0 \mod q\}$, then $\mathrm{syn}(L) = Z_q$. We can describe this language by

$$\mathrm{Maj}\ x\ \mathrm{Maj}\ y\ \langle \underbrace{x < y, \dots, x < y}_{q}, \underbrace{\top, \dots, \top}_{q-2} \rangle.$$

Since $\mathrm{M}\widehat{\mathrm{A}}\mathrm{J}_2[<]$ is a variety all languages with $\mathrm{syn}(L) = \mathbb{Z}_q$ can be described, hence $\mathcal{L}(\overline{\mathbf{Ab}}) \subseteq \mathcal{L}(\mathrm{M}\widehat{\mathrm{A}}\mathrm{J}_2[<])$.

For the second equation it suffices to show that we can compute B_n on every prefix, i.e. given a word w_x, we need to check if $w_{<x} \in B_k$. This can be done by the formula

$$\begin{aligned}
&\exists y \le x \wedge (\neg \mathrm{Maj}\ y\ \langle \mathfrak{c}_b(y) \vee y > x, \neg(\mathfrak{c}_a(y) \vee y > x) \rangle) \wedge \\
&\quad \bigvee_{j=1}^{k} (\neg \mathrm{Maj}\ y\ \langle \mathfrak{c}_a(y) \vee y > x, \neg(\mathfrak{c}_b(y) \vee y > x), \mathfrak{c}_a(y) \vee y > x, \neg(\mathfrak{c}_b(y) \vee y > x), \\
&\qquad\qquad \underbrace{x \neq y, \bot, \dots, x \neq y, \bot}_{k} \rangle)
\end{aligned}$$

The first line checks that the number of b's in the prefix is never greater than the number of a's and the second line checks that the number of a's is not greater than the number of b's plus k. □

Since $\mathbf{SL} \subseteq \mathbf{E}$ we get as a corollary:

Corollary 6.15. $\mathcal{L}(\mathrm{M}\widehat{\mathrm{A}}\mathrm{J}_2[<]) \supsetneq \mathbf{DO} \cap \overline{\mathbf{Ab}}$

Together with the upper bound we get characterizations of some subclasses of the regular languages.

Corollary 6.16. $\mathcal{L}(\mathrm{M}\widehat{\mathrm{A}}\mathrm{J}_2[<]) \cap \mathcal{L}(\mathbf{DS}) = \mathbf{DO} \cap \overline{\mathbf{Ab}}$

Corollary 6.17. $\mathcal{L}(\mathrm{M}\widehat{\mathrm{A}}\mathrm{J}_2[<]) \cap \mathcal{L}(\mathbf{G}) = \mathbf{Ab}$

The language where we have only limited choice in the prefix and suffix, are exactly the languages where we cannot get a tight upper bound, and on the other hand we can recognize all these languages by the previous theorem we conjecture.

Conjecture 6.18. $\widehat{\mathrm{MAJ}}_2[<] = (\cdots((\overline{\mathbf{Ab}} \square \mathbf{E})) \square \mathbf{E} \ldots) \square \mathbf{E}$

The problem with this class is that it is not very well known, so there no known algebraic characterization in terms equations, which leaves us with the problem that even if we could prove for given equations that they are always fulfilled, we do not know which equations to test.

6.5 Summary

In this chapter we aimed to characterize the regular languages recognizable by $\widehat{\mathrm{MAJ}}_2[<]$ formulas. To that goal we first assigned paths to words and half planes to majority formulas of depth one. We then showed that a $\widehat{\mathrm{MAJ}}_2[<]$ formula cannot differentiate between two words where all points of the path are always in the same compartment. Since a finite number of half planes cannot separate all points (Lemma 6.1) we can always find a rectangle which is not intersected by these half planes.

Since the geometric proof might seem intuitive yet not verifiable we proved the same conclusions in algebraic terms. Here we needed non-uniform morphisms (Definition 6.2) to simulate the geometry and obtain the reduction Lemma 6.8.

We could use the reduction Lemma 6.8 to reduce the block depth of the typed semigroup recognizing a language by the cost of prepending/appending prefixes/suffixes to the language. Having a lot of freedom in choosing the prefixes and suffixes for a concrete language it was not difficult to detect whether it is recognizable by a $\widehat{\mathrm{MAJ}}_2[<]$ formula. In order to use this result to prove certain varieties to be recognizable we needed a concrete characterization of semigroups not in the variety as in Theorem 6.10. We used this and show that we can recognize only languages in $\mathbf{DA} \square \mathbf{G} \cap \overline{\mathbf{Ab}}$.

Some languages though, like the bicycle, allow only for few prefixes and suffixes to be appended, e.g. in the case of the bicycle we can only append $(ab)^*$ or $(ab)^*a$, otherwise the remaining language becomes trivial. We conjectured that all languages like the bicycle are recognizable by $\widehat{\mathrm{MAJ}}_2[<]$. We could show that languages with a syntactic semigroup in $(\cdots((\overline{\mathbf{Ab}} \square \mathbf{E})) \square \mathbf{E} \ldots) \square \mathbf{E} \supsetneq \mathbf{DO} \cap \overline{\mathbf{Ab}}$ can be recognized by $\widehat{\mathrm{MAJ}}_2[<]$, where $\mathbf{E}$ is the variety spanned by the generalized bicycles (Definition 6.13).

6.6 Further Research

Of course it is not satisfying to achieve an upper and lower bound but not characterize the exact variety. The question still open is whether we can extend the current proof technique to obtain a better upper bound. One possibility is separating the language L with a language L_1 with only few possible prefixes and suffixes, so it is, geometrically speaking, bound in some small corridors, and a language L_2 where we do not need the half spaces generating these small corridors. While performing induction on L_2 is easy, we still do not know how to handle the problem for L_1 where we know the difference in letters to be very restricted for every prefix and suffix.

Another possible direction of research would be a characterization of the regular languages for the case of an unbounded number of variables. However, any upper bound below all regular languages or any lower bound above $\overline{\mathbf{G_{solv}}}$ would already touch the question if $\mathrm{TC}^0 = \mathrm{NC}^1$; actually $L_{A_5} \notin \mathrm{MAJ}[<]$ implies $\mathrm{FO}[<,+]$−uniform $\mathrm{TC}^0 \neq \mathrm{NC}^1$. For unbounded variables the strong block product would be used and hence the paths would not be in $\mathbb{Z} \times \mathbb{Z}$ but in some non-commutative group, so we would need some "non-commutative" geometric intuition for the proof to work.

Another interesting question is whether we can apply geometric intuition to other classes of logic. The FO quantifiers for example yield half planes that have borders parallel to the edges of the rectangle and with a constant distance. Hence all words with a certain minimum length quality as prefixes and suffixes implying the equations of **DA**. Modulo quantifiers are different since we cannot describe them by an inequality, but still they might be feasible by this approach.

Chapter 7

A5 not in FO+MOD+$\widehat{\text{MAJ}}_2$[reg,arb-un]

In this chapter we will extend majority logic with two variables as far as we can while proving that this class still does not contain all regular languages. Since already $(\text{FO} + \text{MOD})_2[<]$ contains all group languages with solvable syntactic groups, they only differ in the group languages with syntactic semigroups having non-solvable subgroups. Here we show that despite extending the class of logic to $(\text{FO} + \text{MOD} + \widehat{\text{MAJ}})_2[\text{reg}, \text{arb} - \text{un}]$ we still cannot recognize a non-solvable group, like A_5.

We will first show the result without arbitrary unary predicates and then extend it to the unary predicates.

7.1 A5 not in FO+MOD+MAJ$_2$[reg]

In this section we assume that we have a non-uniform morphism that recognize a language with a non-solvable syntactic semigroup and show step by step that with the usage of a neutral letter we can eliminate the extra predicates and quantifiers. First we will eliminate the succ predicate, then we show that the modulo quantifier can be replaced by a modulo predicate, which we eliminate in the next step, and finally we have a $\widehat{\text{MAJ}}_2[<]$ formula. Then we can use the reduction lemma of the previous chapter and reduce the block depth.

We let **SemiLin** be the smallest variety containing $(\mathbb{Z}, \mathbb{Z}_p)$ for all numbers $p > 2$ and $(\mathbb{Z}, \mathbb{Z}^+)$. All semigroups in this variety divide direct products of $\mathbb{Z}$ with a typeset generated by a semilinear set.

In the next lemma we show that in the presents of a neutral letter we can modify a (non-uniform) morphism such that it still recognizes the same language but does not need the predicate group P_{succ} at the lowest level anymore.

For a logic formula this means that we do not need the successor predicate in the

innermost subformulas.

Lemma 7.1. *Let* (h, λ, ϱ) *be a non-uniform morphism to* $\mathbf{T} \boxdot (\mathbf{SemiLin} \boxdot \mathbf{P}_{<,\mathbf{succ}})$ *that recognizes a language* L *with a neutral letter* e*, then there is a non-uniform morphism* $(\tilde{h}, \tilde{\lambda}, \tilde{\varrho})$ *to* $\mathbf{T} \boxdot (\mathbf{SemiLin} \boxdot \mathbf{P}_{<})$ *that recognizes* L*.*

Proof. The basic idea for this construction is to fill the neutral letter between any two letters of the word, then positions reachable by one succ predicate can be computed by the morphism. The following computation proves this. An alternative way to see the result directly without heavy computation is that it suffices that after this modification the computations in the predicate group $\mathbf{P}_{\mathbf{succ}}$, can be computed in the submonoid $(f_<, 0)\mathbf{P}_{\mathbf{succ}}(f_<, 0)$ which is Abelian. But since it is not straight-forward to see that an Abelian predicate semigroup cannot computer the succ predicate we give a computational proof.

We define an endomorphism of (Σ, L) by $\sigma \mapsto e\sigma e$. Then by Lemma 6.3 there is a nonuniform morphism (h', λ', ϱ') where $h' : \Sigma^+ \to T \boxdot (\mathbf{SemiLin} \boxdot \mathbf{P}_{<,\mathbf{succ}})$ with $h'(\sigma) = h(e\sigma e)$, $\lambda'(v) = \lambda(v + (2\|v\|_1 + 2)E_e)h(e)$, $\varrho'(v) = h(e)\varrho(v + (2\|v\|_1 + 2)E_e)$. Also (h', λ', ϱ') still recognizes L since e is a neutral letter of L.

Let $w \in \Sigma^+$ and $h'(w_i) = (f'_{w_i}, g'_{w_i}, p')$. Then $w \in L \iff \lambda'(\#(w))h'(w)\varrho'(\#(w)) \in \mathcal{T}$ which is equivalent to

$$
\begin{aligned}
&f'_\lambda\Big(e, (\prod_j (g'_{w_j}, p'))(g'_\varrho, p'_\varrho)\Big)\cdot \\
&\prod_i f'_{w_i}\Big((g'_\lambda, p'_\lambda)(\prod_{j<i}(g'_{w_j}, p')), (\prod_{j>i}(g'_{w_j}, p'))(g'_\varrho, p'_\varrho)\Big)\cdot \\
&f'_\varrho\Big((g'_\lambda, p'_\lambda)(\prod_j (g'_{w_j}, p')), e\Big) \in \pi_1\mathcal{T}
\end{aligned}
$$

by Lemma 3.35.

A computation shows that $p'_\lambda p' = p'_\lambda$, $p'p' = p'$ and $p'p'_\varrho = p'_\varrho$ (see the multiplication table in Figure 4.2). To reduce the number of cases we examine only words of length > 1:

$$
\begin{aligned}
&f'_\lambda\Big(e, (g'_{w_1}p'_\varrho + (\sum_{j>1} p'g'_{w_j}p'_\varrho) + p'g'_\varrho, p'_\varrho)\Big)\cdot \\
&f'_{w_1}\Big((g'_\lambda, p'_\lambda), (g'_{w_2}p'_\varrho + (\sum_{j>2} p'g'_{w_j}p'_\varrho) + p'g'_\varrho, p'_\varrho)\Big)\cdot \\
&\prod_{1<i<n} f'_{w_i}\Big((g'_\lambda p' + (\sum_{j<i-1} p'_\lambda g'_{w_j}p') + p'_\lambda g'_{w_{i-1}}, p'_\lambda), (g'_{w_{i+1}}p'_\varrho + (\sum_{j>i+1} p'g'_{w_j}p'_\varrho) + p'g'_\varrho, p'_\varrho)\Big)\cdot \\
&f'_{w_n}\Big((g'_\lambda p' + (\sum_{j<n-1} p'_\lambda g'_{w_j}p') + p'_\lambda g'_{w_{n-1}}, p'_\lambda), (g'_\varrho, p'_\varrho)\Big)\cdot \\
&f'_\varrho\Big((g'_\lambda p' + (\sum_{j<n} p'_\lambda g'_{w_j}p') + p'_\lambda g'_{w_n}, p'_\lambda), e\Big) \in \pi_1\mathcal{T}
\end{aligned}
$$

We extend the range of the uniform sum by adding a term and subtract it in each argument.

$$
\begin{aligned}
&f'_\lambda\Big(e, (g'_{w_1}p'_\varrho - p'g'_{w_1}p'_\varrho + (\sum_j p'g'_{w_j}p'_\varrho) + p'g'_\varrho, p'_\varrho)\Big)\cdot\\
&f'_{w_1}\Big((g'_\lambda, p'_\lambda), (g'_{w_2}p'_\varrho - p'g'_{w_2}p'_\varrho + (\sum_{j>1} p'g'_{w_j}p'_\varrho) + p'g'_\varrho, p'_\varrho)\Big)\cdot\\
\prod_{1<i<n} &f'_{w_i}\Big((g'_\lambda p' + (\sum_{j<i} p'_\lambda g'_{w_j}p') - p'_\lambda g'_{w_i-1}p' + p'_\lambda g'_{w_{i-1}}, p'_\lambda),\\
&\qquad (g'_{w_{i+1}}p'_\varrho - p'g'_{w_{i+1}}p'_\varrho + (\sum_{j>i} p'g'_{w_j}p'_\varrho) + p'g'_\varrho, p'_\varrho)\Big)\cdot\\
&f'_{w_n}\Big((g'_\lambda p' + (\sum_{j<n} p'_\lambda g'_{w_j}p') - p'_\lambda g'_{w_{n-1}}p' + p'_\lambda g'_{w_{n-1}}, p'_\lambda), (g'_\varrho, p'_\varrho)\Big)\cdot\\
&f'_\varrho\Big((g'_\lambda p' + (\sum_j p'_\lambda g'_{w_j}p') - p'_\lambda g'_{w_n}p' + p'_\lambda g'_{w_n}, p'_\lambda), e\Big) \in \pi_1\mathcal{T}
\end{aligned}
$$

We used often terms of the form $g'_{w_1}p'_\varrho - p'g'_{w_1}p'_\varrho$ to keep the sums uniform. Since these functions are $\mathbb{Z}$ valued the inverse element always exists. Also the sum is independent of the letter since $g'_\sigma - p'g'_\sigma = (g_ep + pg_\sigma p + pg_e) - p(g_ep + pg_\sigma p + pg_e) = g_ep - pg_ep$. Equivalent for a shift on the left we get: $g'_\sigma p' - g'_\sigma = (g_ep + pg_\sigma p + pg_e)p - (g_ep + pg_\sigma p + pg_e) = pg_ep - pg_e$. We let $g_l = g_e - pg_e$ and $g_r = g_e - g_ep$.

The difference in the arguments for f'_{w_1} and f'_{w_n} from any other f'_{w_i} is that g'_λ is not shifted by p' from the right in f'_{w_1} and g'_ϱ is not shifted by p' from the left in f'_{w_n}. We compute $g'_\lambda - g'_\lambda p = (g_\lambda p + p_\lambda g_e) - (g_\lambda p + p_\lambda g_e)p = p_\lambda g_e - p_\lambda g_ep = p_\lambda g_r$, and also $g'_\varrho - pg'_\varrho = (g_ep_\varrho + pg_\varrho) - p(g_ep_\varrho + pg_\varrho) = g_lp_\varrho$.

Substituting these definitions yields, please note $p'_\lambda pg_r = p'_\lambda g_r$ and $g_lpp'_\varrho = g_lp'_\varrho$:

$$
\begin{aligned}
&f'_\lambda\Big(e, (g_lp'_\varrho + (\sum_j p'g'_{w_j}p'_\varrho) + p'g'_\varrho, p'_\varrho)\Big)\cdot\\
&f'_{w_1}\Big((g'_\lambda p' + p'_\lambda g_r, p'_\lambda), (g_lp'_\varrho + (\sum_{j>1} p'g'_{w_j}p'_\varrho) + p'g'_\varrho, p'_\varrho)\Big)\cdot\\
\prod_{1<i<n} &f'_{w_i}\Big((g'_\lambda p' + (\sum_{j<i} p'_\lambda g'_{w_j}p') + p'_\lambda g_r, p'_\lambda), (g_lp'_\varrho + (\sum_{j>i} p'g'_{w_j}p'_\varrho) + p'g'_\varrho, p'_\varrho)\Big)\cdot\\
&f'_{w_n}\Big((g'_\lambda p' + (\sum_{j<n} p'_\lambda g'_{w_j}p') + p'_\lambda g_r, p'_\lambda), (g_lp'_\varrho + p'g'_\varrho, p'_\varrho)\Big)\cdot\\
&f'_\varrho\Big((g'_\lambda p' + (\sum_j p'_\lambda g'_{w_j}p') + p'_\lambda g_r, p'_\lambda), e\Big) \in \pi_1\mathcal{T}
\end{aligned}
$$

Now we can put everything under the product over i:

$$f'_\lambda\Big(e, (g_l p'_\varrho + (\sum_j p' g'_{w_j} p'_\varrho) + p' g'_\varrho, p'_\varrho)\Big)\cdot$$
$$\prod_i f'_{w_i}\Big((g'_\lambda p' + (\sum_{j<i} p'_\lambda g'_{w_j} p') + p'_\lambda g_r, p'_\lambda), (g_l p'_\varrho + (\sum_{j>i} p' g'_{w_j} p'_\varrho) + p' g'_\varrho, p'_\varrho)\Big)\cdot$$
$$f'_\varrho\Big((g'_\lambda p' + (\sum_j p'_\lambda g'_{w_j} p') + p'_\lambda g_r, p'_\lambda), e\Big) \in \pi_1 \mathcal{T}$$

Since the sum is commutative:

$$(f'_\lambda(g_l p'_\varrho, p'))\Big(e, ((\sum_j p' g'_{w_j} p'_\varrho) + p' g'_\varrho, p'_\varrho)\Big)\cdot$$
$$\prod_i ((p'_\lambda g_r, p') f'_{w_i}(g_l p'_\varrho, p'))\Big((g'_\lambda p' + (\sum_{j<i} p'_\lambda g'_{w_j} p'), p'_\lambda), ((\sum_{j>i} p' g'_{w_j} p'_\varrho) + p' g'_\varrho, p'_\varrho)\Big)\cdot$$
$$((p'_\lambda g_r, p') f'_\varrho)\Big((g'_\lambda p' + (\sum_j p'_\lambda g'_{w_j} p'), p'_\lambda), e\Big) \in \pi_1 \mathcal{T}$$

We let $g''_\sigma = p' g''_\sigma p'$, $g''_\varrho = p' g'_\varrho$ and $g''_\lambda = g'_\lambda p'$:

$$(f'_\lambda(g_l p'_\varrho, p'))\Big(e, ((\sum_j g''_j p'_\varrho) + g''_\varrho, p'_\varrho)\Big)\cdot$$
$$\prod_i ((p'_\lambda g_r, p') f'_{w_i}(g_l p'_\varrho, p'))\Big((g''_\lambda + (\sum_{j<i} p'_\lambda g''_{w_j}), p'_\lambda), ((\sum_{j>i} g''_{w_j} p'_\varrho) + g'_\varrho, p'_\varrho)\Big)\cdot$$
$$((p'_\lambda g_r, p') f'_\varrho)\Big((g''_\lambda + (\sum_j p'_\lambda g''_{w_j}), p'_\lambda), e\Big) \in \pi_1 \mathcal{T}$$

$$(f'_\lambda(g_l p'_\varrho, p'))\Big(e, (\prod_j (g''_j, e))(g''_\varrho, p'_\varrho)\Big)\cdot$$
$$\prod_i ((p'_\lambda g_r, p') f'_{w_i}(g_l p'_\varrho, p'))\Big((g''_\lambda, p'_\lambda)(\prod_{j<i}(g''_{w_j}, e)), (\prod_{j>i}(g''_{w_j}, e))(g'_\varrho, p'_\varrho)\Big)\cdot$$
$$((p'_\lambda g_r, p') f'_\varrho)\Big(((g''_\lambda, p'_\lambda)(\prod_j (g''_{w_j}, e)), e\Big) \in \pi_1 \mathcal{T}$$

So we can define

$$\tilde{h}(\sigma) = ((p'_\lambda g_r, p') f'_{w_i}(g_l p'_\varrho, p'), g''_\sigma, e),$$

$\tilde{\lambda}(v) = (f'_{\lambda(v)}(g_l p'_{\varrho(v)}, p'), g''_\lambda, p'_\lambda)$ and $\tilde{\varrho}(v) = ((p'_\lambda g_r, p') f'_\varrho, g''_\varrho, p'_\varrho)$. Then $(\tilde{h}, \tilde{\lambda}, \tilde{\varrho})$ recognizes L and maps to $\mathbf{T} \boxdot (\mathbf{SemiLin} \boxdot \mathbf{P}_<)$, since all semigroups where the one is the only unit is in $\mathbf{P}_<$. □

In logical terms the following lemma shows that we can modify a modulo quantifier to a modulo predicate for the innermost subformulas if we recognize a language where the syntactic semigroup is a perfect group. This is a very general idea that is independent of the logic class and could be easily extended to other semigroup classes.

Lemma 7.2 (Commutator lemma)**.** *Let* **T** *be a weakly closed class of typed semigroups and* (h, λ, ϱ) *be a non-uniform morphism to a semigroup in* $\mathbf{T} \boxdot (\mathbf{SemiLin} \boxdot \mathbf{P}_<)$ *that recognizes a language* L *where* G *is a subgroup of the syntactic semigroup, then there is a non-uniform morphism* $(\tilde{h}, \tilde{\lambda}, \tilde{\varrho})$ *to a semigroup of* $\mathbf{T} \boxdot (\mathbf{SemiLin} \boxdot \mathbf{P}_<)$*, such that* $|\pi_2(\tilde{h}(\Sigma))| = 1$*, that recognizes a language* $\tilde{L}$ *with* $\mathrm{syn}(\tilde{L}) = G'$*, where* G' *is the commutator group of* $\mathrm{syn}(L)$*.*

Proof. We can assume that $L = \eta^{-1}(e)$ by Lemma 6.7.

First we show that we can find word w'_g for every $g \in G$ such that $\eta(w'_g) = g$ and that all words have the same Parikh vector.

The subgroup G' is generated by all commutators of G, i.e. all elements $[a, b] = a^{-1}b^{-1}ab$ for $a, b \in G$. We define two words for every pair $a, b \in G$. Since η is surjective for every $a \in G$ there is a word w_a with $\eta(w_a) = a$. We let $u_{a,b} = w_{a^{-1}} w_{b^{-1}} w_a w_b$ and $v_{a,b} = w_{a^{-1}} w_a w_{b^{-1}} w_b$, then $\eta(u_{a,b}) = [a, b]$ and $\eta(v_{a,b}) = e$, where $u_{a,b}$ and $v_{a,b}$ have the same Parikh vector.

Unfortunately the set of commutators is not G' but only spans G'. Since the commutators span G' for each $g \in G'$, we can compose a word from the $u_{a,b}$, for $a, b \in G$, as above that maps to g, i.e. there are $(a_1^{(g)}, b_1^{(g)}), \ldots, (a_{n_g}^{(g)}, b_{n_g}^{(g)})$ with $\eta(\overline{u}_g) = g$ where $\overline{u}_g = u_{a_1^{(g)}, b_1^{(g)}} \ldots u_{a_{n_g}^{(g)}, b_{n_g}^{(g)}}$. Also we have a word with equal Parikh vector that maps to e: $\overline{v}_g = v_{a_1^{(g)}, b_1^{(g)}} \ldots v_{a_{n_g}^{(g)}, b_{n_g}^{(g)}}$ and $\eta(\overline{v}_g) = e$.

Now we let $w'_g = \overline{u}_g \prod_{g' \in G' \setminus \{g\}} \overline{v}_{g'}$, then $\eta(w'_g) = g \cdot e = g$. Also the Parikh vector for all w'_g is the same since

$$\begin{aligned} \#(w'_g) &= \#(u_g) + \sum_{g' \in G' \setminus \{g\}} \#(v_{g'}) = \\ &= \#(v_g) + \sum_{g' \in G' \setminus \{g\}} \#(v_{g'}) = \\ &= \sum_{g' \in G'} \#(v_{g'}) \end{aligned}$$

We now let $\Sigma' = G'$ and define a morphism $h' : \Sigma'^* \to T \boxdot (\mathbf{SemiLin} \boxdot \mathbf{P}_<)$ by $h'(\sigma) = h(w'_\sigma)$. We also adopt the prefix and suffix functions $\lambda'(v) = \lambda(\|v\|_1 \cdot \#(w'_e))$, and $\varrho'(v) = \varrho(\|v\|_1 \cdot \#(w'_e))$. Then (h', λ', ϱ') recognizes all words that are products equal to e, hence the syntactic monoid is G'. □

Finally on the logic side we show if we recognize a language with a neutral letter that uses the a mod -predicate can be also recognized without the mod -predicate in the innermost subformulas.

Lemma 7.3. *Let h be a morphism to* $\mathbf{T} \boxdot (\mathbf{SemiLin} \boxdot \mathbf{P_<})$ *with* $|\pi_2(h(\Sigma))| = 1$, *that recognizes L with neutral letter then there is a non-uniform morphism $\tilde{h}$ to* $\mathbf{T} \boxdot (\mathbf{Z} \boxdot \mathbf{P_<})$, *that recognizes the same language.*

Proof. Let h map to $(T, \mathfrak{T}, \mathcal{F}) \boxdot ((\mathbb{Z}^c, SemiLin) \boxdot P_<)$, where $(T, \mathfrak{T}, \mathcal{F}) \in \mathbf{T}$, $(\mathbb{Z}^c, SemiLin) \in \mathbf{SemiLin}$ and $P_< \in \mathbf{P_<}$. Let $\mathcal{S}$ be a type of the semigroup $(\mathbb{Z}^c, SemiLin)$ and $q_{\mathcal{S}}$ be the period of the semilinear set if there is an offset greater than $q_{\mathcal{S}}$ we choose a multiple of $q_{\mathcal{S}}$. So for any vector $v \in \mathbb{Z}^c$, we know that $\{z \mid zqv \in \mathcal{S}\}$ is a union of $0, \mathbb{Z}^-, \mathbb{Z}^+$. We choose a $q = \prod_{\mathcal{S}} q_{\mathcal{S}}$.

We let $h'(\sigma) = h(\sigma e^{q-1})$, $\lambda'(v) = \lambda(v + (q-1)\|v\|_1 E_e)$, $\varrho'(v) = \varrho(v + (q-1)\|v\|_1 E_e)$. Then (h', λ', ϱ') recognizes the same language.

Also a word w is in L iff

$$f'_\lambda\Big(e, (\prod_j (g'_{w_j}, e))(g'_\varrho, p'_\varrho)\Big) \cdot$$
$$\prod_i f'_{w_i}\Big((g'_\lambda, p'_\lambda)(\prod_{j<i}(g'_{w_j}, e)), (\prod_{j>i}(g'_{w_j}, e))(g'_\varrho, p'_\varrho)\Big) \cdot$$
$$f'_\varrho\Big((g'_\lambda, p'_\lambda)(\prod_j (g'_{w_j}, e)), e\Big) \in \pi_1 \mathcal{T}$$

by Lemma 3.35.

Fix a fixed position i, then the value of f'_{w_i} depends on clauses of the form:

$$g'_\lambda(e, p_f p'_\varrho) + (\sum_{j<i} g'_{w_j}(p'_\lambda, p_f p'_\varrho)) + g_f(p_\lambda, p_\varrho) +$$
$$+ (\sum_{j>i} g'_{w_j}(p'_\lambda p_f, p'_\varrho)) + g'_\varrho(p'_\lambda p_f, e) \in \pi_1 \mathcal{S}$$

Since $(g'_{w_\sigma}, e) = (g_{w_\sigma}, e)^q = (qg_{w_\sigma}, e)$ by the choice of q, we need to check if the sum above is greater, equal or less than 0. Hence the type $\mathbb{Z}^+, \mathbb{Z}^-, 0$ suffice and since $(\mathbb{Z}, \{\mathbb{Z}^+, \mathbb{Z}^-, 0\})$ divides $(\mathbb{Z}, \mathbb{Z}^+)^2$ by Lemma 3.15, we have a morphism to $(T, \mathfrak{T}, \mathcal{F}) \boxdot ((\mathbb{Z}, \mathbb{Z}^+)^{2c} \boxdot P_<) \in \mathbf{T} \boxdot (\mathbf{Z} \boxdot \mathbf{P_<})$. □

Theorem 7.4. *The typed semigroups of* wbpc(**SemiLin** $\boxdot$ $\mathbf{P_{<,succ}}$) *cannot recognize regular languages with non-solvable syntactic semigroups.*

$$L_{A_5} \notin \mathcal{L}(\text{wbpc}(\mathbf{SemiLin} \boxdot \mathbf{P_{<,succ}}))$$

Proof. Assume $A_5 \in$ wbpc(**SemiLin** $\boxdot$ $\mathbf{P_{<,succ}}$), then there is a semigroup $(S, \mathfrak{S}, \mathcal{E}) = (S', \mathfrak{S}', \mathcal{E}') \boxdot ((\mathbb{Z}^c, SemiLin) \boxdot P_{<,succ})$ with $(\mathbb{Z}^c, SemiLin) \in \mathbf{SemiLin}$ and $P_{<,succ} \in \mathbf{P_{<,succ}}$, that recognizes L_{A_5} by a non-uniform morphism (h, λ, ϱ). Choosing a typed semigroup of minimal block depth we can assume $(S', \mathfrak{S}', \mathcal{E}')$ does not recognize L_{A_5}.

Now we can apply Lemma 7.1 and get a non-uniform morphism to $(S', \mathfrak{S}', \mathcal{E}')' \boxdot ((\mathbb{Z}^c, SemiLin) \boxdot P_<)$ with $P_< \in \mathbf{P_<}$. This enables us to apply commutator Lemma

7.2 and get a non-uniform morphism to $(S', \mathfrak{S}', \mathcal{E}')' \boxdot ((\mathbb{Z}^c, SemiLin) \boxdot P_<)$ where $|\pi_2(h(\Sigma))| = 1$. Finally Lemma 7.3 modifies the morphism to map into $(S', \mathfrak{S}', \mathcal{E}')' \boxdot ((\mathbb{Z}, \mathbb{Z}^+)^c \boxdot P_<)$ for $(\mathbb{Z}, \mathbb{Z}^+)^c \in \mathbf{Z}$. But then reduction Lemma 6.8 gives a morphism to $(S', \mathfrak{S}', \mathcal{E}')$, a contradiction to the minimality of $(S, \mathfrak{S}, \mathcal{E})$. □

7.2 Arbitrary Unary Predicates

Now we turn our attention to arbitrary unary predicates. The proof idea is essentially the same except that we need to extend the power of non-uniform morphism in such a way that they can compute arbitrary unary predicates for free. But first we need to define the algebraic structure we are working with.

We let **N** be the variety generated by all typed semigroups $(\mathbb{N}, \mathfrak{S}, 1)$. This variety of typed semigroups recognizes exactly the variety generated by all unary languages. We let $\mathbf{P}_{\mathbf{arb-un}}$ be the variety generated by all typed semigroups in $(U_1, \{0,1\}, \{1\}) \boxdot \mathbf{N}$.

Please note that although we allow direct product in algebra this variety cannot recognize binary predicates, that cannot be decomposed into Boolean combinations of unary predicates. It is clear that $(\mathbf{1}, 1)$ is the only unit of these semigroups. Assume $L \subseteq \Sigma^+ \otimes \{x, y\}$ is recognized by a semigroup $(U_1, \{0,1\}, \{1\}) \boxdot (\mathbb{N}, \mathfrak{N}, 1)$. Then there is a type $\mathcal{S}$ such that $w_{x=i,y=j} \in L$ with $i < j$ iff $(\mathbf{1}, i-1)(g_x, n_x)(\mathbf{1}, j-i-1)(g_y, n_y)(\mathbf{1}, n-j) \in \mathcal{S}$. Which is by Lemma 3.35 the case iff $g_x(i-1, n-i-1+n_y)+g_y(j-2+n_x, n-j-1) \in \pi_1\mathcal{S}$. Since the image of g_x, g_y is finite, i.e. either 0 or $1 \in U_1$, and the constants n_x, n_y used are of a finite set, this is a Boolean combination of an unary predicate for x and an unary predicate for y. Hence $\mathbf{P}_{\mathbf{arb-un}}$ is a predicate group for the unary predicates.

We let $\mathbf{P}_{<,\mathbf{succ},\mathbf{arb-un}} = \mathbf{P}_< \vee \mathbf{P}_{\mathbf{succ}} \vee \mathbf{P}_{\mathbf{arb-un}}$ be the smallest variety that contains all unary predicates as well as order and successor. Also we let

$$\mathbf{W}_{\mathbf{arb-un}} = \text{wbpc}(\mathbf{SemiLin} \boxdot \mathbf{P}_{<,\mathbf{succ},\mathbf{arb-un}}),$$

$\mathbf{W}^1_{\mathbf{arb-un}} = \mathbf{SemiLin} \boxdot \mathbf{P}_{<,\mathbf{succ},\mathbf{arb-un}}$ and $\mathbf{W}^d_{\mathbf{arb-un}} = \mathbf{W}^{d-1}_{\mathbf{arb-un}} \boxdot (\mathbf{SemiLin} \boxdot \mathbf{P}_{<,\mathbf{succ},\mathbf{arb-un}})$. We show that we can add block with such a predicate group of $\mathbf{P}_{\mathbf{arb-un}}$ on the right without increasing the block depth.

Lemma 7.5. $\mathbf{W}^d_{\mathbf{arb-un}}$ *is closed under blocking with* $\mathbf{P}_{\mathbf{arb-un}}$ *from the right, i.e.* $\mathbf{W}^d_{\mathbf{arb-un}} \boxdot \mathbf{P}_{\mathbf{arb-un}} = \mathbf{W}^d_{\mathbf{arb-un}}$.

Proof. Let $P, P' \in \mathbf{P}_{\mathbf{arb-un}}$, then

$$\begin{aligned}
&((T, \mathfrak{T}, \mathcal{F}) \boxdot (\mathbf{SemiLin} \boxdot P)) \boxdot P' \preceq \\
&\preceq (T, \mathfrak{T}, \mathcal{F}) \boxdot (((\mathbf{SemiLin} \boxdot P) \boxdot P') \times P') \preceq \\
&\preceq (T, \mathfrak{T}, \mathcal{F}) \boxdot ((\mathbf{SemiLin} \boxdot ((P \boxdot P') \times P')) \times P') \preceq \\
&\preceq (T, \mathfrak{T}, \mathcal{F}) \boxdot ((\mathbf{SemiLin} \boxdot ((P \boxdot P') \times P')) \times (\mathbf{SemiLin} \boxdot P')) \preceq \\
&\preceq (T, \mathfrak{T}, \mathcal{F}) \boxdot \mathbf{SemiLin}^2 \boxdot ((P \boxdot P') \times P'^2)
\end{aligned}$$

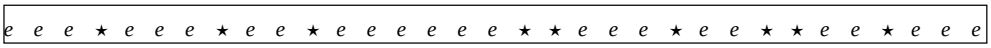


Figure 7.1: *A sample restriction*

Also $((P \boxdot P') \times P'^2)$ is a predicate semigroup hence in $\mathbf{P}_{\mathbf{arb-un}}$ and **SemiLin** is closed under direct products hence the claim follows. □

In the following we modify morphisms in a more extreme way, with the consequence that they do not recognize the same language anymore. But still we will have an infinite set of words, some in the language some outside, such that the modified morphism can still check for this subset if a word belongs to the language. We call this subset of words a restriction and formalize this in the following.

Definition 7.6 (Restriction)**.** A restriction is a map $\xi : \mathbb{N}^\Sigma \to (\Sigma \cup \{\star\})^+$. We say that a word w is in the restriction if for every i we have $w_i = \xi(\#(w))_i$ or $\xi(\#(w))_i = \star$. A non-uniform morphism (h, λ, ϱ) with restriction recognizes a language L if for every word w in the restriction we have $w \in L \iff \lambda(\#(w))h(w)\varrho(\#(w)) \in A$. A restriction is good if ξ maps to $e, \star$, where e is a neutral letter of L, and there is a constant c such that for each n_0 there is a $n > n_0$ with $\#_\star(\xi(n)) \geq c/n$.

We consider only restrictions that map positions to the neutral letter. In this way a word that is in the restriction belongs to the language iff the subword at the $\star$-positions belongs to the language.

Lemma 7.7. *Let $\xi : \mathbb{N}^\Sigma \to \{\star, 0, 1, \ldots, P\}$, where there are words of unbounded Parikh vector such that $\star$ appears at least $\|v\|_1/c$ times in $\xi(v)$, then for every number $l \in \mathbb{N}$, there is a subword $w \in \{\star, 0, 1, \ldots, P\}^+$, with $\#_\star(w) = l$ there are words of unbounded Parikh vector where w appears at least $\|v\|_1/(c^2 l^2 (P+1)^{cl})$ times non-overlapping in $\xi(v)$.*

Proof. First we show that in a word w are on average $\|v\|_1/(c^2 l^2)$ non-overlapping factor words with at least l $\star$'s.

Pick a word of length $n = c^2 l^2 n'(P+1)^{cl}$ with at least n/c $\star$'s and divide it in $m = cln'(P+1)^{cl}$ non-overlapping factor words of length cl. Assume there are less than $k = n'(P+1)^{cl}$ words that contain at least l stars. Then we can approximate the number of $\star$ from the top, since there are less than k words with l or more stars and the remaining words have less than l stars. So there are at most

$$\begin{aligned} k \cdot cl + m \cdot (l-1) = & \\ &= (n'(P+1)^{cl} - 1)(cl) + (cln'(P+1)^{cl})(l-1) = \\ &= cln'(P+1)^{cl} - cl + cl^2 n'(P+1)^{cl} - cln'(P+1)^{cl} = \\ &= cl^2 n'(P+1)^{cl} - cl = \\ &= n/c - cl \end{aligned}$$

$\star$'*s* in the word $\lightning$.

But then there are at most $(P+1)^{cl}$ different words of length cl, hence of the $n'(P+1)^{cl}$ words with l $\star$'*s*, there is a fixed word with l stars that appears at least n' times. □

Now we will use this lemma to find equal non-overlapping subwords, then we fix all $\star$-positions outside these subwords to e, and replace each of these subwords by a $\star$-position. In this way we construct a new morphism where, if you go back to the old morphism, each position has the same number of letters in the prefix/suffix independent of the choice of the letters at the $\star$-positions.

Theorem 7.8. $L_{A_5} \notin \mathcal{L}((\mathrm{FO}+\mathrm{MOD}+\mathrm{M}\widehat{\mathrm{A}}\mathrm{J})_2[\mathrm{reg}, \mathrm{arb}-\mathrm{un}])$

Proof. We will do induction on the block depth. We will use a restriction in the induction that restricts the words allowed in the input. The idea is that we can restrict the words allowed input in such a way that formulas of depth 1 with one free variable depend only on letters in a window around the free variable and the position of the variable. By choosing a suitable different morphism we can the replace these formulas by unary predicates and get a formula of lower depth.

Assume there is a morphism h into $(S, \mathfrak{S}, \mathcal{E}) \in \mathbf{T}' \boxdot (\mathbf{SemiLin} \boxdot \mathbf{P}_{<,\mathbf{succ},\mathbf{arb-un}})$, that recognizes L_{A_5} with the good restriction ξ. We will construct a morphism h' into $\mathbf{T}' \boxdot \mathbf{P}_{\mathbf{arb-un}}$ that recognizes L with a good restriction ξ'. Then by Lemma 7.5 we can reduce the block depth by one.

We assume the truth value of every predicate to the left and to the right of each star in the restriction is the same. This is possible since there are only finitely many predicates involved.

Now we will show that we can apply a similar trick as commutator Lemma 7.2. We choose w'_g as in commutator lemma, and let $l = |w'_g|$. We use Lemma 7.7 to get a word w. We may assume that w does not start nor end with a $\star$. We let w_g be the word w where the $\star$'*s* are replaced by the letters of w'_g. We let $\mu : (\Sigma \cup \{e'\})^+ \to \Sigma^+$ by $h(g) = w_g$ and $h'(e') = e$. We define a restriction ξ' such that the image of μ is in the restriction of ξ, due to Lemma 7.7 the restriction ξ' is good.

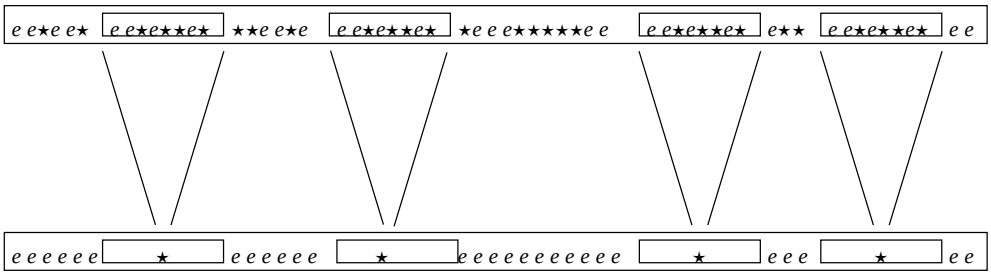


Figure 7.2: *Commutator Lemma in the presents restrictions*

Please note that morphism is not a typed morphism since $\pi_2^2(h(g)) \neq \pi_2^2(h(e'))$, but we can still consider this as a semigroup morphism. Also we know that

$\pi_2^2(h(g)) = \pi_2^2(h(e'))^x$, where $x \in \mathbb{N}$. But for $\mathbf{P}_{<,\mathbf{succ}}$ we know that the unit is an idempotent, and since we consider only words where the $\star$ positions are filled by letters of Σ, i.e. not e', for a fixed restriction we can replace the unary predicate semigroup such that we can assume $\pi_2^2(h(g)) = \pi_2^2(h(e'))$.

Let $(f'_\sigma, g'_\sigma, p') = h'(\sigma)$, then by Lemma 3.35, we add e' as a neutral letter to L, then we know $w \in L$ iff

$$\prod_i f'_{w_i}\Big(\prod_{j<i}(g'_{w_j}, p'), \prod_{j>i}(g'_{w_j}, p')\Big) \in \pi_1\mathcal{T}$$

We want to show that the value of f'_{w_i} depends only on the position i. This is clear since the prefix and suffix for a fixed position i have, independent of the input, the same Parikh vector and the unary predicates have the same value at every position except were the letter e' is, but this letter is fixed by the restriction. We need to show that for any constant (g'_f, p'_f) of f'_{w_i} the following product is always in the same type $\mathcal{S}$ independent of the word w chosen.

$$\prod_{j<i}(g'_{w_j}, p') \cdot (g'_f, p'_f) \cdot \prod_{j>i}(g'_{w_j}, p') \in \mathcal{S} \iff$$
$$\sum_{j<i} g'_{w_j}(p'^{j-1}, p'^{i-j-1}p'_f p'^{n-j}) + g'_f(p'^{i-1}, p'^{n-i}) + \sum_{j>i} g'_{w_j}(p'^{i-1}p'_f p'^{j-i-1}, p'^{n-j}) \in \pi_1\mathcal{S}$$

We what to show that the equation above depends only on j and w_j, but not on the $w_{j'}$ for $j' \neq j$. Assume we have two words w and w' such that $w_j = w'_j$, we will show that the equations above evaluate to the same types.

Please note that the value of the g'_{w_j} is independent of the input, and depends only on the order of i and j.

Now we have only the problem that we extended the alphabet by a new letter e'. But the letter e' appears exactly at some positions, so we can find a unary predicate that marks these positions, and replace e' by e. So on the algebraic side we need to enlarge the typed semigroup by a unary predicate semigroup. But because of Lemma 7.5, this does not increase the block depth.

So now since the type of the product above depends only on the position, we can interpret $\pi_2(h'(\Sigma^+))$ as a unary predicate semigroup, also by the block structure the is only one constant in the computation, hence we can replace $\pi_2(h'(\Sigma^+))$ by a semigroup of $\mathbf{P}_{\mathbf{arb-un}}$. As mentioned before applying Lemma 7.5 we get a morphism to $\mathbf{T}'$, completing the proof. □

7.3 Summary

First we extended the proof of the previous chapter to arbitrary commutative monoidal quantifiers. We could have still provide a geometric proof of this which is intuitive, but fails to be easily checkable. So this proof was done completely algebraically.

In Lemma 7.1 we showed that we do not need a successor predicate (in the innermost subformulas) if the language to be recognized has a neutral letter. We proceeded to show that for perfect groups we can replace the modulo quantifier by a modulo predicate (in the innermost subformulas), see commutator Lemma 7.2). And finally, in the presence of a neutral letter all modulo predicates (in the innermost subformulas) could be removed from the formula. So if a perfect group is recognized by a formula of (FO + MOD + $\widehat{\text{MAJ}}$)$_2$[reg], then we can assume that the innermost formulas have only the order predicate. This enabled us to use reduction Lemma 6.8 from the previous chapter to reduce the quantifier depth of the formula.

We concluded that a perfect group is not recognizable by a (FO+MOD+$\widehat{\text{MAJ}}$)$_2$[reg] formula.

In the second part, we replaced the proof idea of a fixed prefix and suffix by a restriction (Definition 7.6). Restrictions fix some of the inputs while they do not fix others. Restrictions are quite often used in circuit theory (e.g. [FSS81]), where some of the inputs of the circuit are fixed, but have hitherto not been defined for logic formulas or algebraic structures.

The restrictions allowed us to annihilate the power of the arbitrary unary predicates by considering only positions of inputs such that all predicates always have the same value. We could have still given a geometric interpretation of arbitrary unary predicates, but the intuition would lead to false results.

So in Theorem 7.8, we again applied the commutator Lemma 7.2 adopted to restrictions to show that no (FO + MOD + $\widehat{\text{MAJ}}$)$_2$[reg, arb − un] formula can recognize a perfect group.

7.4 Further Research

Allowing for only two variables imposes a severe restriction on the possible formulas, but we will argue that in the presence of prefix/suffix mappings or restrictions this is rather a restriction problem of the uniformity.

The construction of prefix/suffix mappings as given in Definition 6.2 or a polynomial version of a good restriction similar to Definition 7.6 would allow to overcome this by padding words with polynomial many *e*'s. Thus a circuit of polynomial size for any word is a circuit of linear size for a padded word for a sufficiently long padding. On the side of logic this is less intuitive, but with arbitrary predicates for a formula for a language L there is a two variable formula for L padded with suffixes of polynomial length.

In a more uniform version, i.e. allowing less predicates, this observation is not true any more. So what is a minimal set of predicates that this relation is true? More general given a set of quantifiers $\mathfrak{Q}$, what are the minimal sets of predicates $\mathfrak{P}$ such that $L \in \mathfrak{Q}[\mathfrak{P}]$ iff a polynomial padded version of L is in $\mathfrak{Q}_2[\mathfrak{P}]$?

We know that any set $\mathfrak{P}$ equipped with a tuple predicate suffices, but even among the tuple predicate tuples there are various possibilities to consider. The question is

therefore, can we extend the predicate set in this section such that it contains a tuple predicate? Be warned that this would already separate non-uniform TC^0 from NC^1.

Another interesting question is iff the Lemmas 7.1 and 7.3 can be modified to show a Crane Beach like result for $\widehat{\text{MA}}\text{J}_2$[reg], i.e. all languages with neutral letter in $\widehat{\text{MA}}\text{J}_2$[reg] are in $\widehat{\text{MA}}\text{J}_2$[<]. This might lead to a more straight-forward proof for the results in this chapter.

Chapter 8

Conclusion

In this thesis we focused on the interplay of logic, algebra and circuit theory. While their interaction was previously mainly studied for the case of regular languages we extend the focus to non-regular languages and show that similar connections can be proven. One of the big open questions in circuit theory is whether the classes TC^0 and NC^1 coincide, which is equivalent to the question whether $L_{A_5} \in TC^0$. In this thesis we established some separation results for subclasses of TC^0 that might finally result in a separation of TC^0 from NC^1.

We started by giving an algebraic characterization for arbitrary logic and circuit classes. Of course, any such characterization includes non-regular languages and hence finite semigroups are not sufficient, whereas infinite semigroups are too cumbersome. The key ingredient to this characterization was the typed semigroup which allowed for infinite semigroups, taming them by additional algebraic structures. The theory of typed semigroups coincides with the theory of finite semigroups in the finite case, additionally allowing for finer correspondences, and is a natural extension for the infinite case.

The known connections between algebra, logic and circuits were extended in a unifying proof. For this we needed to extend the block product to typed semigroups. One must exercise caution when defining the block product for an infinite structure, since the power of an infinite object is hard to handle. Using the block product principle extended to the infinite case, and to unbounded variables in logic as well, we gave a full picture of the connections between logic, circuits and algebra. It should be noted that this proof covers all known connections in the finite case in a much more general way.

Examining restrictions like two variables, the relations between two variable logic, weakly blocked semigroups and linear size circuits, were also generalized to arbitrary quantifiers and to arbitrary predicates using weakly blocked typed semigroups. In this case, too, the known proofs for these connections were entirely covered by our proof.

Therefore this paper gave an algebraic characterization for any class of logic in terms of typed semigroups. Having found this characterization we can also easily see

when an algebraic characterization in terms of finite semigroups exists for a given class of logic. On the algebraic side restrictions like a bounded number of variables or a bounded quantifier depth can also be modeled.

Having found this algebraic characterization we apply it to the case of majority logic or threshold circuits. The superordinate goal is to separate TC^0 from NC^1. Since a lot of effort was put into this by other researchers with limited success, e.g. TC^0 is still not separated from NP, it is not surprising that we do not accomplish a separation of TC^0 from NC^1. So we restrict ourselves to the subclass of MAJ[<] logic with only two variables.

Using the algebraic characterization via typed semigroup for this class of logic, which is basically a direct product of the integers that can be blocked weakly, we embed it into the Euclidean space. We then show by geometric means how to describe which words can be separated by a majority formula of depth one. It transpires that there is a correspondence between these formulas and hyperplanes in the vector space. It is intuitively obvious that with a constant number of hyperplanes an arbitrarily large cube cannot be split in such a way that all integer valued points are separated. Translated back to our logic this gives us two words, having the same truth value for all formulas of depth one.

Using induction we can give an upper bound on the power of $\widehat{\text{MAJ}}$[<] formulas with two variables. Using geometric intuition there seem to be many possible extensions, but in order to give clean proofs we will formalize them using algebra. Our algebraic concept allows us to enlarge the allowed quantifier set and predicate set to show that FO + MOD + $\widehat{\text{MAJ}}$[reg, arb − un] with two variables cannot recognize any language with a non-solvable syntactic semigroup.

The techniques introduced here differ completely from previous attempts, and it appears to be likely that they can be extended to bigger subclasses of TC^0 or even TC^0. We touch on some possible ways of extending the given proofs in the end of the chapters, and will now finalize this thesis by motivating some other open questions that could lead to a separation of TC^0 and NC^1 or at least from NP.

Open Questions

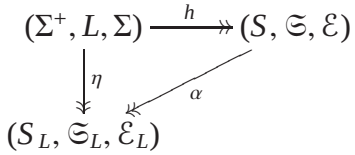
We will discuss some possible direction of further research here, involving the open question whether $TC^0 = NC^1$. Open questions that are more specific can be found at the end of each chapter.

The major open question remains whether $TC^0 = NC^1$, but we can state some intermediate steps on the way of showing such a separation result. Currently the largest class logic is $(\text{FO} + \text{MOD} + \widehat{\text{MAJ}})_2$[reg, arb − un] which we can show to not recognize L_{A_5}.

Intermediate open questions are therefore: Can we extend our proof to addition, or generally speaking is it true that $L_{A_5} \notin \mathcal{L}(\text{MAJ}_2[\text{reg}, +])$? Showing this would still be a smaller step than showing that $L_{A_5} \notin \mathcal{L}(\text{MAJ}[<])$, since the latter class of logic already

contains addition. Since we do not know whether majority logic with three variables is weaker than with four variables, an intermediate step could be examining whether $L_{A_5} \in \mathcal{L}(\mathrm{MAJ}_3[<])$ (we give an algebraic characterization for this in Section 4.5). Still unsolved but related is the question whether $\mathcal{L}(\mathrm{MAJ}_3[<])$ contains $\mathcal{L}(\mathrm{FO} + \mathrm{MOD}[<])$. To give a warning of problems that seem easy at first glance, we know that showing $\mathcal{L}(\mathrm{MAJ}_4[<]) \cap \mathrm{REG} \neq \mathcal{L}(\mathrm{MAJ}_5[<]) \cap \mathrm{REG}$ would imply $\mathrm{TC}^0 = \mathrm{NC}^1$.

In a perfect world. Theorem 5.12 states that TC^0 are the languages recognized by a morphism to block products of the integers. In the special case that $h(\Sigma^+)$ is a group, all problems become decomposed. Assume a group language L is recognized by a morphism h:



We assume here that $(S, \mathfrak{S}, \mathcal{E})$ is the image of h

The morphism α is a group morphism from the infinite group $(S, \mathfrak{S}, \mathcal{E})$ to the finite group $(S_L, \mathfrak{S}_L, \mathcal{E}_L)$. Since $(S, \mathfrak{S}, \mathcal{E})$ is a subgroup of a block product of integers it is solvable, and by the morphism α it follows S_L is also solvable, hence we can only recognize group languages of solvable groups.

So what destroys this simple situation when $h(\Sigma^+)$ is not a group? First of all α is not a group morphism any more, so the preimage of an element of S_L and its inverse are not a set of elements in S and their inverses, but could be unrelated. Since a block product of the integers with themselves already contains the free semigroup, we can pick nearly arbitrarily two elements in this free semigroup and map them to an element of S_L and its inverse. Secondly and even more frustratingly we do not even have a meaning of solvability for the semigroup $(S, \mathfrak{S}, \mathcal{E})$. Some direct possibilities come to mind, but all of them seem to fail.

If h recognizes a group language and $(S, \mathfrak{S}, \mathcal{E})$ is a subsemigroup of group $(T, \mathfrak{T}, \mathcal{F})$ we still do not know whether there is a subgroup of $(T, \mathfrak{T}, \mathcal{F})$ that recognizes the same language. We know some cases where this is impossible, for example if one allows all types on the block product of the integers, but this is more a construction of a counter-example than an application to the case of majority languages.

Even if this is not possible we could extend the alphabet Σ by $\overline{\Sigma}$, and let $h(\overline{\sigma}) = h(\sigma)^{-1}$. This construction however modifies the language being is recognized, and yet for simple languages we have no control about the new language recognized by the same type when not restricting the morphism h heavily.

As a last thought in this direction, it is not really necessary if we have the notion of an "inverse", it would suffice to have a "commutator" notion in our subsemigroups, because we would have the notion of "solvable". There are many ways to redefine "commutators" for the blocked integers in such a way that they remain "solvable",

but again we need to extend the image of h, which modifies the language in an unforeseeable way.

Still this might be a viable way and another step towards a better understanding of the low complexity classes. Another promising continuation would be applying typed semigroups to other circuit classes and reproving known results like parity not in AC^0 algebraically to gain deeper insight, or tackling problems like L_{AND} in CC^0 by approaching it with the uniform circuit classes of our algebra.

Appendix A

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Appendix B

Bibliography

[Alm95] Jorge Almeida. *Finite Semigroups and Universal Algebra*. World Scientific, Singapore, 1995.

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[BCST92] David A. Mix Barrington, Kevin J. Compton, Howard Straubing, and Denis Thérien. Regular languages in nc^1. *J. Comput. Syst. Sci.*, 44(3):478–499, 1992.

[BIL+05] David A. Mix Barrington, Neil Immerman, Clemens Lautemann, Nicole Schweikardt, and Denis Thérien. First-order expressibility of languages with neutral letters or: The crane beach conjecture. *J. Comput. Syst. Sci.*, 70(2):101–127, 2005.

[BIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990.

[BKM07] Christoph Behle, Andreas Krebs, and Mark Mercer. Linear circuits, two-variable logic and weakly blocked monoids. In *MFCS*, pages 147–158, 2007.

[BKRa] Christoph Behle, Andreas Krebs, and Stephanie Reifferscheid. Regular languages definable by majority quantifiers with two variables. draft.

[BKRb] Christoph Behle, Andreas Krebs, and Stephanie Reifferscheid. Separating a subclass of TC^0 from TC^1. draft.

[BL06] Christoph Behle and Klaus-Jörn Lange. FO[<]-uniformity. In *IEEE Conference on Computational Complexity*, pages 183–189, 2006.

[BST90] David A. Mix Barrington, Howard Straubing, and Denis Thérien. Non-uniform automata over groups. *Inf. Comput.*, 89(2):109–132, 1990.

[BT88] David A. Mix Barrington and Denis Thérien. Finite monoids and the fine structure of NC^1. *J. ACM*, 35(4):941–952, 1988.

[Büc60] J.R. Büchi. Weak second-order arithmetic and nite automata. *Z. Math. Logik Gundlag. Math. 6*, 6:66–92, 1960.

[CFL85] Ashok K. Chandra, Steven Fortune, and Richard J. Lipton. Unbounded fan-in circuits and associative functions. *J. Comput. Syst. Sci.*, 30(2):222–234, 1985.

[CPP93] Joëlle Cohen, Dominique Perrin, and Jean-Eric Pin. On the expressive power of temporal logic. *J. Comput. Syst. Sci.*, 46(3):271–294, 1993.

[CPS06] Laura Chaubard, Jean-Eric Pin, and Howard Straubing. First order formulas with modular ppredicates. In *LICS*, pages 211–220, 2006.

[Eil76] Samuel Eilenberg. *Automata, Languages and Machines, Vol. A+B*. Academic Press, 1976.

[EVW97] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. In *LICS*, pages 228–235, 1997.

[FSS81] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. In *FOCS*, pages 260–270, 1981.

[GL84] Yuri Gurevich and Harry R. Lewis. A logic for constant-depth circuits. *Information and Control*, 61(1):65–74, 1984.

[Hås87] Johan Håstad. Computational limitations for small depth circuits. *MIT Press, Cambridge, MA*, 1987.

[HMP+87] András Hajnal, Wolfgang Maass, Pavel Pudlák, Mario Szegedy, and György Turán. Threshold circuits of bounded depth. In *FOCS*, pages 99–110, 1987.

[HMP+93] András Hajnal, Wolfgang Maass, Pavel Pudlák, Mario Szegedy, and György Turán. Threshold circuits of bounded depth. *J. Comput. Syst. Sci.*, 46(2):129–154, 1993.

[How95] John M. Howie. *Fundamentals of Semigroup Theory*. Clarendon Press, Oxford, 1995.

[Imm87] Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987.

[Kam68] Johan Anthony Willem Kamp. Tense logic and the theory of linear order. *Ph.D. thesis, University of California, Berkeley*, 1968.

[KLPT06] Michal Koucký, Clemens Lautemann, Sebastian Poloczek, and Denis Thérien. Circuit lower bounds via ehrenfeucht-fraisse games. In *IEEE Conference on Computational Complexity*, pages 190–201, 2006.

[KLR05] Andreas Krebs, Klaus-Jörn Lange, and Stephanie Reifferscheid. Characterizing TC^0 in terms of infinite groups. In *STACS*, pages 496–507, 2005.

[KLR07] Andreas Krebs, Klaus-Jörn Lange, and Stephanie Reifferscheid. Characterizing TC^0 in terms of infinite groups. *Theory Comput. Syst.*, 40(4):303–325, 2007.

[KPT05] Michal Koucký, Pavel Pudlák, and Denis Thérien. Bounded-depth circuits: separating wires from gates. In *STOC*, pages 257–265, 2005.

[Lan04] Klaus-Jörn Lange. Some results on majority quantifiers over words. In *IEEE Conference on Computational Complexity*, pages 123–129, 2004.

[LMSV01] Clemens Lautemann, Pierre McKenzie, Thomas Schwentick, and Heribert Vollmer. The descriptive complexity approach to LOGCFL. *J. Comput. Syst. Sci.*, 62(4):629–652, 2001.

[Mac98] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1998.

[MP71] Robert McNaughton and Seymour Papert. *Counter-free automata. With an appendix by William Henneman.* Research Monograph No.65. Cambridge, Massachusetts, and London, England: The M. I. T. Press. XIX, 163 p., 1971.

[MTV08] Pierre McKenzie, Michael Thomas, and Heribert Vollmer. Extensional uniformity for boolean circuits. In *CSL*, 2008. to appear.

[Pin86] Jean-Eric Pin. *Varieties of formal languages*. Plenum, London, 1986.

[Pin98] Jean-Eric Pin. Positive varieties and infinite words. In *LATIN*, pages 76–87, 1998.

[PW97] Jean-Eric Pin and Pascal Weil. Ponynominal closure and unambiguous product. *Theory Comput. Syst.*, 30(4):383–422, 1997.

[Raz87] A. Razborov. Lower bounds for the size of circuits of bounded depth with basis $\{\vee, \oplus\}$. *Math. notes of the Academy of Sciences of the USSR*, 41:333–338, 1987.

[RS06] Amitabha Roy and Howard Straubing. Definability of languages by generalized first-order formulas over (N,+). In *STACS*, pages 489–499, 2006.

[RT89] John L. Rhodes and Bret Tilson. The kernel of monoid morphisms. *J. Pure Applied Alg.*, 62:27–268, 1989.

[Ruh99] Matthias Ruhl. Counting and addition cannot express deterministic transitive closure. In *LICS*, pages 326–334, 1999.

[Sak76] Jacques Sakarovitch. An algebraic framework for the study of the syntactic monoids application to the group languages. In *MFCS*, pages 510–516, 1976.

[Sch65] Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[ST02] Howard Straubing and Denis Thérien. Weakly iterated block products of finite monoids. In *LATIN*, pages 91–104, 2002.

[ST03] Howard Straubing and Denis Thérien. Regular languages defined by generalized first-order formulas with a bounded number of bound variables. *Theory Comput. Syst.*, 36(1):29–69, 2003.

[Str92] Howard Straubing. Circuit complexity and the expressive power of generalized first-order formulas. In *ICALP*, pages 16–27, 1992.

[Str94] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.

[Str02] Howard Straubing. On logical descriptions of regular languages. In *LATIN*, pages 528–538, 2002.

[STT95] Howard Straubing, Denis Thérien, and Wolfgang Thomas. Regular languages defined with generalized quanifiers. *Inf. Comput.*, 118(2):289–301, 1995.

[TT05] Pascal Tesson and Denis Thérien. Restricted two-variable sentences, circuits and communication complexity. In *ICALP*, pages 526–538, 2005.

[TT07] Pascal Tesson and Denis Thérien. Logic meets algebra: the case of regular languages. *CoRR*, abs/cs/0701154, 2007.

[TW98] Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *STOC*, pages 234–240, 1998.

[TW04] Denis Thérien and Thomas Wilke. Nesting until and since in linear temporal logic. *Theory Comput. Syst.*, 37(1):111–131, 2004.

[Yao85] Andrew Chi-Chih Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *FOCS*, pages 1–10, 1985.

Lebenslauf

21.11.1978	geboren in Stuttgart

Schulen

1985-1989	Grundschule, Filderstadt
1989-1998	Gymnasium Königsbrunn
1995-1996	Galena High School, Reno, Nevada (USA)
Juni 1996	Abschluss mit dem High School Degree, GPA 4.0 (sehr gut)
Juni 1998	Abschluss mit dem Abitur, Note 1,8

Studium

1999-2002	Studium der Informatik an der Universität in Tübingen
2000-2005	zusätzliches Studium der Mathematik.
2.11.2001	Vordiplom Informatik, Note: sehr gut
2.11.2001	Vordiplom Mathematik, Note: gut
16.7.2002	Abschluss als Diplom-Informatiker mit Auszeichnung
seit Okt. 2002	Wissenschaftlicher Angestellter an der Universität Tübingen
Dez. 02-Aug. 03	Auslandsaufenthalt an der Universität of Oregon in Eugene als Graduate Student und Teaching Assistant
17.10.2005	Abschluss als Diplom-Mathematiker mit Auszeichnung