Gymnasium Bäumlihof

Maturaarbeit

# Theoretical Informatics: Formal languages and Descriptive Complexity

## A study of the connection of first-order logic and context-sensitive languages

Written by:
Yaël Arn, 4A

Supervisor:
Aline Sprunger

Second Examiner:
Bernhard Pfammatter

9th October 2024, 4058 Basel

# Foreword

Some years ago, the Lego Mindstorms sparked my interest in informatics and programming. By attending some courses at the Phænovum in Lörrach, I was able to learn how to program using Java, my first text-based programming language. At that time, I was planning to work at Boston Dynamics, as I loved being able to physically see what I had achieved. But then came the RoboCup robot [Rob97]. It was meant to be a rolling robot for playing football on a miniature playing field. After two years, we were still unable to follow the ball because the Corona pandemic prevented us from working on site together. Also, and to a greater extent, it didn't work because the hardware never did what it was meant to, and we passed interminable hours just trying to make it roll forward. As may have become apparent, I got tired of it and decided to move on to something which didn't include too much hardware and was more abstract.

So I decided to participate in the Swiss Olympiad in Informatics, which organizes national programming contests and selects various teams for international Olympiads. There, I got and still get quite a lot of interesting problems which I love to solve, found like-minded friends and was able to participate in some international competitions. Over the years, I began to notice that I enjoy solving tasks theoretically way more than implementing them. That is also something that got reflected in my competition scores, where I often came out knowing I solved a lot in theory, but failed to get the points. Some internships at informatics firms confirmed that actual programming is still too concrete for me.

Now let's get more abstract. Thanks to the "Schülerstudium", I attended a course on the mathematical background of computer science [HR22] and one on the theory of computer science [Rög23]. There, I learned more about Complexity theory, computability, logic and the Chomsky Hierarchy. The way in which proofs could be made to hold for every problem with certain properties has fascinated me ever since. Further, logic is a tool which captures mathematical reasoning, and can thus be seen as a formalization of every "logical" thought we have. Also, I don't have to bother with implementation any more.

Using books, I began to inform myself more, and found out about the domain of Descriptive Complexity. This domain relates different kinds of logic to different classes of problems in computer science. So I knew I wanted to do my Matura project in this domain, but had difficulties finding an open question which seemed approachable. I asked many people if they had a more exact idea what I could do. In the end, a friend at the informatics Olympiad asked *ChatGPT*, which told me I could study the connection between the Chomsky Hierarchy and Descriptive Complexity. Refining this proposition a bit, I came up with (almost) the current question of relating logic and context-

sensitive languages. Then, I found out that a characterization using second-order logic already exists, so I added “first-order logic” to the question.

# Table of Contents

# 1. Introduction

In our daily lives, we are in contact with various kinds of algorithms at all times. Searching on Google, sending texts, asking questions to AI chatbots, searching for the fastest route with a GPS; all of these consume resources in energy, storage space and time. According to a report from 2021, 3.7 % of global carbon emissions come from the IT domain, with an upward tendency. This is similar to that of the airline industry [Cli21]. At this scale, it is thus vitally important to understand and find out if the resource consumption can be reduced.

To find out how we can improve, we first need to understand how computers work, what we can compute, and why solving some problems is more difficult than solving others. The field of study that investigates this is called Complexity theory. This is done by abstraction of computational models and of the problems themselves. It is then often possible to find classes of similar problems which allow for generalizations. However, it has proven to be very hard to find proofs of either optimality of algorithms or the separations of complexity classes. One of the emblematic open problems is the P versus NP question, which we will go further into in appendix B.2.8. Nevertheless, some significant results concerning the complexity of problems on average in the real world and most importantly in cryptography have been made.

One of the subfields of Complexity theory is Descriptive Complexity. It relates mathematical logic to different complexity classes. This allows us to get new insights into the underlying structure of problems of certain classes. The other main field present in this work, formal languages, is concerned with the abstraction of computational problems as sets, which is often helpful for proofs. This then allows us to define multiple formalisms which describe these sets. Our formal language class of focus, context-sensitive languages, is part of the "Chomsky Hierarchy" introduced by the famous linguist Noam Chomsky.

Towards the end of the $20^{\text{th}}$ century, many equivalences were proven between fragments and extensions of various logics to complexity classes, including all classes in the Chomsky hierarchy. A great summary of all the results is found in Neil Immermans paper "Languages that capture complexity classes" [Imm87].

In this work, we will first introduce the relevant theory. After this, a personal study of connections between first-order logic and context-sensitive languages is presented.

Both chapter 2 and chapter 3 present the most important theory of formal languages and Descriptive Complexity, respectively. Additionally, in section 3.4 full proofs concerning equivalence of context-sensitive languages with logics and automata are presented. These proofs are written in a way trying to show the motivation behind certain crucial steps. Further, some details have been omitted for the sake of the simplicity and length of this work. For the same purpose, many proofs, explanation, and examples for other classes in the Chomsky hierarchy were moved to appendix B.

After this introduction to the material, we go on in chapter 4 with my personal studies about connections between first-order logic and context-sensitive language. In this process, different paths that I tried are presented and investigated. Towards the end of the chapter, the direct connection to context-sensitive languages fades and the objects of study becomes more related to Savitch's theorem, which plays a great role in the theory of space-bounded computation. The most complicated proof is given in section 4.5 and concerns a simulation of alternating Turing machines using iterative logic. This divergence from the original working question was due to the hope that researching other, more loosely connected problems would ultimately help finding a solution for the core problem. Also, I tried to pursue ideas I had instead of trying paths where I did not make any progress.

Finally, in chapter 5 we reflect on what was accomplished in this work. A review about the working process is presented, and finally we discuss further possibilities of research.

For the relevant mathematical background, appendix A contains the basic definitions of Set theory, first and second-order logic, and Turing machines.

A full collection of description, proofs, techniques, and context information about Descriptive Complexity and languages in the Chomsky Hierarchy is given in appendix B.

The entire work is written in English because all mathematical literature is in that language.

# 2. Formal languages

## 2.1 Definition

In informatics, we often get an input as a string of characters, and want to compute some function on it. In Complexity theory, we mostly focus on decision problems where we only want to find out if some input fulfils some given property[1]. To formalize this, there is the concept of formal languages. The following definitions are taken from the lecture Theory of Computer Science [Rög23]. For the mathematical background, refer to Appendix A.

**Definition 2.1** (Alphabet)**.** An alphabet $\Sigma$ is a finite set of symbols

**Definition 2.2** (Word)**.** A word over some alphabet $\Sigma$ is a finite sequence of symbols from $\Sigma$. We denote $\varepsilon$ as the empty word, $\Sigma^*$ as the set of all words over $\Sigma$, $xy$ as the concatenation of $x$ and $y$, $x^n$ as the concatenation of $x$ with itself $n$ times, and $|w|$ as the number of symbols in $w$.

**Definition 2.3** (Formal language)**.** A formal language is a set of words over some alphabet $\Sigma$, or equivalently a subset of $\Sigma^*$

For any computational decision problem, we can reformulate it as the problem of deciding if the input word is contained in the formal language consisting of all words which have the required property.

## 2.2 Chomsky Hierarchy

One of the multiple ways to categorize formal languages was invented by Avram Noam Chomsky, a modern linguist. It is based on the complexity of defining the language using formal grammars, which are a finite representation of formal languages (which can be infinite in the general case).

### 2.2.1 Grammars

A grammar can informally be seen as a set of rules telling us how to generate all words in a language.

**Definition 2.4** (Grammar)**.** A grammar is a 4-tuple $\langle V, \Sigma, R, S\rangle$ consisting of

$V$ The set of non-terminal symbols

$\Sigma$ The set of terminal symbols

$R$ A set of rules, formally over $(V \cup \Sigma)^* V (V \cup \Sigma)^* \times (V \cup \Sigma)^*$

$S$ The start symbol from the set $V$

[1] This can be shown to be equivalent to computing functions

The non-terminal symbols in $V$ are symbols that are not in $\Sigma$ and exist for the purpose of steering the process of word generation. They do not appear in any of the final words generated by the grammar. Further, the rules dictate that there must be at least one non-terminal symbol on the left-hand side of any production rule. This is because $(V \cup \Sigma)^* V (V \cup \Sigma)^*$ means all words having one symbol in $V$ enclosed by words in $(V \cup \Sigma)^*$, which are all words consisting of symbols in $V$ and $\Sigma$. For better readability, we write rules in the form $a \to b$ instead of $\langle a, b \rangle$.

To generate the words, we have the concept of derivations.

**Definition 2.5** (Derivation)**.** First, we can define one derivation step.

We say $u'$ can be derived from $u$ if

- $u$ is of the form $xyz$ and $u'$ is of the form $xy'z$ for some words $x, y, y', z \in (V \cup \Sigma)^*$
- there exists a rule $y \to y'$ in $R$

We say that a word is in the *generated language* of a grammar if it consists only of symbols in $\Sigma$ and can be derived in a finite number of steps from $S$.

**Example 2.1.** Consider the grammar $\langle \{S\}, \{a, b\}, R, S \rangle$ with

$$R = \Big\{ S \to aSb, \quad S \to \varepsilon \Big\}$$

The generated language for this grammar is $\{\varepsilon, ab, aabb, \dots\} = \{a^n b^n \mid n \in \mathbb{N}_0\}$

Now that we have a tool to describe some infinite languages using a finite description, we can further differentiate the complexity of a language by the minimum required complexity of the rules in any formal grammar describing that language. In the main section, we will only present the context-sensitive languages, as they are important for later chapters. In appendix B.1 explanations for regular languages (appendix B.1.1), context-free languages (appendix B.1.2) and recursive languages (appendix B.1.3) are provided.

### 2.2.2 Context-Sensitive Languages

The most important category of languages for this work can be defined by multiple equivalent restrictions on the grammars.

One restriction is that all rules have to be of the form $\alpha\beta\gamma \to \alpha\varphi\gamma$ with $\alpha, \gamma \in (\Sigma \cup V)^*$, $\beta \in V$ and $\varphi \in (\Sigma \cup V)^+$. This means that only the non-terminal is allowed to change. Additionally, if $S$ is the start variable and never occurs on the right-hand side of any rule, we may include $S \to \varepsilon$.

Equivalently, we can require that for any rule $u \to v$ we have $|u| \leq |v|$. This means that applying any rule will make the result longer. Again, we also allow the special rule $S \to \varepsilon$ if $S$ does not occur on the right-hand side of any rule in $R$. These grammars are called noncontracting.

The last, most useful restriction for proofs is the Kuroda normal form [Pet22], where all rules have one of the following structures:

- $A \to BC$

- $AB \to CB$
- $A \to a$
- $S \to \varepsilon$ if $S$ is the start symbol and does not occur on any right-hand side of a rule

where $A, B, C, S \in V$ and $a \in \Sigma$.

**Example 2.2.** Consider the grammar $\langle \{S, B\}, \{a, b, c\}, R, S \rangle$ with

$$R = \begin{Bmatrix} S \to abc, & S \to aSBc, \\ cB \to Bc, & bB \to bb \end{Bmatrix}$$

It generates the language $a^n b^n c^n$ for $n \in \mathbb{N}_1$ and is noncontracting.

The corresponding formalism for these languages is the linearly bounded nondeterministic Turing machine, which can only write on the tape cells that contained the input word in the beginning. This and an equivalent extension of second-order logic will be proven in appendix B.2.6.

# 3. Descriptive Complexity

## 3.1 Aims

In mathematics, abstraction is one of the most important tools, as it enables us to make general statements and prove them for all the concrete instances of a concept. Formal logic takes this even further and makes it possible to abstract mathematical thought itself. In computer science, we are always interested in the amount of resources needed to compute a certain function or solve a certain problem, speaking in terms of time and storage space. By focusing on decision problems[1], we can define a logical characterization of a problem as a formula $\varphi$ which is true if and only if a structure satisfies the required properties. Different forms of logic have the power to describe different classes of problems. By looking at the complexity of formulas which are needed to describe problems in terms of relations, operators, variables and other metrics, we can often find remarkably natural classes of logic corresponding to classes of problems.

Using these results, many insights into the underlying structure of real-world problems can be made, which in turn can give us better ways to tackle them. Further, Descriptive Complexity has applications in database theory, computer aided verification, and proofs.

## 3.2 Tools

As always, we first need to present some tools and techniques which will be used later for proofs. In this section, we will only present Complexity theory. In appendix B.2.1 and appendix B.2.2, we also present Ehrenfeucht-Fraïssé games and first-order reductions. These tools are not needed for chapter 4, but using them was tried extensively during the research phase. Definition are again taken from [Rög23] and [Imm99].

### 3.2.1 Complexity Theory

Complexity theory is the study of the resources, measured mostly in time and space, needed to compute certain classes of problems[2]. As different computers can handle tasks at various speeds, a notation which only considers the asymptotic use of resources is used.

**Definition 3.1** (Big-O notation)**.** Let $f, g$ be functions $f, g : \mathbb{N} \to \mathbb{R}_+$.

[1] Any problem can be reduced to boolean queries, for example by having a boolean query meaning "the i$^{th}$ bit of an encoding of the answer is 1"

[2] The specific model of computation is not important as all give almost the same results. We will assume Turing machines.

We say that $f(n) \in \mathcal{O}(g(n))$ if there exists positive integers $n_0, c$ such that for all $n \geq n_0$ we have

$$f(n) \leq c \cdot g(n)$$

Complexity classes can be defined as the set of all the problems for which there exists a Turing machine satisfying some bounds that computes the answer to the problem. Now, we will define some common complexity classes.

**Definition 3.2** (DTIME[$\mathcal{O}(t(n))$])**.** We say that a decision problem is in DTIME[$\mathcal{O}(t(n))$] if there exists a deterministic Turing machine that takes a maximum of $f(n)$ steps on any input of size $n$ and $f(n) \in \mathcal{O}(t(n))$.

**Definition 3.3** (P)**.** We say that a decision problem is in P if there exists a polynomial $q$ such that the problem is in DTIME[$\mathcal{O}(q(n))$]

**Definition 3.4** (DSPACE[$\mathcal{O}(t(n))$])**.** We say that a decision problem is in DSPACE[$\mathcal{O}(t(n))$] if there exists a deterministic Turing machine that visits a maximum of $f(n)$ tape cells on any input of size $n$ and $f(n) \in \mathcal{O}(t(n))$.

**Definition 3.5** (PSPACE)**.** We say that a decision problem is in PSPACE if there exists a polynomial $q$ such that the problem is in DSPACE[$\mathcal{O}(q(n))$]

We can go do the same for nondeterministic Turing machines, and get the corresponding complexity classes NTIME, NP, NSPACE, NPSPACE. There, we always take the maximum of tape cells and steps over any computation branch.

The complexity class P has a special meaning for computer scientists, as the problems in P are deemed "feasible" on modern computers.

## 3.3 Important Results

In this section, we will only introduce Savitch's theorem. In appendix B.2.3, we also present the space hierarchy theorem.

### 3.3.1 Quadratic simulation of nondeterministic Turing machines using deterministic Turing machines

This result is part of Savitch's theorem, which introduces alternating Turing machines as an intermediate step. These Turing machines are a generalization of nondeterministic Turing machines, which can be seen as machines taking the "or" over all their computation paths.

**Definition 3.6** (Alternating Turing Machine)**.** An alternating Turing machine is a Turing machine with two types of states: existential and universal. The accepting conditions change compared to a nondeterministic Turing machine and depend on the current state of the machine. If it is in an existential state, the current configuration is accepting if and only if *at least one* of the directly reachable configurations is accepting. If the machine is in a universal state, the current configuration is accepting if and only *all* the directly reachable configurations are accepting.

In addition to taking the "or" over its children, an alternating Turing machine can also take the "and".

We define ATIME[$\mathcal{O}(t(n))$] and ASPACE[$\mathcal{O}(t(n))$] analogously to NTIME[$\mathcal{O}(t(n))$] and NSPACE[$\mathcal{O}(t(n))$].

We can proceed to the (quite technical) proof of Savitch's theorem using the technique presented in [Imm99].

**Theorem 3.1** (Savitch's Theorem)**.** *For all space constructible functions with $t(n) \geq \log n$ we have*

$$\text{NSPACE}[\mathcal{O}(t(n))] \subseteq \text{ATIME}[\mathcal{O}(t(n)^2)] \subseteq \text{DSPACE}[\mathcal{O}(t(n)^2)]$$

*Proof.* We start with the first inclusion, NSPACE[$\mathcal{O}(t(n))$] $\subseteq$ ATIME[$\mathcal{O}(t(n)^2)$]. We need to show that any NSPACE[$\mathcal{O}(t(n))$] Turing machine can be simulated by a ATIME[$\mathcal{O}(t(n)^2)$] alternating Turing machine. Let $N$ be a NSPACE[$\mathcal{O}(t(n))$] Turing machine. Without loss of generality, we assume that $N$ clears its tape after accepting and goes back to the first cell.

Consider $G_w$, the computation graph of $N$ on input $w$. This graph consists of configurations as vertices and directed edges from each configuration to all configurations that are directly reachable from it. We see that $N$ accepts $w$ if and only if there is a path from the start configuration $s$ to the accepting configuration $t$ in $G_w$. We now present a routine $P(d, x, y)$ which asserts that there is a path of length at most $2^d$ from vertex $x$ to $y$. Inductively, we can define $P$ as follows:

$$P(d, x, y) = (\exists z)(P(d-1, x, z) \wedge P(d-1, z, y))$$

This formula asserts that there exists a middle vertex $z$ for which there is a $2^{d-1}$ path from $x$ to $z$ and from $z$ to $y$. The base case for $d = 0$ is to check if we can reach $x$ from $y$ by a transition in the transition table of $N$. Using an alternating Turing machine, we can evaluate the formula using existential states to find the middle vertex $z$, and then a universal state choosing which one of the two new paths we check.

For the runtime analysis, we proceed as follows. It takes $\mathcal{O}(t(n))$ time to write down the middle vertex $z$, as each configuration includes the tape contents, which have length $\mathcal{O}(t(n))$. Further, we then evaluate one of the new shorter paths $P(d-1, z, y)$. By induction, we find that we need $\mathcal{O}(d \cdot t(n))$ time to compute $P(d, x, y)$. There are only $2^{\mathcal{O}(t(n))}$ possible configurations, we get that the initial $d$ is also in $\mathcal{O}(t(n))$, and thus our total runtime is $\mathcal{O}(t(n) \cdot t(n)) = \mathcal{O}(t(n)^2)$. For the second inclusion,

we need to simulate an ATIME[$\mathcal{O}(s(n))$] machine $A$ using a DSPACE[$\mathcal{O}(s(n))$] machine (here, we substituted $s(n)$ for $t(n)^2$). Again, we consider the computation graph of $A$ of on input $w$. This graph has depth $\mathcal{O}(s(n))$ and size $2^{\mathcal{O}(s(n))}$.

We can systematically search this computation graph to get our answer. This is done by keeping a string of choices $c_1 c_2 \dots c_r$ of length $\mathcal{O}(s(n))$ made until this point. Each choice takes up $\mathcal{O}(1)$ place, as the number of transitions from one state is bounded by a constant depending only on $A$. Note that this uniquely determines which state we are in.

Now, we can find the answer recursively. If we are in a halt state, we report this back to the

previous state. In an existential state, we simulate its children, and if we get a positive result from one of them, we also report a positive result. In a universal state, we simulate its children, and if we get a positive result from all of them, we report a positive result.

Overall, we use only $\mathcal{O}(s(n))$ space to simulate $A$.

Thus, the second part of the theorem follows and by the transitivity of $\subseteq$ we have $\text{NSPACE}[\mathcal{O}(t(n))] \subseteq \text{DSPACE}[\mathcal{O}(t(n)^2)]$. □

We do not know if the containment is strict for any of the inclusions. Neither is it known if the quadratic overhead for simulating nondeterministic space is optimal. From this theorem, we also get the following interesting corollary.

**Corollary 3.1.1.** *We have* PSPACE = NPSPACE*.*

*Proof.*

$$\begin{aligned}
\text{NPSPACE} &= \bigcup_{k\in\mathbb{N}}^{\infty} \text{NTIME}[\mathcal{O}(n^k)] \\
&\subseteq \bigcup_{k\in\mathbb{N}}^{\infty} \text{DTIME}[\mathcal{O}(n^{2k})] \\
&= \bigcup_{k\in\mathbb{N}}^{\infty} \text{DTIME}[\mathcal{O}(n^k)] \\
&= \text{PSPACE} \\
&\subseteq \bigcup_{k\in\mathbb{N}}^{\infty} \text{NTIME}[\mathcal{O}(n^k)] \\
&= \text{NPSPACE}
\end{aligned}$$

□

## 3.4 Results concerning the Chomsky Hierarchy

Now that we have seen most of the required theory, we can start to apply it to the main subject of this work, the Chomsky Hierarchy. For this section, we define the vocabulary on strings to be $\sigma = \langle Q_a, Q_b, \ldots, Q_z, \leq, 0, 1, \max\rangle$. We have a unary predicate for each character in $\Sigma$, a total ordering on the universe, and the constants $0, 1$ and max. For a universe $|\mathcal{A}| = \{0, 1, \ldots, n\}$, we require that $\max = n$. Also, for each $a \in \Sigma$ we have $Q_a(x)$ if and only if the $x^{th}$ character of the string is $a$. Again, only the results for context-sensitive languages are discussed in this section. Further results can be found in the context appendix under appendix B.2.

### 3.4.1 Context-Sensitive Languages

This is the language class that interests us most, as it has been studied less extensively than other language classes. Nevertheless, there are some known formalisms. One of them is the linear bounded nondeterministic Turing machine. This first equivalence is shown in appendix B.2.6. The second

formalism is given by the logic MSO(TC). This logic is defined as second-order logic restricted to quantification over unary relations supplemented with a transitive closure operator. The transitive closure operator takes the transitive closure over some graph and is defined as follows:

**Definition 3.7** (Transitive closure)**.** Let $\phi\begin{pmatrix}\overset{k}{a},\overset{k}{b}\end{pmatrix}$ be a formula with $2k$ free variables. We can see this formula as a directed edge relation over the graph with vertices $\overset{k}{c} \in |\mathcal{A}|^k$. Then, the transitive closure $\left(TC_{\substack{\overset{k}{a},\overset{k}{b}}}\phi\begin{pmatrix}\overset{k}{a},\overset{k}{b}\end{pmatrix}\right)\begin{pmatrix}\overset{k}{u},\overset{k}{v}\end{pmatrix}$ is true if and only if there is a path in the graph generated by $\phi$ from $\overset{k}{u}$ to $\overset{k}{v}$.

**Theorem 3.2.** *A language $L$ is context-sensitive if and only if it can be described by a formula in $MSO(TC)$.*

*Proof.* By theorem B.7, we can also show the equivalence of MSO(TC) and NSPACE[$\mathcal{O}(n)$] Turing machines.

First, we show that any formula in MSO(TC) can be evaluated by a NSPACE[$\mathcal{O}(n)$] Turing machine. The first step is to notice that any relation of MSO can be represented on the tape in $\mathcal{O}(n)$ space. For any sub-formula of the form $\left(TC_{\substack{\overset{k}{a},\overset{k}{b}}}\phi\begin{pmatrix}\overset{k}{a},\overset{k}{b}\end{pmatrix}\right)\begin{pmatrix}\overset{k}{u},\overset{k}{v}\end{pmatrix}$, we write down $\overset{k}{u}$ and guess the next vertex on the tape. Then, we can check if the transition is valid by evaluating $\phi$. If it is, we replace $\overset{k}{u}$ with the new vertex and repeat until we reach $\overset{k}{v}$. Because Immerman showed that NSPACE is closed under complementation in [Imm88], we know that we can also compute any sub-formula of the form $\neg\left(TC_{\substack{\overset{k}{a},\overset{k}{b}}}\phi\begin{pmatrix}\overset{k}{a},\overset{k}{b}\end{pmatrix}\right)\begin{pmatrix}\overset{k}{u},\overset{k}{v}\end{pmatrix}$. The other parts of the formula can be evaluated easily in linear space, as we just need to write down relations when quantifying and remember in which part of the constant-size formula we are.

For the other direction, consider a NSPACE[$\mathcal{O}(n)$] Turing machine $N$. We define the string vocabulary for our logic to be over the alphabet $\Sigma$ which contains the symbol $b_{a_i,q_j}$ for all pairs of tape symbols and Turing machine states.

Consider the tuple $\overline{X} = \langle Q_{b_1}, \ldots, Q_{b_r}\rangle$ containing a unary relation for each symbol in $\Sigma$. This tuple completely represents an instantaneous configuration of the tape and the machine. We can now write a formula $\varphi(\overline{X},\overline{Y})$ which means that a transition from state $\overline{X}$ to $\overline{Y}$ is possible in $N$. This can be done by a big disjunction over all rules of $N$. After one step of a Turing machine, a character $b$ at position $x$ is only determined by the actual character at position $x$ and the two characters left and right of it[3]. This means that we can write all transitions of $N$ in the form $\langle b_k, b_l, b_m\rangle \to \langle b_i, b_j, b_w\rangle$. If $P$ is the set of all transitions, we have

$$\varphi(\overline{X},\overline{Y}) \equiv \exists i\left(\forall j\left(|i-j|>1 \to \bigwedge_{b_i}\left(Y_{Q_{b_i}}(j) \leftrightarrow X_{Q_{b_i}}(j)\right)\right)\wedge \right.$$
$$\left.\bigvee_{\langle b_k,b_l,b_m\rangle\to\langle b_u,b_v,b_w\rangle\in P}\left(X_{Q_{b_k}}(i-1)\wedge X_{Q_{b_l}}(i)\wedge X_{Q_{b_m}}(i+1)\wedge Y_{Q_{b_u}}(i-1)\wedge Y_{Q_{b_v}}(i)\wedge Y_{Q_{b_w}}(i+1)\right)\right)$$

[3]we include markers at the two ends of the tape, which makes this well-defined

The first line asserts that apart from the indices where the head is and thus where we change something, the tape stays unchanged. The second line tells us that there exists a rule in $P$ such that the characters in $\overline{Y}$ at positions $i-1, i, i+1$ follow from the previous characters in $\overline{X}$.

Now, we can take the transitive closure over $\varphi$, starting with the start configuration and ending at an accepting position. Without loss of generality, we can assume that $N$ clears its tape after accepting, so the end position is unique. Thus, our formula holds if and only if there is an accepting path in $N$ on the input word $w$.

This concludes the equivalence of MSO(TC) and context-sensitive languages. □

An interesting normal form for MSO(TC) can be derived from the proof.

**Corollary 3.2.1.** *Every formula in MSO(TC) can be written in the form*

$$\left(TC_{\overline{X},\overline{Y}}\varphi(\overline{X},\overline{Y})\right)\left(\overline{U},\overline{V}\right)$$

*Proof.* We have given an explicit way to convert any formula into a Turing machine and back in theorem 3.2. By construction this always gives us a formula of the required form. □

# 4. Personal Contribution

In this chapter, results of searches for new first-order logics and other related concepts are presented. I did not find any fundamentally new result about the characterization of context-sensitive languages using first-order logic, but managed to prove that various approaches could not work. Further, I also lowered some upper bounds for simulating alternating Turing machines using iterative logic. Using a restricted form of iterative logic, I could lower the bounds for an iterative simulation of a nondeterministic Turing Machine.

## 4.1 Direct Transformation

Similar to Immerman in [Imm99], we will use extended variables to model second-order variables in first-order logic. This will then directly give us a characterization of NSPACE[$s(n)$] without requiring any new insights, as we can just use the same technique as for second-order logic.

**Definition 4.1** (Extended Variable)**.** The logic FO-VAR[$1, s(n)$] has two types of variables. One type are the domain variables, which we will denote by lowercase characters, ranging from 0 to $n-1$. The other type are extended variables, which we will denote by uppercase characters, ranging from 0 to $2^{s(n)\log(n)} - 1$, and thus having $s(n)\log(n)$ bits. The extended variables are not allowed to appear as input to any relation apart from BIT. Thus, we can only query if a specific bit in the binary representation of an extended variable is on. For extended variables with more than $n$ bits, we can extend BIT to accept a tuple of domain variables encoding the position we want to query. This makes extended variables with a polynomial number of bits possible.

The extended variables in FO-VAR[$1, n^k$] have exactly the same capabilities as second-order variables of arity $k$ in second-order logic. This is true as for every second-order variable $X$ we can have an extended variable $X'$ such that bit $\overline{x}$ of $X'$ is set if and only if $X(\overline{x})$ is true and vice versa. By generalizing the proof of theorem 3.2, we have NSPACE[$n^k$] = SO(TC, arity $k$). We will now prove that SO(TC, arity $k$) = FO-VAR[$1, n^k/\log(n)$](TC), thus also capturing NSPACE[$n^k$] with an extension of first-order logic.

*Proof.* We show by induction on the structures of the formulas that each formula in SO(TC, arity $k$) has a very similar equivalent in FO-VAR[$1, n^k/\log(n)$(TC) and vice versa.

Any atomic formula of SO(TC, arity $k$) which does not include any second-order variable is trivially writable in FO-VAR[$1, n^k/\log(n)$](TC). The same holds for formulas in FO-VAR[$1, n^k/\log(n)$](TC) without extended variables. As remarked before, an atomic formula with a second-order variable $Y(\overline{x})$ can be represented as querying the bit $\overline{x}$ of an extended variable $Y'$. We can do the same transformation back to a second-order variable for an atomic formula of the form BIT$(Y', \overline{x})$.

Using this, we then can induct on the structure of the formulas. Taking the conjunction, disjunction, or negation of two equivalent formulas makes them stay equivalent. When we quantify over a second-order or extended variable, we can directly exchange this with quantifying over the other type of variable. By induction, the sub-formulas without the quantification are equivalent, and binding the new second-order or extended variable makes the new formula equivalent. Taking a transitive closure can also be done by replacing all variables of the current type with variables of the other type. Again using induction on the sub-formulas, we get that the new formulas are equivalent.

By induction, we then have that every formula in either logic has an equivalent formula in the other logic. □

Using this equivalence and the proof in appendix B.2.6, we get that FO-VAR$[1, n/\log(n)]$(TC) describes exactly the context-sensitive languages. This gives us our first characterization of context-sensitive languages in first-order logic.

## 4.2 Analogues to Proof for DSPACE

For DSPACE$[s(n)]$, there are multiple other logical characterization which do not use the transitive closure operator.

We first need to define the logic FO$[t(n)]$, which formalizes iterative definitions, and then VAR$[k]$, which restricts the number of variables but allows for unbounded first-order iterations.

**Definition 4.2** (FO$[t(n)]$)**.** Let $Q_1, \ldots, Q_n$ be a series of quantifiers, $s_1, \ldots, s_n$ a series of variables and $M_1 \ldots, M_n$ a series of quantifier-free formulas. Then, we can form a quantifier block

$$QB = (Q_1 s_1.M_n) \ldots (Q_n s_n.M_n)$$

. For a universal quantifier, $(\forall s.M)\varphi \equiv \forall s(M \rightarrow \varphi)$. For an existential quantifier, $(\exists s.M)\varphi \equiv \exists s(M \wedge \varphi)$. Both of these effectively mean that we restrict the quantifiers to range only over the elements that satisfy $M$. Then, a formula of FO$[t(n)]$ is of the form

$$\left([QB]^{t(n)} M_0\right)(\overline{c}/\overline{s})$$

where $M_0$ is a quantifier-free formula, $\overline{c} = c_1, \ldots, c_n$ is a tuple of constants and $\overline{s} = s_1, \ldots, s_n$ are the variables occurring in the quantifier block. The notation $(\overline{c}/\overline{s})$ means that in the beginning, we set $s_1 \coloneqq c_n, \ldots, s_n \coloneqq c_n.[QB]^{t(n)}$ means $QB$ literally repeated $t(n)$ times. The truth values of these formulas for a specific structure are defined by evaluating the formula obtained by iterating the quantifier block $t(|\mathcal{A}|)$ times.

This gives us a formalism for iterative procedures.

If we restrict the number of variables such that all $s_i$ need to be in $\{x_1, \ldots, x_k\}$, limiting the number of distinct variables to $k$, but allow some additional boolean variables[1], we get FO-VAR$[t(n), k]$. In

[1] variables which have only two possible values

particular, we can reuse the same variable multiple times in the same quantifier block. Additionally, we define

$$\mathrm{VAR}[k] = \bigcup_{c=1}^{\infty} \text{FO-VAR}[2^{cn^k}, k]$$

This is the same as saying that we have unbounded iterations, as after at most $2^{cn^k}$ the truth values of the formula will loop or stay the same. The $c$ depends only on the number of boolean variables in the formula.

For DSPACE, we have

$$\mathrm{DSPACE}[n^k] = \mathrm{VAR}[k+1]$$

A proof of this can be found in [Imm99]. In the main part of the proof, a construction is made to simulate a DSPACE$[n^k]$ Turing machine using VAR$[k+1]$. There, the relation $C_t(\overline{x}, \overline{b})$ is inductively defined to mean that at time $t$, the character on the tape at position $\overline{x}$ is the one encoded by $\overline{b}$, where $\overline{b}$ is a tuple of boolean variables. The characters are defined in the same way as in theorem 3.2 to include the machine state as well as the tape symbol. Because the Turing machine we are contemplating is deterministic, this character is uniquely determined. Further, it only depends on the characters at positions $\overline{x}-1, \overline{x}$ and $\overline{x}+1$ at time $t-1$. Because we can assume without loss of generality that the Turing machine returns to the starting tape cell after accepting and clears its tape, finding out the value of $C_{2^{cn^k}}(\overline{0}, \overline{b})$ is sufficient to determine if the machine accepts.

If we want to extend this to nondeterministic Turing machines using $C_t(\overline{x}, \overline{b})$ to mean that the character at position $\overline{x}$ *can* be $\overline{b}$ at time $t$, we run into problems. As we do not have the guarantee that the computation is deterministic, we can not say any more that we depend only on the possible characters on the tape at time $t-1$. Doing this would mean that impossible states could be reached. This would happen because we do not know which combinations are possible, as we can not choose the characters of each tape cell independently. One way to fix this is by remembering the nondeterministic choices we made in the previous steps. This would make the values deterministic again, fixing the issue. In the worst case, this would take $\mathcal{O}(2^{cn^k})$ additional bits. But we already know by Savitch's theorem (section 3.3.1) and the equivalence for DSPACE that we only need $\log(n) \cdot (k+2)$ bits to represent a NSPACE$[n^k]$ computation.

Let VAR$[r(n)]$ be the generalization of VAR$[k]$ which means that a total of $r(n) + \mathcal{O}(1)$ bits of variables are allowed. Any proof which would show that NSPACE$[n^k]$ can be expressed in VAR$[r(n)]$ with $\log(n) \cdot (k+1) \leq r(n) < \log(n) \cdot (k+2)$ would be an improvement on Savitch's theorem. That is because the proof for DSPACE$[n^k]$ can easily be extended to more general polynomial functions, as Immerman showed in [IBB99]. Actually, this is a two-way relationship: if Savitch's theorem can be improved, we can describe NSPACE with less than $k+2$ variables and unbounded iterations.

## 4.3 Mixing iterations and transitive closure

One question that came up during the process was:Can we find a logic that is more restricted than

$$\text{FO-VAR}[s(n), s(n)/\log(n)]$$

but still captures NSPACE[$s(n)$]? This led to the idea of mixing up iterative procedures with the transitive closure operator. We will now show that a success of this approach would mean that there exists a restriction using only iterative procedures that also contains NSPACE[$s(n)$]. As formulas mixing up multiple operators and procedures are more difficult to analyse, it makes this approach unpractical.

There are two ways of combining the transitive closure with iterated formulas. The first is surrounding a formula in FO-VAR[$t(n), r(n)/\log(n)$] with a TC operator using variables of size $r(n)$. For this to contain NSPACE[$s(n)$], for any formula of the form $(\mathrm{TC}_{a,b}\varphi)(c,d)$ we need a quantifier block, a formula $\phi$ and some constants $\overline{e}, f, g$ such that

$$\left(\mathrm{TC}_{a,b}\left([QB]^{t(n)}\phi(\overline{e}/\overline{s})\right)\right)(f,g) \equiv (\mathrm{TC}_{a,b}\varphi)(c,d)$$

We know that we can simulate any TC formula with variables of size $r(n)$ using $r(n)$ iterations of a quantifier block. This is done by the technique of halving paths used in Savitch's theorem. An exact formula achieving this is presented in section 4.4. Thus, we find an equivalent formula

$$\left(\mathrm{TC}_{a,b}\left([QB]^{t(n)}\phi(\overline{e}/\overline{s})\right)\right)(f,g) \equiv [QB_1]^{r(n)}[QB_2]^{t(n)}\phi(\overline{e}/\overline{s}, c/a, d/b)$$

. The two quantifier blocks can then be merged into one by additionally maintaining a counter variable of size $\log(r(n))$ such that the quantifier block acts like $[QB_1]$ during the first $r(n)$ round and then acts like $[QB_2]$.

The other case where we write the TC formula after the quantifier block can be treated similarly.

$$\left([QB]^{t(n)}\left(\mathrm{TC}_{a,b}\phi\right)(f,g)\right)(\overline{e}/\overline{s}) \equiv (\mathrm{TC}_{a,b}\varphi)(c,d)$$

If the condition $c \cdot r(n) > \log(t(n))$ is satisfied for some fixed $c$, we can do the same counting trick. As any $t(n) > 2^{cr(n)}$ can be seen as having an unbounded amount of iterations, we can assume that the condition holds without loss of generality.

In both cases, we have a formula in FO-VAR[$r(n) + t(n), r(n)/\log(n)$]. Now, there are multiple cases

$r(n) \leq t(n)$ Then, FO-VAR[$r(n) + t(n), r(n)/\log(n)$] = FO-VAR[$t(n), r(n)/\log(n)$]. This is only interesting in the case where $r(n) < s(n)$

$r(n) > t(n)$ Then, FO-VAR[$r(n) + t(n), r(n)/\log(n)$] = FO-VAR[$r(n), r(n)/\log(n)$]. This is again only interesting when $r(n) < s(n)$.

We thus get that any improvement on our simulation results happen only when we have less than $s(n)$ variable bits. Using Savitch's theorem once more, we also get that $r(n) > \sqrt{n}$. Another result is that Savitch's theorem can be improved if $t(n)r(n) < s(n)^2$ as

$$\text{FO-VAR}[t(n), r(n)/\log(n)] \subset \text{DSPACE}[t(n)r(n)]$$

The same method also tells us that $t(n)r(n) \geq s(n)$ because otherwise, we could simulate a

NSPACE[$s(n)$] machine deterministically using less than $s(n)$ space, which is impossible by the space hierarchy theorem described in appendix B.2.3.

## 4.4 Restricting universal quantification

One further approach that I attempted was restricting universal quantification in the FO-VAR formulas. I called the resulting logic FO$\exists$-VAR. In FO$\exists$-VAR, universal quantification is only allowed over boolean variables. We will denote boolean variables by $b_j$. Further, the requirement of the $M_i$ to be quantifier-free is dropped. This step was motivated by the fact that nondeterministic Turing machines in essence capture existential quantification.

We use a similar idea to the one used in Savitch's theorem, guessing the middle of a path and checking if both shorter paths are connected. A formula in FO$\exists$-VAR[$s(n), s(n)/\log(n)$] which is equivalent to a formula in FO-VAR[$1, s(n)/\log(n)$](TC) of the form $\left(TC_{\overline{X},\overline{Y}}\varphi\right)(\overline{C},\overline{D})$ is

$$\begin{aligned}
QB &\equiv (\forall b_1.M_1)(\exists\overline{Z})(\forall b_2)(\exists\overline{A},\overline{B}.M_2)(\exists\overline{X},\overline{Y}.M_3)\\
M_1 &\equiv \neg(\forall\overline{z}(\mathrm{BIT}(\overline{X},\overline{z}) \leftrightarrow \mathrm{BIT}(\overline{Y},\overline{z})) \vee \varphi(\overline{X},\overline{Y}))\\
M_2 &\equiv (b_2 \wedge \forall\overline{z}(\mathrm{BIT}(\overline{X},\overline{z}) \leftrightarrow \mathrm{BIT}(\overline{A},\overline{z})) \wedge \forall\overline{z}(\mathrm{BIT}(\overline{B},\overline{z}) \leftrightarrow \mathrm{BIT}(\overline{Z},\overline{z})))\vee\\
&\quad (\neg b_2 \wedge \forall\overline{z}(\mathrm{BIT}(\overline{Z},\overline{z}) \leftrightarrow \mathrm{BIT}(\overline{A},\overline{z})) \wedge \forall\overline{z}(\mathrm{BIT}(\overline{B},\overline{z}) \leftrightarrow \mathrm{BIT}(\overline{Y},\overline{z})))\\
M_3 &\equiv \forall\overline{z}(\mathrm{BIT}(\overline{X},\overline{z}) \leftrightarrow \mathrm{BIT}(\overline{A},\overline{z})) \wedge \forall\overline{z}(\mathrm{BIT}(\overline{B},\overline{z}) \leftrightarrow \mathrm{BIT}(\overline{Y},\overline{z}))
\end{aligned}$$

and finally

$$[QB]^{cs(n)}(false)(\overline{C}/\overline{X},\overline{D}/\overline{Y})$$

for some $c.M_1$ describes the breaking condition, being false whenever either $\overline{X}$ and $\overline{Y}$ are equal or connected directly. So in $QB$, the first quantification means that we break with the actual branch of the formula being true whenever we have a connection. When this happens, we quantify universally over the empty set, which is defined as true. If no connection exists yet, a middle configuration $\overline{Z}$ is guessed, and both sides are checked by universally choosing $b_2$. In $M_2$, we then check that $\overline{A}$ and $\overline{B}$ are the endpoints of the path determined by $b_2, \overline{X}, \overline{Z}$ and $\overline{Y}$. After this, we copy $\overline{A}$ to $\overline{X}$ and $\overline{B}$ to $\overline{Y}$ in $M_3$. When iterating this, we guess a path from $\overline{C}$ to $\overline{D}$, checking every connection between adjacent states.

Define FO$\forall,\exists$-VAR[$t(n), f(n), s(n)$] to contain all formulas with $t(n)$ iterations of a quantifier block, extended variables of $f(n)$ bits which can be universally quantified and extended variables of $s(n)\log(n)$ bits which can be used in existential quantification. Using this, we can make the more general statement that for any function $r(n) \leq 2^{cs(n)}$, we can define a formula in

$$\mathrm{FO}\forall,\exists\text{-VAR}[s(n)/\log(r(n)), \log(r(n)), s(n)r(n)/\log(n)]$$

which simulates a nondeterministic Turing machine using $s(n)$ space. These formulas are very similar to the one defined for FO$\exists$-VAR. The only difference lies in how we split the path. Instead of splitting paths into two parts, we split them into $r(n)$ parts. This is done by guessing $r(n) - 1$ middle

configurations at once in an existential extended variable and choosing which part we check using the universal extended variable. Thus, for a path of length $2^{cs(n)}$, we need at most

$$\log_{r(n)}\left(2^{cs(n)}\right) = \frac{\log\left(2^{cs(n)}\right)}{\log(r(n))} = cs(n)/\log(r(n))$$

iterations. As for simulating this formula with a Turing machine, the product of the number of iterations and the size of the extended variables are important, we do not gain any tighter results from this.

## 4.5 Alternating bounds

In the proof of Savitch's theorem, we use alternating Turing machines as an intermediate step for proving that NSPACE[$s(n)$] is a subset of DSPACE[$s(n)^2$]. We can also add another intermediate step using FO-VAR[$s(n), s(n)/\log(n)$] with the method shown in section 4.4. It is also quite easy to simulate any formula in FO-VAR[$t(n), s(n)/\log(n)$] with an alternating Turing machine in ATSR[$t(n)s(n), s(n), t(n)$], where in this new class ATSR, we specify *t*ime, *s*pace and *r*eversals. With reversals, we mean the number of times we switch from universal states to existential states and back. We do this by writing the current values of all variables on the tape during iterating and evaluating the quantifier-free FO formulas on the way, which can be done efficiently in terms of space.

We will now investigate two ways of simulating an ATSR[$t(n), s(n), r(n)$] Turing machine using FO-VAR.

The first one is a straightforward simulation of each step of the computation. For this, we encode the whole state of the Turing machines in a tuple of $s(n)$ bit extended variables and iteratively guess the next configuration. Also, we assume without loss of generality that a predicate $existential(\overline{X})$ exists and that there exists exactly one accepting state, denoted by $\overline{E}$.

$$\begin{aligned}
QB &\equiv (\forall b.M_1)(\exists\overline{A}.\varphi(\overline{X},\overline{A}))(\forall\overline{B}.M_2)(\exists\overline{X}.\forall\overline{z}(\mathrm{BIT}(\overline{X},\overline{z}) \leftrightarrow \mathrm{BIT}(\overline{B},\overline{z})))\\
M_1 &\equiv \neg\forall\overline{z}(\mathrm{BIT}(\overline{X},\overline{z}) \leftrightarrow \mathrm{BIT}(\overline{E},\overline{z}))\\
M_2 &\equiv (existential(\overline{X}) \rightarrow \forall\overline{z}(\mathrm{BIT}(\overline{A},\overline{z}) \leftrightarrow \mathrm{BIT}(\overline{B},\overline{z}))) \wedge \varphi(\overline{X},\overline{B})
\end{aligned}$$

with

$$\left([QB]^{t(n)}(false)\right)(\overline{S}/\overline{X})$$

This gives us a formula in FO-VAR[$t(n), s(n)/\log(n)$] which simulates a computation of an alternating Turing machine. To simplify it, some $M_i$ are not quantifier-free. These can be moved out of the $M_i$, but make the formula less readable. The sub-formulas work as follows: The formula $QB$ starts off with checking if we already reached the accepting configuration using $M_1$. Next, if we are not done yet, we first existentially guess the next configuration, restricting this quantification to only those configurations which are directly connected to the actual one. In $M_2$ we either retain this existentially guessed next state if we are in an existential state (thus effectively doing nothing) or ignore the existentially quantified state and just quantify universally over all states which are connected to the

current one. The last thing we do is copying $\overline{B}$ to $\overline{X}$, making it the new start of our path.

By combining the above techniques for switching between universal and existential quantification with the path-halving method from Savitch's theorem, we get an even stronger result.

From now on, we will refer to configurations as vertices to emphasize that we are searching for paths on a graph. On a high level, our plan is to do the following:

1. From the actual starting vertex, find all reachable vertices using only vertices which have the same type (existential or universal) as the starting vertex.
   (a) For existential states, we can use the normal technique of halving paths naively
   (b) For universal states, we need to quantify over all ending vertices and then consider two cases
      i. We claim that there is no path using only universal states going to this vertex. Then, we need to show that for each middle vertex, one of the two new paths still does not exist.
      ii. We claim that a path exists. In this case, we can try to find a path normally.
2. Repeat with all the reached vertices

Before presenting the formula, we can first think of how much iterations we need to do. Because of the halving trick, we use $\log(a)$ iterations for a path of length $a$. By summing over all path segments with only one quantification type, we can bound the total number of iterations. To get an upper bound on this, we can use Jensen's inequality ([Mar19], p.28). We then get

$$\begin{aligned}\frac{\sum_{j=0}^{r(n)} \log(a_j)}{r(n)} &\leq \log\left(\frac{\sum_{j=0}^{r(n)} a_j}{r(n)}\right) \\ &= \log\left(\frac{t(n)}{r(n)}\right)\end{aligned}$$

because of the condition that $\sum_{j=0}^{r(n)} a_j = t(n)$. Multiplying this with $r(n)$, we get that a path can require a maximum of $r(n)\log\left(\frac{t(n)}{r(n)}\right)$ iterations. Thus, the formula is in FO-VAR$[r(n)\log\left(t(n)/r(n)\right), s(n)/\log(n)]$

The formula is then as follows:

$$
\begin{aligned}
QB \equiv\ & (\forall b_0.M_1)(\exists b_0.N_1)(\exists b_1.M_2)(\exists Z_e.M_3)(\forall Z.N_2) \\
& (\exists b_{2e})(\forall b_2.N_3)(\exists A,B.M_4)(\forall S,E.M_5)(\exists A,B.M_6)(\exists I,X.M_7)(\exists b_0.N_4)(\exists b_{path}.N_5) \\
M_1 \equiv\ & b_{path} \rightarrow \neg(I = Y \wedge (S = E \vee \varphi(S,E))) \\
N_1 \equiv\ & \neg b_{path} \rightarrow \neg(S = E \vee \varphi(S,E)) \\
M_2 \equiv\ & \neg b_1 \leftrightarrow (S = E \vee \varphi(S,E)) \\
M_3 \equiv\ & existential(Z_e) \leftrightarrow existential(X) \\
N_2 \equiv\ & (b_{path} \rightarrow Z = Z_e) \wedge (existential(Z) \leftrightarrow existential(X)) \\
N_3 \equiv\ & \neg b_{path} \rightarrow b_2 = b_{2e} \\
M_4 \equiv\ & (b_1 \wedge ((\neg b_2 \wedge A = S \wedge B = Z) \vee (b_2 \wedge A = Z \wedge B = E))) \vee \\
& (\neg b_1 \wedge A = I \wedge (B = Y \vee (existential(B) \leftrightarrow \neg existential(I)))) \\
M_5 \equiv\ & S = A \wedge (((b_1 \vee existential(S)) \wedge E = B) \vee \\
& (\neg(b_1 \vee existential(S)) \wedge (E = Y \vee (existential(E) \leftrightarrow \neg existential(S))))) \\
M_6 \equiv\ & (\neg b_1 \wedge S = A \wedge E = B) \vee (b_1 \wedge X = A \wedge I = B) \\
M_7 \equiv\ & X = A \wedge I = B \\
N_4 \equiv\ & b_0 = b_{path} \\
N_5 \equiv\ & (b_1 \wedge b_{path} = b_0) \vee (\neg b_1 \wedge (existential(X) \rightarrow b_{path}))
\end{aligned}
$$

and $\left([QB]^{r(n)\log(t(n)/r(n))}(\neg b_{path})\right)(true/b_{path}, AS/S, AS/X, AS/I, AS/S, AS/E, AE/Y)$ being the final formula.

First, we will look at the meaning of the variables:

$Y$**:** the accepting vertex

$X$**:** the starting vertex in the actual component

$I$**:** the vertex for which we are checking that a path from $X$ exists

$S$**:** the start of the path segment we are checking right now

$E$**:** the end of the path segment we are checking right now

$Z, Z_e$**:** the guessed middle vertex of a path segment from $S$ to $E$

$A, B, b_0$**:** variables used for copying or as dummies

$b_{path}$**:** whether we are trying to show that there is a path or that there is no path from $S$ to $E$

$b_1$**:** remember whether $S$ and $E$ are connected

$b_2, b_{2e}$**:** which half of the $S \rightarrow Z \rightarrow E$ path should be checked

As this formula is quite intricate, it is proposed to read it four times, each time focusing on a different stage of the search.

1. First, we look at the formula in the easiest case: We are still trying to find a path from $X$ to $I$ using $S$ and $E$, which are not directly connected. In this case, we have that $b_{path}$ and $b_1$ are true. Then, $QB$ works like this: First, in $M_1$, we check that we are not done, which we are not as $S$ and $E$ are not connected. As $b_{path}$ is true, $N_1$ does not affect anything. In $M_2$, $b_1$ is set to true. Then, we guess a middle vertex between $S$ and $E$ which has the same type as $X$ in $M_3$. $N_2$ just copies $Z_e$ into $Z$. Next, we choose a side of the path. As $b_{path}$ is true, this is done universally, because $N_3$ ignores the choice that was made for $b_{2e}$. $M_4$ copies the relevant new starting and ending vertices to $A$ and $B$. Because $b_1$ is true, we then copy $A$ and $B$ to $S$ and $E$ in $M_5$. Together $M_6$ and $M_7$ just copy $X$ and $I$ to $A$ and $B$ and back again. The same happens with $b_{path}$ using $N_4$, $N_5$ and $b_0$. In the end, we get a new configuration with a guessed middle vertex replacing either the former ending or starting vertex. If no path exists, this will continue until the formula $\neg b_{path}$ marks this branch as false.

2. The second option we look at is when we claim that no path exists. In that case, we have $\neg b_{path}$. Because of this, $M_1$ does not have any effect. In $N_1$, we check that we did not find any connection, and if we did find one, this logical branch will be false because of the existential quantification over the empty set. If we get $N_1$ is true, we get on to $M_2$, which will establish $b_1$. The formula $N_2$ completely ignores $M_3$, which thus has no effect. $N_2$ itself then universally quantifies over all middle vertices $Z$ which could lie on a path between $S$ and $E$. The next two quantifiers have the effect of quantifying existentially which half of $S \to Z \to E$ is not connected. This half must exist, as otherwise a path from $S$ to $E$ exists, which we claimed to be false by setting $b_{path}$ to false. Because $b_1$ is again true, $M_4$ will copy the relevant part of the path to $A$ and $B$. These are then copied back to $S$ and $E$ in $M_5$. $M_6$, $M_7$, $N_4$ and $N_5$ do the same as in the first case, which was not changing $X$, $I$ and $b_{path}$. If effectively no path exists, we will continue doing this until we hit $\neg b_{path}$. At this point, this branch of the formula will be true, as effectively, no path from $X$ to $I$ exists by making these choices.

3. The third possibility we need to consider is that we are presently trying to find a path from $X$ to $I$ in a universal component and that we succeed in finding a connection between $S$ and $E$. In this case, we have $b_{path}$ and $\neg b_1$. In $M_1$, if $I = Y$, we are done as we reached the accepting vertex. Otherwise, we can again ignore $N_1$, and $M_2$ will set $b_1$ to false. We can ignore the quantification for $Z$ and $b_2$ as they will not be used in this iteration of the formula. In $M_4$, $A$ and $B$ are guessed such that $A$ is the new starting vertex, which was the ending vertex $I$ before, and $B$ is either the accepting vertex $E$ or a universal vertex. Then, $M_5$ copies $A$ to $S$, marking it as the starting vertex. Further, because we changed quantification type, we have that a valid $S$ must now be existential. Thus, we also copy $B$ too $E$ directly. After that, we copy $S$ to $X$ and $E$ to $I$ via $A$ and $B$ in $M_6$ and $M_7$. The last change happens in $N_5$, where we set $b_{path}$ to be true. Thus, the variables are set for a new block of existential states.

4. The last option is similar to the third one, with the difference that the actual state $X$ is existential. In this case, we have $b_{path}$ and $\neg b_1$. In $M_1$, if $I = Y$, we are done, as we reached the accepting vertex. Otherwise, we can again ignore $N_1$, and $M_2$ will set $b_1$ to false. We can

ignore the quantification for $Z$ and $b_2$ as they will not be used in this iteration of the formula. In $M_4$, $A$ and $B$ are guessed such that $A$ is the new starting vertex, which was the ending vertex $I$ before, and $B$, which will be ignored later, is either the accepting vertex $X$ or an existential vertex. Then, $M_5$ copies $A$ to $S$, marking it as the starting vertex. Further, because we changed quantification type, we have that a valid $S$ must now be universal. Thus, we quantify universally over all vertices which are either the accepting vertex or are existential. After that, we copy $S$ to $X$ and $E$ to $I$ via $A$ and $B$ in $M_6$ and $M_7$. The last change happens in $N_5$, where we quantify existentially if we claim that a path from $X$ to $I$ exists or if no such path does. Thus, the variables are set for a new block of universal states.

The complete formula sets the starting values of the variables. By setting $b_{path} = true$ and all of $X, S, E$ and $I$ to the staring configuration $AS$, we trigger either case three or four. This is what we want as after this, we will search the component including $AS$. The accepting vertex $Y$ is set to the desired ending $AE$.

These results allow for enclosing an ATSR class between two FO-VAR classes with only logarithmic overhead.

$$\text{FO-VAR}\,[a(n), s(n)/\log n] \subseteq \text{ATSR}[a(n)s(n), s(n), a(n)] \subseteq \text{FO-VAR}\,[a(n)\log(s(n)), s(n)/\log n]$$

# 5. Conclusion and Direction

## 5.1 Conclusion

In this work, we investigated multiple logics which have a relation to Savitch's theorem. This has led us to a better notion of how these logics are related to NSPACE$[s(n)]$ and thus also to linear bounded automata and context-sensitive languages.

In the beginning, we introduced the theory of formal languages and the Chomsky Hierarchy. Afterward, the main tool of Descriptive Complexity used in this work, Complexity theory, was presented. Savitch's theorem is the next proof included in this chapter. This theorem is then present throughout the work. Then, the equivalence between context-sensitive languages and monadic second-order logic is shown.

Personally, I investigated multiple ideas I got during the time I was reading the theory.

The first one was a direct transformation of the results of the equivalence between second-order transitive closure logic and nondeterministic Turing machines with bounded space to first-order logic. This approach does not contain any new insights, but showcases the close relationship between first- and second-order logic.

After this, I wanted to explicitly prove that some methods could not work. This branch of my work started with the investigation of the proof that DSPACE is equivalent to VAR$[k+1]$. There, I showed that naively applying this method gives wrong results, and that a natural extension by remembering decisions is suboptimal. A full proof was impossible to make, as it is unclear how to even define a generalization of this technique. The difficulty of this problem is illustrated by Savitch's theorem, which has not been improved for a long time and thus makes it seem unlikely that a simulation of NSPACE machines with subquadratic DSPACE machines is possible.

The next method I tried was mixing transitive closure logic with iterative logic. This turned out to be not very powerful, as any interesting result in one of the mixed logics would imply one in pure iterative logic. Nevertheless, some upper and lower bounds were shown using Savitch's theorem and the space hierarchy theorem.

In an attempt to make FO-VAR less powerful, I then considered restricting universal quantification. A new formula made this possible. As for the other restrictions, I was unable to find a way to simulate formulas in my restriction of FO-VAR using NSPACE Turing machines. The generalization of this idea by adding more bits to the extended variables and decreasing the iteration count gives a space-time tradeoff, but does not solve the inherent problem of simulating this formula by NSPACE machines.

The most complicated result I was able to get does not have any tight connection to context-sensitive languages. It concerns the simulation of alternating Turing machines using FO-VAR. There, a quite tight containment could be found which bounds some alternating Turing machine class from

above and below by a factor of only $\log(s(n))$. A combination of different techniques used in the previous results was used to get the logical formula. This result could lead onto a path to describe alternating space-time classes exactly using an equivalent logic.

### 5.1.1 Working process

Generally, this work was a great learning opportunity for me. I was able to see how scientific research is done and could immerse myself in a domain that interests me. I knew that I could not plan with any significant results, and indeed I worked on a lot of different ideas, but most of the time hit a dead end.

In the beginning of the working process, I spend a lot of time reading multiple books about Descriptive Complexity and logic. Then, I researched more papers on the topic and tried to understand lower and upper bounds that were described there. This took more time than expected, but I still was able to start investigating myself as planned. As this is my first scientific research work, I did not know exactly how to approach it, so I wrote to Dr. Královič from the ETH Zürich. His tips were helpful and gave me a starting point for the research. During the months that followed, I tried multiple ideas and read a lot of papers concerning things I wanted to prove or needed for own proofs. Quite soon, Ms. Sprunger and I discussed that starting to write the theoretical part of this work would help reduce the stress before handing in. I did this during the summer break, experiencing some difficulties in the simplification of proofs while still maintaining their correctness. Afterward, I continued doing research. Two months before handing in, the International Olympiad in Informatics took place. My time plan did not account for this very well, so I had to work quite a lot in the last two weeks to write down my personal contribution.

Overall, I think there are three main points which could be improved:

- First, I often feel that my proofs are not explained in a way that is clear and concise. This makes it difficult for the reader to follow the arguments and understand the underlying thought process.
- Another point are the methods of research. I found it difficult to find any resources on this matter for the highly abstract field of mathematics. Further, I had no mentor on this project which could help me along the right path.
- The time management was suboptimal, as I spend a lot of time on ideas that were clearly not working and thus did not have time to investigate everything I wanted. Also, I did not start writing down my own research until very late, which generated some stress in the end.

## 5.2 Directions

In the future, multiple topics could be investigated further to get an even deeper understanding on the inner workings of Savitch's theorem.

In all proofs concerning equivalences between transitive closure logics and NSPACE$[s(n)]$, we assumed that $s(n)$ is at most polynomial. This is still a quite profound limitation, and no characterization

for superpolynomial bounds is known to me. Finding a generalization could lead to new insights that could help in the polynomial case, and in the end to the case where $s(n) = n$, capturing exactly the context-sensitive languages.

In the context of Savitch's theorem, it is interesting to investigate the computation graphs of NSPACE and DSPACE machines. This has been discussed on the theoretical computer science stack exchange in [Bar10]. A motivation for this is that the number of nodes in both graphs are the same, only the number of edges vary.

Another direction that could be investigated is looking at normal forms. One that seems to have some potential is the Kuroda Normal Form described in [Kur64]. This approach could yield a semantic restriction similar to the one for context-free languages described in appendix B.2.5.

After showing that mixing TC and iterative procedures does not give any strong characterizations in section 4.3 and seeing that all improvements happen when the extended variables have less than $s(n)$ bits, I came up with an idea that took up a lot of time and yielded very little results. A very interesting result would have been to show that any iterative formula simulating a TC computation needs to have variables of at least the same size. This would have shown that the path of investigating any logic which is a subset of these logics can not work. To prove this, I read multiple papers concerning hierarchies over transitive closure operators, generalized quantifiers and alternating space-time classes. One of these is "A double arity hierarchy theorem for transitive closure logic" [GH96] which combines transitive closure operators and generalized quantifiers and shows strict hierarchies on unordered graphs. The main problem with the approaches of the papers I read is that they can not be generalized to include string structures, which are inherently ordered. This makes it impossible to connect them to Turing machines, which work on strings. Some steps in this direction were made in the chapter about proving lower bounds in [Imm99].

# Acknowledgements

I would like to express my gratitude to all the people who helped me on this journey. First of all, I want to thank my supervisor Aline Sprunger for her guidance on mathematical research and her valuable feedback. Further, I would like to thank Dr. Gabriele Röger of the university of Basel for teaching me the basics of the theory of computer science and sparking my interest in this domain. I am thankful to Dr. Richard Královič who gave me good tips for where to start my research. I am also deeply grateful to my family for their support during stressful moments. More thanks go to all the people who accepted to reread this work and who gave me useful feedback.

# Bibliography


[Bar10] Boaz Barak. *Answer to "Tight Lower bounds on Savitch's theorem"*. Nov. 2010. URL: https://cstheory.stackexchange.com/a/2634 (visited on 03/10/2024).

[BB23] Ivo Blöchinger and Luca Bosin. *Vorlage für Maturarbeit der Kantonsschule am Burggraben in St. Gallen*. Apr. 2023. URL: https://github.com/techlabksbg/matura-arbeit-vorlage-latex (visited on 24/06/2024).

[Cli21] Climate Impact Partners. *The carbon footprint of the internet*. Apr. 2021. URL: https://www.climateimpact.com/news-insights/insights/infographic-carbon-footprint-internet/ (visited on 30/07/2024).

[Ent20] Rising Entropy. *The Arithmetic Hierarchy and Computability*. Aug. 2020. URL: https://risingentropy.com/the-arithmetic-hierarchy-and-computability/ (visited on 24/07/2024).

[GH96] Martin Grohe and Lauri Hella. 'A double arity hierarchy theorem for transitive closure logic'. en. In: *Archive for Mathematical Logic* 35.3 (May 1996), pp. 157–171. ISSN: 1432-0665. DOI: 10.1007/BF01268616.

[Gro96] Martin Grohe. 'Arity hierarchies'. In: *Annals of Pure and Applied Logic* 82.2 (1996), pp. 103–163. ISSN: 0168-0072. DOI: https://doi.org/10.1016/0168-0072(95)00072-0. URL: https://www.sciencedirect.com/science/article/pii/0168007295000720.

[Hal18] Sylvain Hallé. *TeXtidote*. Github. Version 0.9. Used for grammatical and orthographical correction of the whole work. June 2018. URL: https://github.com/sylvainhalle/textidote (visited on 04/10/2024).

[HR22] Malte Helmert and Gabriele Röger. *Lecture: Discrete Mathematics in Computer Science*. University of Basel. 2022. URL: https://dmi.unibas.ch/en/studies/computer-science/courses-in-fall-semester-2022/lecture-discrete-mathematics-in-computer-science/ (visited on 26/06/2024).

[Hro89] Juraj Hromkovič. 'Tradeoffs for language recognition on alternating machines'. In: *Theoretical Computer Science* 63.2 (Feb. 1989), pp. 203–221. ISSN: 0304-3975. DOI: 10.1016/0304-3975(89)90078-9.

[IBB99] Neil Immerman, Jonathan Buss and David Barrington. 'Number of Variables Is Equivalent To Space'. In: *Journal of Symbolic Logic* 66 (May 1999). DOI: 10.2307/2695103.

[Imm87] Neil Immerman. 'Languages that Capture Complexity Classes'. en. In: *SIAM Journal on Computing* 16.4 (Aug. 1987), pp. 760–778. DOI: 10.1137/0216051.

[Imm88] Neil Immerman. 'Nondeterministic space is closed under complementation'. In: *SIAM J. Comput.* 17.5 (Oct. 1988), pp. 935–938. ISSN: 0097-5397. DOI: 10.1137/0217058.

[Imm99] Neil Immerman. *Descriptive Complexity*. Springer New York, 1999. ISBN: 9781461205395. DOI: 10.1007/978-1-4612-0539-5.

[Kur64] S. Y. Kuroda. 'Classes of languages and linear-bounded automata'. In: *Information and Control* 7.2 (June 1964), pp. 207–223. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(64)90120-2.

[Lan03] LanguageTooler GmbH. *Free Grammar Checker*. Used for grammatical and orthographical correction of the whole work. 2003. URL: https://languagetool.org/ (visited on 06/10/2024).

[Lan63] Peter S. Landweber. 'Three theorems on phrase structure grammars of type 1'. In: *Information and Control* 6.2 (June 1963), pp. 131–136. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(63)90169-4.

[LST95] Clemens Lautemann, Thomas Schwentick and Denis Thérien. 'Logics for context-free languages'. en. In: *Computer Science Logic*. Ed. by Leszek Pacholski and Jerzy Tiuryn. Berlin, Heidelberg: Springer, 1995, pp. 205–216. ISBN: 9783540494041. DOI: 10.1007/BFb0022257.

[Mar19] Arnaud Maret. *Ungleichungen 1*. Swiss Mathematics Olympiad. Jan. 2019. URL: https://mathematical.olympiad.ch/fileadmin/user_upload/Archiv/Intranet/Olympiads/Mathematics/deploy/scripts/algebra/inequalities-1/script/inequalities-1_script_de.pdf (visited on 02/10/2024).

[Mat96] Jurij V. Matijasevič. *Hilbert's tenth problem*. 3. print. Foundations of computing. Aus dem Russ. übers. Cambridge, Mass. [u.a.]: MIT Press, 1996. 264 pp. ISBN: 0262132958.

[MW67] J. Mezei and J. B. Wright. 'Algebraic automata and context-free sets'. In: *Information and Control* 11.1 (July 1967), pp. 3–29. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(67)90353-1.

[Par02] Alan P. Parkes. *Introduction to languages, machines and logic: computable languages, abstract machines and formal logic*. eng. London: Springer, 2002. ISBN: 9781852334642.

[Pet22] Alberto Pettorossi. 'Formal Grammars and Languages'. In: *Automata Theory and Formal Languages*. Springer International Publishing, 2022, pp. 1–24. ISBN: 9783031119651. DOI: 10.1007/978-3-031-11965-1_1.

[Rob97] RoboCup Federation. *RoboCup Federation official website*. en-US. 1997. URL: https://www.robocup.org/ (visited on 29/07/2024).

[Rög23] Gabriele Röger. *Lecture: Theory of Computer Science*. Universität Basel. 2023. URL: https://dmi.unibas.ch/de/studium/computer-science-informatik/lehrangebot-fs23/main-lecture-theory-of-computer-science-1/ (visited on 26/06/2024).

[Str94] Howard Straubing. *Finite automata, formal logic, and circuit complexity*. eng. Progress in theoretical computer science. Boston: Birkhäuser, 1994. ISBN: 9780817637194.

[TW68] J. W. Thatcher and J. B. Wright. 'Generalized finite automata theory with an application to a decision problem of second-order logic'. en. In: *Mathematical systems theory* 2.1 (Mar. 1968), pp. 57–81. ISSN: 1433-0490. DOI: 10.1007/BF01691346.

[Wes19] Dag Westerståhl. 'Generalized Quantifiers'. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Winter 2019. Metaphysics Research Lab, Stanford University, 2019. URL: https://plato.stanford.edu/archives/win2019/entries/generalized-quantifiers/ (visited on 05/08/2024).

[Žák83] Stanislav Žák. 'A Turing machine time hierarchy'. In: *Theoretical Computer Science* 26.3 (Oct. 1983), pp. 327–333. ISSN: 0304-3975. DOI: 10.1016/0304-3975(83)90015-4.

# A. Mathematical Background

The definitions are taken from the lectures Discrete Mathematics in Computer Science [HR22] and Theory of Computer Science [Rög23], as well as from the book Descriptive Complexity [Imm99].

## A.1 Set Theory

**Set** An unordered collection of distinct elements, written with curly braces $\{\}$

**Tuple** An ordered collection of elements, written with pointed braces $\langle\rangle$

**Set operations** There are multiple ways to form new sets from already existing sets:

- **Union** denoted as $\cup$, an element is in $A \cup B$ if and only if it is in $A$ or $B$
- **Intersection** denoted as $\cap$, an element is in $A \cap B$ if and only if it is in $A$ and $B$
- **Cartesian product** denoted as $\times$, $A \times B$ is the set of 2-tuples $\langle a, b\rangle$ with an element $a$ of $A$ and an element $b$ of $B$
- **Cartesian power** $A^k$ denotes the Cartesian product of $A$ with itself repeated $k$ times
- **Power set** denoted as $\mathcal{P}(A)$, contains all subsets of $A$

## A.2 First-Order Logic

We abbreviate first-order logic as FO.

**Variable** A variable is an element that can have a value from a set

**Universe** The set over which variables and constants can range

**Relation** A relation of arity $k$, $R(x_1, \ldots, x_k)$ can be either true or false for any $k$-tuple of variables. In this work we always consider an equality relation $=$, an ordering relation $\leq$, and $\mathrm{BIT}(x, y)$, which means that the $y^{th}$ bit of $x$ is 1 in binary notation, to be present

**Vocabulary** A tuple $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s\rangle$ of relations $R_i$ with arity $a_i$ and constants $c_j$[1]

**Structure** A tuple $\mathcal{A} = \langle |\mathcal{A}|, R_1^{\mathcal{A}}, \ldots, R_r^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_s^{\mathcal{A}}\rangle$ where $|\mathcal{A}|$ is the universe, the constants are assigned a value from $|\mathcal{A}|$, and each relation is assigned a truth value for each $a_i$-tuple in $|\mathcal{A}|^{a_i}$. The set of all structures for a given universe is denoted as $\mathrm{STRUCT}[\tau]$

[1] We omit functions, which are included in most textbook definitions, as they can be simulated by a relation in our case

**First-Order Formula** A first-order formula is inductively defined as follows:

**Atoms** Any formula of the form $R(x_1, \ldots, x_k)$ for some relation of arity $k$ is called an atomic formula

**Conjunction** If $\varphi$ and $\psi$ are formulas, $(\varphi \wedge \psi)$ is a formula

**Disjunction** If $\varphi$ and $\psi$ are formulas, $(\varphi \vee \psi)$ is a formula

**Negation** If $\varphi$ is a formula, $\neg\varphi$ is a formula

**Existential quantification** If $\varphi$ is a formula, $\exists x\varphi$ is a formula

**Universal quantification** If $\varphi$ is a formula, $\forall x\varphi$ is a formula

**Free variables** A variable is called free if there is an occurrence of it in a formula which is not bound by a quantifier whose scope surrounds it

**Semantics** For any structure, we can assign a truth value to any formula over the corresponding vocabulary. If the formula contains free variables, these need to be assigned a value from the universe first. We say $\mathcal{A}$ satisfies $\phi$, denoted as $\mathcal{A} \models \phi$, if and only if $\phi$ is true under the interpretation of the constant and relations in $\mathcal{A}$. This truth value is inductively assigned to all formulas as follows:

**Atoms** For a formula $\phi$ of the form $R(x_1, \ldots, x_k)$, we have $\mathcal{A} \models \phi$ if and only if the interpretation of the relation $R^{\mathcal{A}}$ maps $\langle x_1, \ldots, x_k \rangle$ to true

**Conjunction** We have $\mathcal{A} \models (\varphi \wedge \psi)$ if and only if $\mathcal{A} \models \varphi$ and $\mathcal{A} \models \psi$

**Disjunction** We have $\mathcal{A} \models (\varphi \vee \psi)$ if and only if $\mathcal{A} \models \varphi$ or $\mathcal{A} \models \psi$

**Negation** We have $\mathcal{A} \models \neg\varphi$ if and only if $\mathcal{A} \not\models \varphi$

**Existential quantification** We have $\mathcal{A} \models \exists x\varphi$ if and only if there exists a $y \in |\mathcal{A}|$ such that $\mathcal{A} \models \varphi(y/x)$, where $\varphi(y/x)$ denotes $\varphi$ with any occurrence of $x$ replaced with the element $y$

**Universal quantification** We have $\mathcal{A} \models \forall x\varphi$ if and only if for all $y \in |\mathcal{A}|$ we have $\mathcal{A} \models \varphi(y/x)$

**First-Order Queries** A first-order query is a map from structures over one vocabulary $\sigma$ to structures over another vocabulary $\tau$. The mapping is done in such a way that first-order formulas define the universe, which is a subset of $|\mathcal{A}|^k$ for some $k$, the relation symbols, and all the constants. For a more formally thorough definition see [Imm99]

**Isomorphism** An isomorphism is a map $I : |\mathcal{A}| \to |\mathcal{B}|$ with $\mathcal{A}, \mathcal{B}$ over the same vocabulary which satisfies the following properties:

- $I$ is bijective
- for every available relation $R_i$ of arity $a_i$ and every $a_i$-tuple $\overline{e} = \langle e_1, \ldots, e_{a_i} \rangle$ in $|\mathcal{A}|^{a_i}$, we have

$$R_i^{\mathcal{A}}(e_1, \cdot, e_{a_i}) \Leftrightarrow R_i^{\mathcal{B}}(I(e_1), \cdot, I(e_{a_i}))$$

- for every constant symbol $c_j$, we have $I(c_j^{\mathcal{A}}) = c_j^{\mathcal{B}}$

If such an $I$ exists for two structures $\mathcal{A}$ and $\mathcal{B}$, we write $\mathcal{A} \cong \mathcal{B}$

## A.3 Second-Order Logic

In second-order logic, we extend the capabilities of first-order logic with the ability to quantify over relations. We thus also need to extend our definitions. We abbreviate second-order logic as SO.

**SO variables** A relation that is not given in the vocabulary and can be substituted with a specific interpretation

**SO formula** In addition to the inductive rules from the FO formulas, we can quantify over second-order formulas

**SO Existential quantification** If $\varphi$ is a formula, then $\exists V\varphi$ is a formula

**SO Universal quantification** If $\varphi$ is a formula, then $\forall V\varphi$ is a formula

**SO Semantics** Here we also need to extend the FO semantics

**SO Existential quantification** We have $\mathcal{A} \models \exists V\varphi$ if and only if there exists a relation $U$ over $|\mathcal{A}|$ such that $\mathcal{A} \models \varphi(U/V)$, where $\varphi(U/V)$ denotes $\varphi$ with any occurrence of $V$ replaced with $U$

**SO Universal quantification** We have $\mathcal{A} \models \forall V\varphi$ if and only if for all relations $U$ over $|\mathcal{A}|$ we have $\mathcal{A} \models \varphi(U/V)$

## A.4 Turing Machines

Turing machines are the most common model of computation.

**Informal definition** A Turing machine is an automaton with a finite number of states and an infinite tape. Using a read/write head, which can read one symbol on the tape, modify one symbol on the tape and move left and right, a Turing Machine can compute functions

**Formal definition** Formally, a Turing machine is a 7-tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\rangle$, where

$Q$ is the set of states

$\Sigma$ is the alphabet of the input word

$\Gamma$ is the set of symbols which can be written or read on the tape, which we call the tape alphabet

$\delta$ is the transition function, with $\delta : \Gamma \times Q \to \Gamma \times Q \times \{L, R\}$. This means that when a Turing machine is in state $n$ and reads $a$ on the tape, $\delta$ tells us to which state we should transition, which symbol we should write and which direction we should move the read/write head

$q_0$ the start state

$q_{accept}$ the accepting state

$q_{reject}$ the rejecting state

**Turing computation** At the beginning, the Turing machine is in the start state, the input word is written on a consecutive part of the tape and the read/write head is on the first character of the input word. In the following steps, the machine state changes according to the transition function. If at some point the machine enters the accepting or the rejecting state, the computation halts, and the Turing machine is said to have accepted / rejected the input. It can happen that the machine continues indefinitely or loops. In that case, we also say that it has rejected its input. In this work, we will ignore the tape content after the computation and focus on decision problems.

**Decidability** If a Turing machine halts on all inputs, we say that it decides a problem, as we can always be sure that the machine will accept or reject an input in finite time.

**Nondeterministic Turing machine** We can extend the transition function $\delta$ to allow multiple transitions from a given state. Formally, we then have $\delta : \Gamma \times Q \to \mathcal{P}(\Gamma \times Q \times \{L, R\})$. If at some point there are multiple transitions that are possible from the current state, we can take any of them. If there exists any computational path which leads to an accepting state, the nondeterministic Turing machine accepts. This is not analogous to how real sequential computers work, but allows interesting results, and is as powerful as a normal deterministic Turing machine.

**Space/Time-Constructible functions** A function $f(n)$ is time constructible if there exists a Turing machine which on input $1^n$ writes $f(n)$ in binary on its tape in time $f(n)$. Space-constructible functions are defined analogously.

**Church-Turing Thesis** The Church-Turing Thesis states that anything that can be done on a real-world computer can be done using a Turing machine.

# B. Mathematical Context and further proofs

## B.1 Formal languages

### B.1.1 Regular Languages

The regular languages have the most restricted type of grammars. Formally, any regular language can be described by a grammar with rules in $V \times (\Sigma \cup \Sigma V \cup \varepsilon)$. This means that we only have exactly one non-terminal on the left-hand side and the right-hand side is either a terminal, the empty word or a terminal symbol followed by a non-terminal symbol.

**Example B.1.** Consider the grammar $\langle \{S, O\}, \{a\}, R, S \rangle$ with

$$R = \begin{Bmatrix} S \to aO, & S \to \varepsilon, \\ O \to aS & \end{Bmatrix}$$

The generated language are exactly all words with even length.

The regular languages have been studied quite thoroughly and have multiple equivalent formalisms:

- The language is recognized by a Deterministic finite automaton, which process the input word one character at a time
- The language can be decided by a read-only Turing machine, that is a Turing machine that can not modify it's tape
- The language can be described by a regular expression

For a more in-depth analysis of regular languages and equivalent formalisms refer to appendix B.2.4 and [Str94].

### B.1.2 Context-Free Languages

The context-free languages extend the regular languages by allowing arbitrary right-hand sides for the rules of the defining grammar. Formally, that gives us rules in $V \times (\Sigma \cup V)^*$. Most valid arithmetic expressions, logical formulas and formally correct code in programming languages are context-free, as we can see the non-terminal symbols as types which are then converted to specific expressions of that type.

**Example B.2.** Consider the grammar $\langle\{\mathbf{Exp}, \mathbf{NumF}, \mathbf{Num}\}, \{0, 1, (, ), -, +\}, R, \mathbf{Exp}\rangle$ with

$$R = \begin{Bmatrix} \mathbf{Exp} \to \mathbf{NumF}, & \mathbf{Exp} \to (\mathbf{Exp} + \mathbf{Exp}), \\ \mathbf{Exp} \to (\mathbf{Exp} - \mathbf{Exp}), & \mathbf{Exp} \to (-\mathbf{Exp}), \\ \mathbf{Num} \to 0\mathbf{Num}, & \mathbf{Num} \to 1\mathbf{Num}, \\ \mathbf{Num} \to \varepsilon, & \mathbf{NumF} \to 0, \\ \mathbf{NumF} \to 1\mathbf{Num} & \end{Bmatrix}$$

This generates the language of all well-formed formulas using addition and subtraction over binary numbers. For clarity, **Exp** denotes an arbitrary expression, **NumF** any number without leading zeroes and **Num** any number (possibly empty or with leading zeroes).

Those languages have less known formalisms, the Push-Down Automaton (again see [Rög23]) being the most common. For a characterization of the context-free languages using logic, see appendix B.2.5.

### B.1.3 Recursive Languages

The recursive languages are the most general languages in the hierarchy, as they don't have any restrictions on the rules. It can be shown that this set of languages is equivalent to the languages recognizable by a Turing machine. By the Church-Turing thesis, this means that these are exactly the languages that can be computed by any of our computers and algorithms. Thus, we have a huge number of equivalent formalisms, including a RAM machine, while-programs and lambda calculus.

It is worth noting that there are languages which are not recursive. One of the most important example of these languages is the set of all (descriptions) of Turing machines which halt on every input, also known as the halting problem. For the characterization using logic, again refer to appendix B.2.7.

## B.2 Descriptive Complexity

### B.2.1 Ehrenfeucht-Fraïssé Games

Ehrenfeucht-Fraïssé games are combinatorial games which have a strong connection to first-order formulas and their extensions. Using these games, it is often possible to show inexpressibility results for certain problems in some logic $\mathcal{L}$.

As a motivation, we can look at what it means for a formula to hold on some structure. Assume the formula has the form $\forall x\varphi$. This can be seen as some opponent choosing some element $a \in |\mathcal{A}|$ and us then needing to show that $\varphi(x/a)$ holds. The case where the formula has the form $\exists x\psi$ can be treated similarly, but we can select the element ourselves.

**Definition B.1** (Ehrenfeucht-Fraïssé Game)**.** The *$k$-pebble Ehrenfeucht-Fraïssé Game* $\mathcal{G}_k$ is played by two players, the Spoiler and the Duplicator, on a pair of structures $\mathcal{A}$ and $\mathcal{B}$ using $k$ pairs of pebbles. In each move, the Spoiler places one of the remaining pebbles on an element of one of the two structures. Then, the Duplicator tries to match the move by placing the corresponding pebble on an element of the other structure. For a pebbled element $x$ in $\mathcal{A}$, we denote by $p(x)$ the corresponding

pebbled element in $\mathcal{B}$. We say that the Duplicator wins the $k$-pebble Ehrenfeucht-Fraïssé Game on $\mathcal{A}, \mathcal{B}$ if after the $k$ rounds, the map $i : |\mathcal{A}| \to |\mathcal{B}|$ defined as

$$i(x) = \begin{cases} c_j^{\mathcal{B}} & x = c_j^{\mathcal{A}} \\ p(x) & x \text{ is pebbled} \\ undefined & \text{otherwise} \end{cases}$$

is a partial isomorphism. A partial isomorphism is an isomorphism over some subset of the universe with all relations restricted to that subset.

In the Ehrenfeucht-Fraïssé Game, the Spoiler wants to show that $\mathcal{A}$ and $\mathcal{B}$ are different, whereas the Duplicator wants to show that they are equivalent.

As this is a zero-sum game with full information, one of the two players must have a winning strategy. It can be proven that the Duplicator has a winning strategy for the $k$-pebble Ehrenfeucht-Fraïssé on $\mathcal{A}$ and $\mathcal{B}$ if and only if $\mathcal{A}$ and $\mathcal{B}$ agree on all formulas with at most $k$ nested quantifiers. We can use these facts to prove the inexpressibility of some problems in first-order logic. Assume we have a decision problem $\mathcal{C}$. If we can exhibit two structures $\mathcal{A}_k \in \mathcal{C}$ and $\mathcal{B}_k \notin \mathcal{C}$ for each $k$ and a winning strategy for the Duplicator on these two structures, we show that no first-order formula defining $\mathcal{C}$ can exist. This methodology can be extended to other logics by adding new moves or restrictions to the game.

### B.2.2 Reduction and Completeness

A reduction can informally be seen as a method of using a problem we already solved to solve a new problem by converting this new problem into an instance of the old problem. These reductions can be very useful to define complete problems for complexity classes, which in turn enables us to prove theorems for all problems of a specific complexity class.

**Definition B.2** (first-order reduction)**.** Let $\mathcal{C}$ be a complexity class and $A$ and $B$ be two problems over vocabularies $\sigma$ and $\tau$. Now suppose that there is some first-order query $I : \text{STRUC}[\sigma] \to \text{STRUC}[\tau]$ for which we have the following property:

$$\mathcal{A} \in A \Leftrightarrow I(\mathcal{A}) \in B$$

Then $I$ is a first-order reduction from $A$ to $B$, denoted as $A \leq_{fo} B$.

First-order reductions can then be used to show that some problem is also a member in some complexity class, as in most complexity classes, we can compute the first-order query, and then we are left with a problem that we already know is in the required class. The converse can also be shown: for some problem $B$ which is not in some complexity class $\mathcal{C}$, if we have $B \leq_{fo} A$, then $A$ is also not in $\mathcal{C}$, as otherwise $B$ would also be in $\mathcal{C}$, which is a contradiction.

Using the reductions, we can define completeness.

**Definition B.3** (Completeness via first-order reductions for Complexity Class $\mathcal{C}$)**.** We say some problem $A$ is complete for $\mathcal{C}$ via $\leq_{fo}$ if and only if

- $A \in \mathcal{C}$
- for all $B \in \mathcal{C}$, we have $B \leq_{fo} A$

Informally, a complete problem captures the essence of the complexity class. Further, they have an application in some proofs of equivalences between complexity classes $\mathcal{C}$ and logics $\mathcal{L}$. These proofs follow the following steps as in [Imm99]:

1. Show that $\mathcal{L} \subseteq \mathcal{C}$ by providing a way to convert any formula $\varphi \in \mathcal{L}$ into an algorithm in $\mathcal{C}$.
2. Find a complete problem $T$ for $\mathcal{C}$ via first-order reductions.
3. Show that $\mathcal{L}$ is closed under first-order reductions, that is that any formula can be extended by first-order quantifiers and boolean connectives and stay in $\mathcal{L}$.
4. Find a formula for $T$ in $\mathcal{L}$, which shows $T \in \mathcal{L}$.

The above steps work, as for any problem $B$ in $\mathcal{C}$, there is a first-order reduction $I$ to $T$, and both $\mathcal{L}$ and $\mathcal{C}$ are complete via these reductions, so we also have $B \in \mathcal{L} = \mathcal{C}$.

### B.2.3 Space Hierarchy Theorem

The space hierarchy theorem states that for both nondeterministic and deterministic space, we have problems that can be solved using $t(n)$ space, but not with any tighter space bound. Formally, we have

$$\text{DSPACE}[o(t)] \subsetneq \text{DSPACE}[\mathcal{O}(t)]$$

where $o(t)$ is the set of functions $f$ such that $f \in \mathcal{O}(t)$ but $t \notin \mathcal{O}(f)$, that is all functions that grow more slowly than $t$. This holds for all space constructible $t \geq \log n$. The same holds for NSPACE.

We will present a proof for deterministic space.

*Proof.* The proof uses a diagonalization argument by presenting some machine $D$ that takes a Turing machine $M$ and an input size in unary as input and does the opposite of $M$ if it halts. We want to show that for all $M$ which run in space $f(n) \in o(t(n))$, we have an input on which $D$ and $M$ do not agree. This would show that the language computed by $D$ is not in DSPACE$[o(t)]$, and thus the strict containment.

On input $\langle M, 1^k \rangle$ our machine $D$ marks of $t(|\langle M, 1^k \rangle|)$ tape cells, which are the cells that are allowed for the computation. Further we also maintain a counter with size $|M| \cdot 2^{t(|\langle M,1^k\rangle|)}$, which is the maximum amount of different configurations a Turing machine can pass before looping on a binary tape of size $t(|\langle M, 1^k \rangle|)$. Then, we simulate $M$ on input $\langle M, 1^k \rangle$. If we transcend any bound, we reject. For all $M$ in DSPACE$[o(t)]$, there is a $k$ such that $f(n) \leq t(n)$ by definition. On this input, the simulation finishes, and we can invert the output.

This directly gives us an input for which $M$ and $D$ differ, and thus proves our claim. Furthermore, $D$ runs in DSPACE$[\mathcal{O}(t)]$ as by construction we assured that we do not run infinitely and that we stay within the space bound. □

### B.2.4 Regular Languages

Here, we will show that the regular languages are captured exactly by second-order logic where we restrict ourselves to quantify only over predicates of arity one and do not include $\leq$. Further, we also are not allowed to use $\leq$, but have access to equality $x = y$ and the successor relation $x = y + 1$. We call this class SOM[+1].

First we need to present a formal definition of deterministic finite automata.

**Definition B.4** (DFA)**.** A deterministic finite automaton is a 5-tuple $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ where

- $Q$ is the set of states
- $\Sigma$ is the alphabet
- $\delta$ is the transition function mapping a state and a symbol to the next state, so formally $\delta : Q \times \Sigma \to Q$.
- $q_0$ the start state
- $F$ a subset of $Q$ which are the accepting states.

We say that a DFA $D$ accepts a word $w \in \Sigma^*$ if when starting at the start state, if we go through $w$ and always transition to the next state according to the actual symbol in $w$ and the actual state, we end up in an accepting state.

In [Rög23] and [Str94] there is a proof of the following fact we will use in our proof for SOM[+1]:

**Theorem B.1.** *For any alphabet $\Sigma$, there is a DFA recognizing language $L \subseteq \Sigma^*$ if and only if it is regular.*

Now we can start to prove our main theorem for regular languages.

**Theorem B.2.** *For any alphabet $\Sigma$, a language $L \subseteq \Sigma^*$ is expressible in SOM[+1] if and only if it is regular.*

*Proof.* First we show that any regular language can be expressed in SOM[+1]. Let $L$ be regular, end $D_L$ be a DFA recognizing the language. We assume $L$ does not contain the empty word, otherwise we can recognize the language $L \setminus \{\varepsilon\}$ and then add $\varphi \vee \forall x(x \neq x)$, which adds the empty string back.

Now let $D_L$ have $k$ states. We can existentially quantify unary relations $X_1, \ldots, X_k$ to have the meaning that $X_i(y)$ is true if and only if $D_L$ is in state $i$ after $y$ steps. Then, we need to make consistency checks. We present formulas for each of the consistency checks, and then can take the "and" of those to get our final formula $\exists X_1, \ldots, X_k(\varphi_1 \wedge \varphi_2 \wedge \varphi_3)$.

**The start state is $q_j$** We have

$$\varphi_1 \coloneqq \bigwedge_{i=1}^{k} (i = j \leftrightarrow X_j(0))$$

**We end in an accepting state** Let $T_i$ be the set of all characters which lead from $q_i$ to an accepting state. Then we have

$$\varphi_2 \coloneqq \bigwedge_{i=1}^{k} \left( X_i(\max) \to \bigvee_{a \in T_i} Q_a(\max) \right)$$

**We move according to the transition function** We have

$$\forall x \left( \forall y \left( y = x + 1 \to \left( \bigwedge_{i=1}^{k} \bigwedge_{a \in \Sigma} \Big( (X_i(x) \wedge Q_a(x)) \to X_{\delta(i,a)}(y) \Big) \right.\right.\right.$$
$$\left.\left.\left. \wedge \bigwedge_{i=1}^{k} \bigwedge_{a \in \Sigma} \left( (X_i(y) \wedge Q_a(x)) \to \bigvee_{r=1}^{k} (X_r(x) \wedge \delta(r, a) = j) \right) \right) \right) \right)$$

By induction, we can show that always exactly one $i$ satisfies $X_i(x)$ for any $x$. Thus, if the created formula is satisfied, we know that $D_L$ accepts the word, and thus we have described $L$ in SOM[+1].

For the other direction, we need to introduce two new concepts.

One of them is the nondeterministic finite automaton, which is analogous to the nondeterministic Turing machine as it can also have multiple transitions going from the same state. As with the nondeterministic Turing machine and the Turing machine, both the DFA and the NFA have the same expressive power.

The other concept is that of $(\mathcal{V}_1, \mathcal{V}_2)$-structures. These structures are generalizations of our former vocabulary $\sigma$ as they have characters in $A \times \mathcal{P}(\mathcal{V}_1) \times \mathcal{P}(\mathcal{V}_2)$. These structures are useful as we can make $\mathcal{V}_1$ to be the set of free first-order variables in a formula $\varphi$ and $\mathcal{V}_2$ be the set of free second-order variables in the formula. If at a position $i$ in our $(\mathcal{V}_1, \mathcal{V}_2)$-structures we have $x$ in the first-order component of its character, we see this as meaning that $x = i$. For the second-order variables in the third component, an $X$ at position $i$ means that $X(i)$ holds.

Now, we can prove by induction that all formulas in SOM[+1] with free variables in $\mathcal{V}_1$ and $\mathcal{V}_2$ are regular. Sentences, the formulas without free variables are the special case where $\mathcal{V}_1, \mathcal{V}_2 = \emptyset$.

First, we need to check that the $(\mathcal{V}_1, \mathcal{V}_2)$-structures are consistent, and no first-order variable $x$ appears more than once. This can be done by a NFA which has one state for each subset of variables, and extends its subset while going over the string. If a variable appears twice, we enter a state that always loops and rejects.

Then, we see that the atomic formulas can be checked, as $x = y$, $x = y + 1$ and $Q_a(x)$ are easy to check, and checking $X(x)$ is equivalent to looking if the occurrence of $x$ has $X$ in the third component. We always need to take the intersection with the NFA which checks if the structure is valid.

All boolean connectives are also valid, as regular languages are closed under complement, intersection and union as seen in [Rög23].

The most difficult case is a formula of the form $\exists x \varphi$ (as $\forall x \varphi \equiv \neg \exists x \neg \varphi$)). If $\exists x \varphi$ is over $(\mathcal{V}_1, \mathcal{V}_2)$-structures, then $\varphi$ is over $(\mathcal{V}_1 \cup \{x\}, \mathcal{V}_2)$-structures. By induction, we know that $\varphi$ defines a regular language and thus there is a NFA $N$ which recognizes it. For the new automaton, we duplicate our states, with the meanings "used $x$" and "not used $x$". If we are in a state where $x$ was used, we can not take any transition with $x$ in the second set. If we are in a state where $x$ was not used, we can take a transition with $x$ in the second set and go to the corresponding state with $x$ used or take a transition where $x$ is not used and go to the corresponding state where $x$ was not used.

The remaining case with second-order variables is treated analogously, without the restriction on the number of times the variable is used, so we do not need to duplicate our states.

By induction, we have thus showed the other direction, and we see that SOM[+1] and the regular languages are equivalent. □

### B.2.5 Context-Free Languages

For the context-free languages, we will show that they have an underlying structure that includes matchings.

A matching relation is a binary relation $M$ on the universe $\{0, \dots, n-1\}$ which has the following properties:

**increasing** If $M(i, j)$, then $i < j$

**uniqueness** Any $k$ in the universe appears at most once in the relation

**non-crossing** If we were to draw arcs for the matching, none of them would intersect. Formally, if $M(i, j)$ and $M(k, l)$, then either $j < k$ or $l < i$

Visually, we can think of these relations as nested ranges over the universe.

With matchings, we can define FO(∃Match) as the first-order logic extended with existential quantification over matching relations.

**Theorem B.3.** *For any alphabet $\Sigma$, a language $L \subseteq \Sigma^*$ is expressible in FO(∃Match) if and only if it is context-free.*

For this, we first introduce a normal form for context-free grammars. The proof that every context-free grammar can be converted to this form can be found in [LST95].

**Lemma B.4.** *Every context free language has a grammar which satisfies*

- *All rules are of one of the two forms*
  - $S \to \alpha$ *with* $\alpha \in \Sigma$
  - $X \to \alpha u \beta$ *with* $\alpha, \beta \in \Sigma$ *and* $u \in (\Sigma \cup V)^*$
- *For all production rules with a right-hand side that has at least one non-terminal we define its* pattern. *The pattern of rule $X \to v_0 X_1 v_2 X_2 \dots X_s v_s$ is defined to be $v_0 | v_1 | \dots | v_s$, where | is a new symbol not in $\Sigma$ We require that for any two rules with the same pattern, they have the same left-hand side and thus the source non-terminal can be uniquely identified by the pattern.*

For a specific arch $\langle i, j \rangle$ in a matching $M$ on some word structure, we can also determine a pattern. For this, we go through all indices from $i$ to $j$ and add the character at the actual position. If we are at a starting point of some arch $\langle k, l \rangle$, instead of adding the actual character, we add | and continue at $l + 1$.

Now, we can start with the proof of our theorem.

*Proof.* We want to find a formula such that for all arches $\langle i, j\rangle$ in $M$, the substring from $i$ to $j$ can be derived from the non-terminal for which we have a rule with the pattern of $\langle i, j\rangle$.

For this, we say that arch $\langle i, j\rangle$ *corresponds* to a production rule $p$ if they have the same pattern. For any string $u$, we have a formula $\phi_u(i, j)$ which means that the substring from $i$ to $j$ is $u$. This formula is easy to write as we only need to check each character one by one using the successor relation. We can express correspondence to $p \equiv X \to v_0 X_1 v_2 X_2 \dots X_s v_s$ (including rules with terminal right-hand side) by the following formula:

$$\begin{aligned}\mathcal{X}_p(x, y) \equiv \exists x_1, y_1, \dots, x_s, y_s \,((x < x_1 \wedge x_1 < y_1 \wedge y_1 < x_2 \wedge \dots \wedge y_s < y) \wedge \\ (\phi_{v_0}(x, x_1 - 1) \wedge \phi_{v_1}(y_1 + 1, x_2 - 1) \wedge \dots \wedge \phi_{v_s}(y_s + 1, y)) \wedge \\ (M(x_1, y_1) \wedge \dots \wedge M(x_s, y_s) \wedge M(x, y)) \wedge \\ \forall k, l \,(M(k, l) \to ((x \le k \wedge l \le y) \vee (x_1 \le k \wedge l \le y_1) \vee \dots \vee (x_s \le k \wedge l \le y_s))))\end{aligned}$$

The first line means that the $v_i$ are in the right order, the second that they do correspond, the third that there are arches between the $v_i$ and the last that there are no other arches.

Now, we want to be more general, and express the that the pattern of $\langle i, j\rangle$ corresponds to some production rule with left-hand side $X$. Let $\widetilde{\mathcal{X}}_X(x, y)$ be the disjunction of all $\mathcal{X}_p$ with left-hand side $X$. Because our normal form says that the pattern uniquely determines the left-hand side, for each arch which has arches underneath, we have a unique corresponding non-terminal.

Now, we want to have a formula which expresses not only correspondence for a production rule $p$, but also that the non-terminals are correct. For this, we supplement our formula $\mathcal{X}_p$ with a new line, expressing that the non-terminals correspond.

$$\begin{aligned}\widetilde{\mathcal{X}}_p(x, y) \equiv \exists x_1, y_1, \dots, x_s, y_s \,((x < x_1 \wedge x_1 < y_1 \wedge y_1 < x_2 \wedge \dots \wedge y_s < y) \wedge \\ (\phi_{v_0}(x, x_1 - 1) \wedge \phi_{v_1}(y_1 + 1, x_2 - 1) \wedge \dots \wedge \phi_{v_s}(y_s + 1, y)) \wedge \\ (M(x_1, y_1) \wedge \dots \wedge M(x_s, y_s) \wedge M(x, y)) \wedge \\ \forall k, l \,(M(k, l) \to ((x \le k \wedge l \le y) \vee (x_1 \le k \wedge l \le y_1) \vee \dots \vee (x_s \le k \wedge l \le y_s))) \\ \left(\widetilde{\mathcal{X}}_{X_1}(x_1, y_1) \wedge \dots \wedge \widetilde{\mathcal{X}}_{X_s}(x_s, y_s)\right)\Big)\end{aligned}$$

Finally, with $P$ being our set of rules, we have a formula that tells us a word can be derived from the start symbol:

$$\exists M \left( \widetilde{\mathcal{X}}_S(0, \max) \wedge \forall x, y \left( M(x, y) \to \bigvee_{p \in P} \widetilde{\mathcal{X}}_p(x, y) \right) \right)$$

Now, by construction, if and only if a word satisfies the formula, there is a derivation from $S$ for the word, as each arch can be seen as a derivation step, which we check is valid with our formula. We used our assumption from the normal form to show that the $\widetilde{\mathcal{X}}_p$ are only satisfied when the arches which are non-terminal do not belong to any other non-terminal symbol then the one in the rule. If two terminal rules coincide, we do not care as no further derivation is possible.

Now, we come to the other direction. This direction requires some new notation and lemmas. We

will use the notion of tree languages.

**Definition B.5** (Tree Language)**.** In a tree language, we have a rooted tree, which has an order on its vertices in leftmost depth-first way[1]. Each node has a label, and each label has an arity which corresponds to the out-degree of the node. A tree language is the set of all trees over a finite label set.

We define the *leaf alphabet* of a tree language to be the set of 0-ary labels. For any tree $T$ in a tree language, we define the *yield* of $T$ to be the leaf labels concatenated according to the order relation from left to right.

The vocabulary of a tree language $\mathcal{T}$ is $\tau = \langle \{0, \ldots, n-1\}, Q_a, Q_b, \ldots, Q_z, \leq, 0, 1, \max, C^2 \rangle$ In addition to the relations for each label, there is a child relation $C(i, j)$ which means "node $i$ is a child of node $j$".

Now, we present two lemmas for recognizable tree languages and their relation to context-free languages.

**Lemma B.5** ([MW67])**.** *A language $L \subseteq \Sigma^*$ is context free if and only if there is a recognizable tree language $T$ with leaf alphabet $\Sigma$ for which a word is in $L$ if and only if it is the yield of some tree in $T$.*

**Lemma B.6** ([TW68])**.** *A tree language $T$ is recognizable if and only if there is a monadic second-order sentence that recognizes it.*

We now present some relations that can be written in MSO on trees.

**Lf($i$)** Node $i$ is a leaf $\mathrm{Lf}(i) \equiv \forall x \neq C(x, i)$

**Lc($i, j$)** Node $i$ is the leftmost child of $j$ $\mathrm{Lc}(i, j) \equiv C(i, j) \wedge \forall x(C(x, j) \to i < x)$

**Rc($i, j$)** Node $i$ is the rightmost child of $j$ $\mathrm{Rc}(i, j) \equiv C(i, j) \wedge \forall x(C(x, j) \to x < i)$

**An($i, j$)** Node $i$ is an ancestor or $j$

$$\begin{aligned}\mathrm{An}(i, j) \equiv \exists U(U(i) \wedge U(j) \wedge \forall x(U(x) \to \\ ((x \neq i \leftrightarrow \exists y(C(x, y) \wedge U(y))) \wedge (x \neq j \leftrightarrow \exists y(C(y, x) \wedge U(y))))))\end{aligned}$$

**Pt($U, i, j$)** Node $i$ is an ancestor or $j$, $j$ is a leaf and $U$ contains all nodes in the path from $i$ to $j$.

$$\begin{aligned}\mathrm{Pt}(U, i, j) \equiv U(i) \wedge U(j) \wedge \mathrm{Lf}(j) \wedge \forall x(U(x) \to \\ ((x \neq i \leftrightarrow \exists y(C(x, y) \wedge U(y))) \wedge (x \neq j \leftrightarrow \exists y(C(y, x) \wedge U(y)))))\end{aligned}$$

Using these two lemmas, we can continue by presenting for any formula $\varphi$ in FO($\exists$Match) a formula $\phi$ in MSO over trees such that

$$w \models \varphi \Leftrightarrow \text{there exists a tree } T \text{ with yield } w \text{ such that } T \models \phi$$

[1]The order which we find by doing a depth-first search on the tree, entering the leftmost node first

For this, we present a class of trees which correspond to a word with a matching. For any word $w$ with matching $M$, we can construct a tree over $\Sigma \cup \{\oplus^2, \odot^2\}$. We do this using an intermediate step. First, we construct a tree with wrong arity for nodes of type $\oplus$ by assigning one node of type $\oplus$ to each arch, with edges to every direct arch underneath it and every character directly underneath it. If we have multiple trees, we add a new $\odot$ node on top with an edge to all roots of these trees. To fix the arity issue, we repeat the following procedure until there are no nodes with more than outdegree 2.

Take some node with outdegree greater than 2. Take the two leftmost children and add a $\odot$ node with an edge to both, and an edge from the new node to the former parent. This procedure will eventually terminate as we always decrease the outdegree of some node by one and add a valid node.

We can see that this procedure can also be done backward if and only if for any node of type $\oplus$, the leaf on the leftmost and rightmost path of a node are distinct and no $\oplus$ node occurs on the path between the two. We can express this property of a tree by the following formula.

$$\begin{aligned}\Upsilon \equiv \forall x(Q_\oplus(x) \to (\exists y, z, U_y, U_z(y \neq z \wedge \mathrm{Pt}(U_z, x, z) \wedge \mathrm{Pt}(U_y, x, y) \wedge \\ \forall r(U_y(r) \to ((r = y \vee \exists w(U_y(w) \wedge \mathrm{Rc}(w, r))) \wedge (r = y \vee r = x \vee Q_\odot(r)))) \wedge \\ \forall r(U_z(r) \to ((r = z \vee \exists w(U_z(w) \wedge \mathrm{Lc}(w, r))) \wedge (r = y \vee r = x \vee Q_\odot(r)))))))\end{aligned}$$

Here, in the first line we assert that for every node with label $\oplus$, we have two distinct leaves $y, z$ with paths $U_y$ and $U_z$. The second and third line assert the same for $x$ and $y$, that they are the rightmost / leftmost leaf and that no other $\oplus$ type node lies on the path from them to $x$.

Now, we want to convert our formula $\varphi$ over strings with a matching to a formula $\gamma$ over trees. Then we can assert that the tree represents a string with matching and the yield satisfies $\varphi$ using $\Upsilon \wedge \gamma$.

For this, we need to restrict any quantifiers in $\varphi$ to the leaves by replacing $\exists x\phi$ with $\exists x \mathrm{Lf}(x) \wedge \phi$ and $\forall x\phi$ with $\forall x \mathrm{Lf}(x) \to \phi$. Further, we replace $M(z, y)$ by

$$\begin{aligned}m(z, y) \equiv \exists x(Q_\oplus(x) \wedge (\exists U_y, U_z(y \neq z \wedge \mathrm{Pt}(U_z, x, z) \wedge \mathrm{Pt}(U_y, x, y) \wedge \\ \forall r(U_y(r) \to ((r = y \vee \exists w(U_y(w) \wedge \mathrm{Rc}(w, r))) \wedge (r = y \vee r = x \vee Q_\odot(r)))) \wedge \\ \forall r(U_z(r) \to ((r = z \vee \exists w(U_z(w) \wedge \mathrm{Lc}(w, r))) \wedge (r = y \vee r = x \vee Q_\odot(r)))))))\end{aligned}$$

which is very similar to $\Upsilon$.

This is already everything we need to do, and thus we can conclude that

$$w \models \varphi \Leftrightarrow \text{there exists a tree } T \text{ with yield } w \text{ such that } T \models \Upsilon \wedge \gamma$$

Thus, we have proved both directions and see that the context free languages are exactly captured by FO(∃Match) □

### B.2.6 Context-Sensitive Languages

Here, we show the equivalence between context-sensitive languages and nondeterministic Turing machines using $\mathcal{O}(n)$ space. The two directions of the proof were presented separately in [Kur64] and [Lan63].

**Theorem B.7.** *The class of context-sensitive languages is exactly the class of languages accepted by a linear bounded nondeterministic Turing machine.*

*Proof.* For the direction from grammar to Turing machine, we only need to show that our Turing machine can simulate a derivation backwards. We know that every context-sensitive language has a noncontracting grammar $G$. Using this fact, we can construct a nondeterministic Turing machine $N$ which scans the current tape and whenever it recognizes a pattern of the right-hand side of a production rule in $G$, it decides whether it replaces it or not by the left-hand side of the rule. If some computation branch of $N$ ends up with only the start symbol of $G$ on the tape, we accept. Essentially, $N$ simulates a derivation of $w$ from the start symbol backwards. Because we try all possibilities by nondeterminism, we know that if $N$ does not accept $w$, there is no derivation ending in $w$ from the start symbol of $G$. As we assumed $G$ is noncontracting, replacing the right-hand side of a rule with the left-hand side never makes the word longer, and thus we only need $\mathcal{O}(|w|)$ space.

The other direction works by explicitly defining a grammar which simulates any linear bounded automaton backwards. Without loss of generality, we include an end marker #.

Let $N = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\rangle$ be a NSPACE[$\mathcal{O}(n)$] Turing machine. Then, we construct a grammar $G = \langle V, \Sigma, P, S\rangle$ with $V = \bigcup_{q_i \in Q} \bigcup_{a_j \in \Gamma} \{b_{q_i,a_j}\} \cup \{S, L, R, \#\} \cup \bigcup_{a_w \in \Gamma \setminus \Sigma} \{a_w\}$. The $b_{q,a}$ represent a position on the tape including the actual state and the actual character, in addition we also have the start state, the end marker and some utility non-terminals.

We now add the following rules to $P$:

- For each $a_i \in \Gamma$, we add a rule $S \to Lb_{q_{accept},a_i}R$ to $P$. These rules mean that we are in a final accept state. To extend the final tape, we add again for each $a_i \in \Gamma$ the rules $L \to La_i$, $L \to \#$, $R \to a_iR$ and $R \to \#$ to our rule set. These rules allow derivations from $S$ to an end tape of the form $\#a_{i_1} \dots b_{q_{accept},a_{i_j}} \dots a_{i_k}\#$.
- For each rule $\langle a_k, q_l, L\rangle \in \delta(q_i, a_j)$, we add rule $b_{a_w,q_l}a_k \to a_w b_{a_j,q_i}$ for every $a_w \in \Gamma$. Similarly, for each rule $\langle a_k, q_l, R\rangle \in \delta(q_i, a_j)$, we add rule $a_k b_{a_w,q_l} \to b_{a_j,q_i}a_w$ for every $a_w \in \Gamma$. We can clearly see that these rules simulate the nondeterministic Turing machine backwards.
- For the start, we include $\#b_{a_i,q_0} \to \#a_i$ for all $a_i \in \Sigma$ This rule allows us to say that we are at the start of our computation and "remove" the read-write head to get our initial input word.

As for any word in $G$ we can follow back the derivation to an accepting state, we have that both $N$ and $G$ define the same language.

Thus, we are done and have proven the equivalence of context-sensitive languages and linear bounded automata. □

### B.2.7 Recursive Languages

The most general case is interesting as it gives us a logical formalism for all problems which are computable at all.

The proof relies on Diophantine sets. Those sets are the sets that correspond to the tuples which have a solution for some Diophantine equation. A Diophantine equation is a polynomial equation $P$ with a tuple $\overline{x}$ of parameters and a tuple $\overline{y}$ of variables. A tuple $\overline{x}$ has a solution if there exists a tuple $\overline{y}$ such that $P(\overline{x}, \overline{y}) = 0$. The famous MRDP theorem states that the Diophantine sets are exactly the computable sets. At the same time, this shows that Hilbert's $10^{th}$ problem is unsolvable. A full proof of these facts can be found in [Mat96].

With this new characterization, there is a quite straightforward characterization of computable sets. The logic FO($\exists\mathbb{N}$) consists of all formulas $\phi(\overline{x}) \equiv \exists\overline{y}(\varphi(\overline{x}, \overline{y}))$ with $\varphi$ having only bounded quantifiers (of the form $\exists x < y$ or $\forall x < y$), addition, multiplication, equality, and any constant natural number [Ent20]. The existential quantifiers at the beginning are allowed to range over all the natural numbers.

These formulas can define exactly the Diophantine sets, as the formulas we present are exactly those that mean "there is a tuple $\overline{y}$ of natural numbers such that some polynomial is satisfied". Thus, they also define exactly all computable sets.

### B.2.8 Open questions

The domain of Descriptive Complexity is full of open questions as the proofs of lower bounds seems to be very difficult in most cases. Further, even separation between complexity classes which seem to take an exponential amount of resources compared to another one in practice can not be shown to be different.

#### $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$

The P vs. NP question is the most emblematic question in Descriptive Complexity theory. In practice, for any NP-complete problem, only exponential worst-case algorithms are known. This leads to the widely believed conjuncture that P $\neq$ NP. The problem is one of the seven Millennium Problems and a solution of equality or inequality is worth 1 Million US dollars.

The consequences of a solution stating that P = NP could have many practical advantages if it was constructive and had a low constant, as many important problems in research and logistics could be solved quickly. It would also mean the breakdown of most of modern cryptography, which relies on problem being intractable. On a conceptual level, it would mean that finding a proof to a problem is not harder than verifying its correctness, which would greatly impact the work of mathematicians. If a proof of the contrary would be known, this would focus the research more on the average case complexity of NP problems, but because of the continued lack of success on the question, this shift has already widely taken place.

### $\mathbf{NSPACE}[\mathcal{O}(n)] \stackrel{?}{=} \mathbf{DSPACE}[\mathcal{O}(n)]$

This problem is known under the name first Linear bounded automaton problem since its proposal by Kuroda in [Kur64], and asks if nondeterminism adds power in the context of bounded space. This comes from the fact that a NSPACE[$\mathcal{O}(n)$] Turing machine can be seen as a Turing machine with a linear bound on its space usage. This theorem is of interest as we know that NSPACE[$\mathcal{O}(n)$] is equivalent to the context-sensitive languages by appendix B.2.6.

Since the proposal, there were two advances. One is the proof that NSPACE is closed under complement. The contrary would have implied NSPACE[$\mathcal{O}(n)$] $\neq$ DSPACE[$\mathcal{O}(n)$] as DSPACE is closed under complement. The second advance is Savitch's theorem in section 3.3.1 which already gives a bound for simulating NSPACE using DSPACE machines. It is not known if this theorem is optimal, that is whether the blow-up by a power of 2 is optimal or if we can do better.

An equality would imply that the context-sensitive languages can be recognized by a deterministic linear bounded automaton, which could make recognizing words in context-sensitive languages easier and faster.

# C. Communication

## C.1 Email Dr. Dennis Komm

Sehr geehrter Dr. Komm,

Ich heisse Yaël Arn und bin Gymnasiast am Gymnasium Bäumlihof in Basel. Auf Empfehlung meiner Maturarbeitsbetreuungslehrperson schreibe ich Ihnen.

In meiner Maturarbeit beschäftige ich mich mit Finite Model Theory, insbesondere im Bezug auf deren Verbindung mit Formal Languages. Somit will ich insbesondere untersuchen, ob ich eine korrespondierende Logik zu den Context-Sensitive Languages finde, da ich keinen Artikel gefunden habe, der eine solche erwähnt.

Die Grundlagen der Theoretischen Informatik konnte ich mir im Rahmen des Schülerstudiums an der Uni Basel aneignen. Auch interessiere ich mich stark für Logik, Complexity Theory und Finite Model Theory und habe darüber Bücher gelesen.

Damit Sie meine Fähigkeiten einschätzen können: Auch bei der Schweizer Informatikolympiade, der Schweizer Mathematikolympiade sowie bei mehreren internationalen Olympiaden (MEMO, CEOI, BOI, WEOI) habe ich teilgenommen.

Jetzt zur eigentlichen Frage: Haben sie Tipps, wie ich vorgehen sollte und in welche Richtung ich forschen sollte? Und halten sie mein Unterfangen überhaupt für möglich?

Vielen Dank im Voraus für Ihre Zeit und Hilfe.

Viele Grüsse,

Yaël Arn

## C.2 Response Dr. Dennis Komm

Lieber Herr Arn

Bitte nehmen Sie es mir nicht übel, dass ich Ihnen noch nicht geantwortet habe. Ich habe Ihre Mail einem sehr guten Kollegen weitergeleitet, der sich sicherlich bei Ihnen melden wird. Momentan fehlen mir leider die zeitlichen Ressourcen, Ihnen persönlich detaillierter zu schreiben, was ich sehr bedauere.

Liebe Grüsse

Dennis Komm

## C.3 Email Dr. Gabriel Röger

Sehr geehrte Dr. Röger,

Ich heisse Yaël Arn (Nickname 42kangaroo, falls Sie sich daran erinnern) und bin Gymnasiast am Gymnasium Bäumlihof in Basel. Da ich durch Ihre Vorlesung "Theory of computer science" auf mein Maturarbeitsthema gestossen bin schreibe ich Ihnen jetzt.

In meiner Maturarbeit beschäftige ich mich mit Finite Model Theory, insbesondere in Bezug auf deren Verbindung mit Formal Languages. Somit will ich insbesondere untersuchen, ob ich eine korrespondierende Logik zu den Context-Sensitive Languages finde, da ich keinen Artikel gefunden habe, der eine solche erwähnt.

Neben Ihren Vorlesungen habe ich auch durch diverse Bücher über Logik, Complexity Theory und Finite Model Theory Einblicke in das Thema gehabt.

Damit Sie meine Fähigkeiten einschätzen können: Auch bei der Schweizer Informatikolympiade, der Schweizer Mathematikolympiade sowie bei mehreren internationalen Olympiaden (MEMO, CEOI, BOI, WEOI) habe ich teilgenommen.

Jetzt zur eigentlichen Frage: Haben sie Tipps, wie ich vorgehen sollte und in welche Richtung ich forschen sollte? Und halten sie mein Unterfangen überhaupt für möglich?

Vielen Dank im Voraus für Ihre Zeit und Hilfe.

Viele Grüsse,

Yaël Arn

## C.4 Email Dr. Angelika Steger

Sehr geehrte Dr. Steger,

Ich heisse Yaël Arn und bin Gymnasiast am Gymnasium Bäumlihof in Basel. Auf Empfehlung von Charlotte Knierim schreibe ich Ihnen.

In meiner Maturarbeit beschäftige ich mich mit Finite Model Theory, insbesondere im Bezug auf deren Verbindung mit Formal Languages. Somit will ich insbesondere untersuchen, ob ich eine korrespondierende Logik zu den Context-Sensitive Languages finde, da ich keinen Artikel gefunden habe, der eine solche erwähnt.

Die Grundlagen der Theoretischen Informatik konnte ich mir im Rahmen des Schülerstudiums an der Uni Basel aneignen. Auch interessiere ich mich stark für Logik, Complexity Theory und Finite Model Theory und habe darüber Bücher gelesen.

Damit Sie meine Fähigkeiten einschätzen können: Auch bei der Schweizer Informatikolympiade, der Schweizer Mathematikolympiade sowie bei mehreren internationalen Olympiaden (MEMO, CEOI, BOI, WEOI) habe ich teilgenommen.

Jetzt zur eigentlichen Frage: Haben sie Tipps, wie ich vorgehen sollte und in welche Richtung ich forschen sollte? Und halten sie mein Unterfangen überhaupt für möglich?

Ein volles Mentoring suche ich aktuell nicht, eine Antwort in der Form einer Mail, eines Telefonates oder eines kurzen Treffens würde mich aber freuen.

Vielen Dank im Voraus für Ihre Zeit und Hilfe.

Viele Grüsse,

Yaël Arn

## C.5 Response Dr. Královič

Dear Mr. Arn,

Dr. Komm asked me to write you some tips about your planned thesis; I am working in his group as senior scientific assistant. I hope it is ok if I write in English.

As you are likely aware, context-sensitive languages are exactly the languages that can be recognized by nondeterministic Turing machines in linear space. This class is often denoted as NSPACE(n) = nondeterministic space O(n).

I am not really an expert in the field of descriptive complexity theory, but I am not aware of logic-based characterization of this class. There are, however, some related results that are known. For example, characterization of DSPACE(n^k) is known (see e.g. `https://people.cs.umass.edu/~immerman/descriptiveComplexity.html`), where DSPACE(f(n)) denotes the class of languages accepted by deterministic Turing machines using space O(f(n)). NSPACE(n) is trivially a superset of DSPACE(n), and it is also a subset of DSPACE(n^2) (due to Savitch's theorem). This immediately gives two logic-based characterizations that bound NSPACE(n) from below and from above. This observation is already mentioned in `https://people.cs.umass.edu/~immerman/pub/ams_notices.pdf` as Theorem 2.

To get an exact logic-based characterization of NSPACE(n) may be not easy, most likely, it is a very ambitious goal. You could try to adapt the results for DSPACE(n^k) to NSPACE; even if you do not succeed there, explaining the proof ideas and showing why they are not so easy to transfer to NSPACE would make a valuable thesis. Augmenting such results with explanations of the upper and lower bounds of NSPACE(n) would, in my opinion, form a very solid "Maturaarbeit".

If you have more questions, feel free to ask. I wish you good luck in writing your thesis and I hope you will also enjoy it.

Best regards,

Richard Kralovic

## C.6 Further Communication with Dr. Královič

Dear Dr. Královič,

First of all, thanks a lot for your response and your time!

English is ok for me, in fact I'm writing my Matura project in English.

I effectively wasn't aware of the characterisation of DSPACE(n^k), so this is very valuable advice into which I will look. I had found upper / lower bounds using other formal languages, notably `https://link.springer.com/chapter/10.1007/BFb0022257` (existential SO restricted to binary matching predicates) for context-free languages and FO(∃N) for enumerable languages (As described in the book "Descriptive Complexity" by Neil Immermann), but they had the disadvantage of being less tight.

If (or rather when) I have more questions, I will write you another e-mail, thanks again for giving me this possibility!

Best,

Yaël Arn

## C.7 Question about writing my MA in English

Sehr geehrter Herr Ziegler, sehr geehrter Herr Leuthardt,

Gerne würde ich meine Maturarbeit über “Theoretische Informatik und Logig” auf Englisch schreiben, da die mathematische Literatur zum Grossteil auf Englisch ist und ich Mathematik lieber in dieser Sprache löse. Ist dies möglich?

Vielen Dank und viele Grüsse,

Yaël Arn

Sehr geehrter Herr Ziegler, sehr geehrter Herr Leuthardt,

Da ich gerne bald mit der schriftlichen Arbeit anfangen würde, wäre ich froh um eine Rückmeldung.

Viele Grüsse,

Yaël Arn

Lieber Yael

Die Schulleitung hat Ihren Antrag genehmigt. Sie dürfen die Maturaarbeit auf Englisch schreiben.

@Aline Steiner: Auflage der Schulleitung: Koreferent*in muss über entsprechende Englischkenntnisse verfügen, um die Arbeit berwerten zu können.

Liebe Grüsse

Andreas Leuthardt

# D. Notes

## D.1 Grammars

Equivalent: surrounded by context, length left $\leq$ length right, Kuroda normal Form

## D.2 Summary

$FO[t(n)]$: $A \in S \Leftrightarrow A \models \left([QB]^{t(|A|)} M_0\right)(\overline{c}/\overline{x})$
$VAR[v+1] = \bigcup_{c=1}^{\infty} FO-VAR[2^{cn^v}, v+1]$: Unbounded iterations, using at most $v+1$ vars
$ITER$[arity $k$]: Simultaneous iterations of relations of arity k as in LFP (no monoticity requirement)
$VAR[k+1,r]/ITER$[arity $k,r$]: Same plus restricted variables of total bits $r + \mathcal{O}(1)$
$DSPACE[s(n)] = VAR\left[k+1, \log\left(\frac{s(n)}{n^k}\right)\right] = ITER[\text{arity } k, \log\left(\frac{s(n)}{n^k}\right)]$ with $k = \lfloor \log_n(s(n)) \rfloor$ Problem for generalisation: For $C_t(\overline{x}, \overline{b})$ (tape cell $\overline{x}$ at time $t$ is $\overline{b}$), for nondeterminism we could have the same, but other tape cells are not independent.
$FO-VAR[s(n), v(n)]$: $VAR[s(n)]$ + constant number of $v(n)\log(n)$ bit variables, which can only be tested by $BIT$.
$SO[t(n), \text{arity } k]$: $FO[t(n)]$, but with quantification over relation variables of arity at most k
$NSPACE[n^k] = SO(\text{arity } k)(TC) \subseteq SO(\text{arity } k)[n^k]$

All formulas of $SO(\text{arity } k)(TC)$ can be written in the form

$$(TC_{struct_1, struct_2}\alpha)(\overline{false}, \overline{true})$$

where $\alpha$ is quantifier-free and both structs have at most $O(n^k)$ bits

$$SO(\text{arity } k)[t(n)] = FO-VAR\left(t(n), \frac{n^k}{\log(n)}\right)$$

$$\text{DSPACE}[s(n)] \subseteq \text{NSPACE}[s(n)] \subseteq \text{FO-VAR}\left[s(n), \frac{s(n)}{\log(n)}\right] \subseteq ATIME[s(n)^2] \subseteq \text{DSPACE}[s(n)^2]$$

Proof works by looking at alternating time paths of length $2^r$ in the $NSPACE$ computation graph.

## D.3 Extending $C_t(\overline{x}, \overline{b})$

Adding an additional variable for non-deterministic choices $C_t(\overline{c}, \overline{x}, \overline{b})$, where $\overline{c}$ are the non-deterministic choices, makes the states size grow exponentially in $O(2^{cn^k})$ (one for each move of

the NTM). So we have way more memory then $ITER[k+2]$, which makes it strictly more powerful. We need to have all the choices as we can only inductively define $C_t$ using the previous related choices.

## D.4 Direct transformation of SO(arity $k$)($TC$) to FO-VAR $\left[1, \frac{n^k}{\log n}\right]$ ($TC$)

$SO$(arity $k$)($TC$) $\subseteq FO-VAR\left[1, \frac{n^k}{\log n}\right]$ ($TC$): We know that any formula in $SO$(arity $k$)($TC$) can be written in the form

$$(TC_{struct_1, struct_2}\alpha)(\overline{false}, \overline{true})$$

where $\alpha$ is quantifier-free. Using the equivalence $BIT(Y, \overline{x}) \Leftrightarrow Y(\overline{x})$, we can write anything in $\alpha$ using $FO-VAR\left[1, \frac{n^k}{\log n}\right]$. Any relation in $struct_1, struct_2$ can also be rewritten using the extended variables. As we now any Relation with $false$ is converted to 0 and any relation with $true$ is converted to 1, we have $(\overline{false}, \overline{true})$ that can now be represented by $(\overline{0}, \overline{max})$.

$SO$(arity $k$)($TC$) $\supseteq FO-VAR\left[1, \frac{n^k}{\log n}\right]$ ($TC$): We know that any formula in $FO(TC)$ can be written as $(TC_{struct_1, struct_2}\alpha)(\overline{0}, \overline{max})$. We can extend this result to include the extended variables. For the usage of the extended variables in $BIT$, we have an atomic formula, so this is trivial. For quantification, we can handle it the same way as normal quantification with the modification of the "counting" variable with a tuple of $k$ counting variables, and the successor relation on these $k$ variables. We haven't shown yet that negation can be handled by this. In the proof that $FO(pos\, TC) = FO(TC)$, we only ask for a successor relation on the vertices (consisting of the extended variables and normal variables), which we assume from the beginning, so this also holds for $FO-VAR\left[1, \frac{n^k}{\log n}\right]$ $(pos\ TC) = FO-VAR\left[1, \frac{n^k}{\log n}\right]$ ($TC$). Now, we can do the same as above for the other direction to convert a formula of the form $(TC_{struct_1, struct_2}\alpha)(\overline{0}, \overline{max})$ to $(TC_{struct_1, struct_2}\alpha)(\overline{false}, \overline{true})$.

## D.5 Can we prove that there is no way to use $TC$ with a $FO$ language that has less then $O(n^k)$ variables?

We have Arity theorems Arity hierarchies Martin Grohe. Is this applicable because ordered structure? Diagonalisation in logic?

FO[$2^n$], SO[n] can both express REACH$_{dl}$

*Claim:* If extended variables are allowed everything, nothing changes.

*Define* Superrelations, using the extended variables. Does not work because input to big Not needed as we are working on string structures

If we allow extended variables for relations we have undefined for some, and if we check, we could "copy" bits into FO variable Numerical relations $\leq, SUC(X, Y), +, \times$ are definable via BIT anyways.

If no further relations are added, we have already that they reflect exactly some NSPACE complexity class, so we can not do better.

**Approach 2** We have that the FO[1, s(n)/log(n)](TC) correspond exactly to our complexity classes. The space hierarchy tells us that we can not do better while using these constructions.

Now assume we add iterated quantifier blocks. We have multiple ways of doing this. We can have the TC inside or outside

$$TC_{s(n)}([QB]^{t(n)}\varphi) \equiv TC_{O(n)}(\phi)$$

In both cases, we can replace the TC operator with a quantifier block iterated $o(n)$ times. So we have formulas of the form

$$[QB_1]^{s(n)}[QB_2]^{t(n)}\psi$$

We can include a counting variable of which counts to $o(n)$ and then switches block to get only one combined block.

so we then have a formula in FO[s(n) + t(n), s(n)/log(n)], which is also in FO[t(n), s(n)/log(n)](TC) or FO[s(n), s(n)/log(n)](TC) (or TC[FO[s(n), s(n)/log(n)]], but does just not use the TC operator. So if s(n) $\notin o(n)$ then this is equivalent or more difficult then showing that $SO$(arity $k$)$[n^k] = SO$(arity $k$)$(TC)$ Otherwise, if $t(n) \geq s(n)$ we have to show that $FO\left[t(n), \frac{s(n)}{log(n)}\right] = SO$(arity $k$)$(TC)$. The only problem arises if $s(n) > t(n)$, as then the generated new formula is not in the original class. What would happen is that then $SO(arity\ k)(TC) = FO\left[t(n), \frac{s(n)}{\log(n)}\right](TC) \subseteq FO\left[s(n), \frac{s(n)}{\log n}\right] \subseteq SO$(arity $k$)$[n^k]$ We know that $s(n)$ must be in $\Omega(\sqrt{n})$ by Savitch's theorem. Also if $s(n) \in o(n)$, we would have a stronger result then Savitch's theorem as then $NSPACE[n] \subseteq DSPACE[s(n)^2]$, which is an open problem since almost 50 years.

In the end, this proves that combining TC and quantifier iterations is as difficult either as showing an improvement of Savitch's theorem or is equivalent to having an alternative characterisation using only iterated quantifiers.

## D.6 Generalised quantifiers

By double arity we have FO(TC) = FO($Q_{TC}$) which is the tc quantifier. The tc Quantifier has type $\langle 2, 1, 1\rangle$ and $k$-vectorisation to make it generalised to vectorisations. There is a theorem which states that no family of quantifiers of arity $< 2k$ suffices to express these properties. So We can effectively say that they are the easyest.

This is true for FO-generalised quantifiers on unordered structures, but unclear for SO-gq

Problem is that we can query bits, so we have even more than just an order

Can we port the results directly?

## D.7 FO∃[n]

**FO∃[s(n)]** A set $S \subseteq$ STRUCT$[\tau]$ is in FO∃[s(n)] if and only if there exists formulas $M_i, N_j$, $0 \leq i \leq k, 1 \leq j \leq r$ from $\mathcal{L}(\tau)$, a tuple $\overline{c}$ of constants and a quantifier block

$$QB = [(\exists x_1.M_1)(\forall b_1.N_1)\dots(\exists x_k.M_k)(\forall b_r.N_r)]$$

such that for all $\mathcal{A} \in \text{STRUCT}[\tau]$, we have

$$\mathcal{A} \in S \Leftrightarrow \mathcal{A} \models ([QB]^{s(|\mathcal{A}|)} M_0)(\overline{c}/\overline{x})$$

FO∃-VAR[t(n), s(n)] is the same with extended variables

We can write TC as

$$\begin{aligned}
QB &\equiv (\forall b_1.M_1)(\exists \overline{Z})(\forall b_2)(\exists \overline{A}, \overline{B}.M_2)(\exists \overline{X}, \overline{Y}.M_3) \\
M_1 &\equiv \neg(\forall \overline{z}(\text{BIT}(\overline{X}, \overline{z}) \leftrightarrow \text{BIT}(\overline{Y}, \overline{z})) \vee \varphi(\overline{X}, \overline{Y})) \\
M_2 &\equiv (b_2 \wedge \forall \overline{z}(\text{BIT}(\overline{X}, \overline{z}) \leftrightarrow \text{BIT}(\overline{A}, \overline{z})) \wedge \forall \overline{z}(\text{BIT}(\overline{B}, \overline{z}) \leftrightarrow \text{BIT}(\overline{Z}, \overline{z})))\vee \\
&\quad (\neg b_2 \wedge \forall \overline{z}(\text{BIT}(\overline{Z}, \overline{z}) \leftrightarrow \text{BIT}(\overline{A}, \overline{z})) \wedge \forall \overline{z}(\text{BIT}(\overline{B}, \overline{z}) \leftrightarrow \text{BIT}(\overline{Y}, \overline{z}))) \\
M_3 &\equiv \forall \overline{z}(\text{BIT}(\overline{X}, \overline{z}) \leftrightarrow \text{BIT}(\overline{A}, \overline{z})) \wedge \forall \overline{z}(\text{BIT}(\overline{B}, \overline{z}) \leftrightarrow \text{BIT}(\overline{Y}, \overline{z}))
\end{aligned}$$

And then $[QB]^{cs(n)}(false)(\overline{false}/\overline{X}, \overline{true}/\overline{Y}) \equiv (TC_{\overline{X},\overline{Y}}\varphi(\overline{X}, \overline{Y}))(\overline{false}, \overline{true})$

Thus, we have NSPACE[s(n)] = FO-VAR[1, s(n)/log(n)](TC) ⊆ FO∃-VAR[s(n), s(n)/log(n)]

For the reverse containment, we can simulate the formula with a NSPACE[s(n)] machine by existentially guessing the existentially quantified variables and remembering the universal choices in $s(n)$ space. Does not work because we need some backtracking stuff, and thus remembering the existential vars.

Also, $FO - VAR[1, s(n)/log(n)](TC) \subseteq FO - VAR\left[\frac{s(n)}{\log(r(n))}, \frac{s(n)r(n)}{\log(n)}\right]$ by doing multiple steps at once, and universal quantification over $r(n)$ bit variables only.

## D.8 Alternating machines and FO-VAR

$ATIME - SPACE[s(n)^2, s(n)] \subseteq FO - VAR\left[s(n)^2, \frac{s(n)}{\log(n)}\right]$

We have a state space of $2^{\mathcal{O}(s(n))}$, so we need only $s(n)$ bit Variables to encode it. We can assume that there exist formulas $existencial(\overline{X})$ which is true if $\overline{X}$ is an existential state, and one $\varphi(\overline{X}, \overline{Y})$ telling us that we can transition from $\overline{X}$ to $\overline{Y}$.

Then, we have

$$\begin{aligned}
QB &\equiv (\forall b.M_1)(\exists \overline{A}.\varphi(\overline{A}, \overline{X}))(\forall \overline{B}.M_2)(\exists \overline{X}.\forall z(\text{BIT}(\overline{X}, \overline{z}) \leftrightarrow \text{BIT}(\overline{B}, \overline{z}))) \\
M_1 &\equiv \neg\forall z(\text{BIT}(\overline{X}, \overline{z}) \leftrightarrow \text{BIT}(\overline{Y}, \overline{z})) \\
M_2 &\equiv (existential(\overline{X}) \rightarrow \forall z(\text{BIT}(\overline{A}, \overline{z}) \leftrightarrow \text{BIT}(\overline{B}, \overline{z}))) \wedge \varphi(\overline{X}, \overline{B})
\end{aligned}$$

and

$$\left([QB]^{s(n)^2}(false)\right)(\overline{false}/\overline{X}, \overline{true}/\overline{Y})$$

$$\begin{aligned}
ATSR[t(n), s(n), r(n)] &\subseteq FO - VAR\left[r(n)\log\left(\frac{t(n)}{r(n)}\right), \frac{s(n)}{\log n}\right] \\
&\subseteq ATSR\left[r(n)s(n)\log\left(\frac{t(n)}{r(n)}\right), s(n), r(n)\log\left(\frac{t(n)}{r(n)}\right)\right]
\end{aligned}$$

The second containment is a consequence of how we simulate the formula with the alternating Turing machine, as we have $FO-VAR\left[a(n), \frac{s(n)}{\log n}\right] \subseteq ATSR[a(n)s(n), s(n), a(n)]$ by using the normal construction in Savitch's theorem.

In the first containment, we try to use the method of simulating NSPACE for each alternation. First, we show that

$$\sum_{j=0}^{r(n)} \log(a_j) \leq r(n) \log\left(\frac{t(n)}{r(n)}\right)$$

under the condition that $\sum_{j=0}^{r(n)} a_j = t(n)$.

This can be shown using Jensen's inequality.

$$\frac{\sum_{j=0}^{r(n)} \log(a_j)}{r(n)} \leq \log\left(\frac{\sum_{j=0}^{r(n)} a_j}{r(n)}\right) = \log\left(\frac{t(n)}{r(n)}\right)$$

as log is concave. Multiplying with $r(n)$ gives the claim.

$Y$ accepting $X$ start $I$ actual intermediate end $S$ actual start $E$ actual end $A, B$ copy vars $b_{path}$ if we are trying to find or not find a path

$$\begin{aligned}
QB \equiv\ & (\forall b_0.M_1)(\exists b_0.N_1)(\exists b_1.M_2)(\exists Z_e.M_3)(\forall Z.N_2) \\
& (\exists b_{2e})(\forall b_2.N_3)(\exists A, B.M_4)(\forall S, E.M_5)(\exists A, B.M_6)(\exists I, X.M_7)(\exists b_0.N_4)(\exists b_{path}.N_5) \\
M_1 \equiv\ & b_{path} \rightarrow \neg(I = Y \wedge (S = E \vee \varphi(S, E))) \\
N_1 \equiv\ & \neg b_{path} \rightarrow \neg(S = E \vee \varphi(S, E)) \\
M_2 \equiv\ & \neg b_1 \leftrightarrow (S = E \vee \varphi(S, E)) \\
M_3 \equiv\ & existential(Z_e) \leftrightarrow existential(X) \\
N_2 \equiv\ & b_{path} \rightarrow Z = Z_e \\
N_3 \equiv\ & \neg b_{path} \rightarrow b_2 = b_{2e} \\
M_4 \equiv\ & (b_1 \wedge ((\neg b_2 \wedge A = S \wedge B = Z) \vee (b_2 \wedge A = Z \wedge B = E))) \vee \\
& (\neg b_1 \wedge A = I \wedge (B = Y \vee (existential(B) \leftrightarrow \neg existential(I)))) \\
M_5 \equiv\ & S = A \wedge (((b_1 \vee existential(S)) \wedge E = B) \vee \\
& (\neg(b_1 \vee existential(S)) \wedge (E = Y \vee (existential(E) \leftrightarrow \neg existential(S))))) \\
M_6 \equiv\ & (\neg b_1 \wedge S = A \wedge E = B) \vee (b_1 \wedge X = A \wedge I = B) \\
M_7 \equiv\ & X = A \wedge I = B \\
N_4 \equiv\ & b_0 = b_{path} \\
N_5 \equiv\ & b_1 \rightarrow b_{path} = b_0
\end{aligned}$$

In this formula, we have $\left([QB]^{r(n)\log(t(n)/r(n))}(\neg b_{path})\right)(true/b_{path}, AS/S, X, I, S, E, AE/Y)$ is true if and only if the machine accepts.

$\implies FO-VAR\left[a(n), \frac{s(n)}{\log n}\right] \subseteq ATSR[a(n)s(n), s(n), a(n)] \subseteq FO-VAR\left[a(n)\log(s(n)), \frac{s(n)}{\log n}\right]$

# E. Independence declaration (German)

Ich, Yaël Arn, 4A

bestätige mit meiner Unterschrift, dass die eingereichte Arbeit selbstständig und ohne unerlaubte Hilfe Dritter verfasst wurde. Die Auseinandersetzung mit dem Thema erfolgte ausschliesslich durch meine persönliche Arbeit und Recherche. Es wurden keine unerlaubten Hilfsmittel benutzt. Ich bestätige, dass ich sämtliche verwendeten Quellen sowie Informanten/-innen im Quellenverzeichnis bzw. an anderer dafür vorgesehener Stelle vollständig aufgeführt habe. Alle Zitate und Paraphrasen (indirekte Zitate) wurden gekennzeichnet und belegt. Sofern ich Informationen von einem KI-System wie bspw. ChatGPT verwendet habe, habe ich diese in meiner Maturaarbeit gemäss den Vorgaben im Leitfaden zur Maturaarbeit korrekt als solche gekennzeichnet, einschliesslich der Art und Weise, wie und mit welchen Fragen die KI verwendet wurde. Ich bestätige, dass das ausgedruckte Exemplar der Maturaarbeit identisch mit der digitalen Version ist. Ich bin mir bewusst, dass die ganze Arbeit oder Teile davon mittels geeigneter Software zur Erkennung von Plagiaten oder KI-Textstellen einer Kontrolle unterzogen werden können.

Ort & Datum

Unterschrift