Gymnasium Bäumlihof

Maturaarbeit

Theoretical Informatics: Formal languages and finite model thoery

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

Written by:
Yaël Arn, 4A

Supervisor:
Aline Sprunger

Coreferent:
Bernhard Pfammatter

15th July 2024, 4058 Basel

Contents

Forword

1. Introduction

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 fulfills some given property. 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 Σ is a finite set of symbols

Definition 2.2 (Word)**.** A word over some alphabet Σ is finite sequence of symbols from Σ. We denote ε as the empty word, Σ^* as the set of all words over Σ and $|w|$ as the number of symbols in w.

The concatenation of two words or symbol is written after each other, examples are ab and $\Sigma^* a \Sigma^*$ (the set of all words containing at least one a).

Definition 2.3 (Formal Language)**.** A formal language is a set of words over some alphabet Σ, that is a subset of Σ^*

For any computational decision problem, we can then 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 in some finite way, namely using grammars, but other formalisms are equivalent.

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

Σ 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

The non-terminal symbols are symbols that are not in the end alphabet Σ and exist for the purpose of steering the process of word generation. Further, the rules dictate that there must be at least one non-terminal symbol on the left-hand side of the production rule, as $(V \cup \Sigma)^*$ contains all words consisting of symbols from V and Σ, and thus $(V \cup \Sigma)^* V (V \cup \Sigma)^*$ is the language of all words containing at least one non-terminal symbol. We normally 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 for some words $x, y, z \in (V \cup \Sigma)^*$ and u' is of the form $xy'z$
- there exists a rule $y \to y'$ in R

We say that a word is in the *generated language* of a grammar if it 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 grammar that describes the language.

2.2.2 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 2.2. 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.

These languages have been studied quite thoroughly and have multiple equivalent formalisms:

- The language is recognised Deterministic finite automata, which process the input word one character at a time (for a formal definition see [Rög23])

- 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 section 3.3.1 and [Str94].

2.2.3 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 wich are then converted to specific expressions of that type.

Example 2.3. Consider the grammar $\langle\{\mathbf{Exp}, \mathbf{NumF}, \mathbf{Num}\}, \{0, 1, (,), -, +\}, R, \mathbf{Exp}\rangle$ with

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

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 characterisation of the context-free languages using logic, see section 3.3.2.

2.2.4 Context-Sensitive Languages

The most important category of languages for this work have multiple restrictions on the grammars which produce the same set.

One restriction is that all rules are of the form $\alpha\beta\gamma \to \alpha\varphi\gamma$ with $\alpha, \gamma \in (\Sigma \cup V)^*$, $\beta \in V$ and $\varphi \in (\Sigma \cup V)^+$. 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 have all grammars with $u \leq v$ for any rule $u \to v$, in addition to the special case with the start variable mentioned above. These grammars are called noncontracting.

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

- $A \to BC$
- $AB \to CB$

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

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

Example 2.4. 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 are the linearly bounded nondeterministic Turing machines which can only write on the tape cells that contained a non-blank symbol. This and an equivalent extension of Second-Order logic will be proven in section 3.3.3.

2.2.5 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 recognisable 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 characterisation using logic, again refer to section 3.3.4.

3. Descriptive Complexity

3.1 Aims

3.2 Important Results

3.2.1 NSPACE[$s(n)$] $\subseteq$ DSPACE[$s(n)^2$]

3.2.2 FO(LFP) = P

3.3 Results concerning the Chomsky hierarchy

3.3.1 Regular Languages

3.3.2 Context-Free Languages

3.3.3 Context-Sensitive Languages

3.3.4 Recursive Languages

3.4 Open questions

3.4.1 P$\stackrel{?}{=}$NP

3.4.2 NSPACE[$O(n)$]$\stackrel{?}{=}$DSPACE[$O(n)$]

4. Personal Contribution

5. Results

6. Conclusion and Direction

Thanks

List of Figures

Listings

Bibliography

[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).

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

[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.

[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.

A. Mathematical Background

The definitions are taken from the lectures Discrete Mathematics in Computer Science [HR22] and Theory of Computer Science [Rög23] and also 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 tuples with an element of A and an element of B

Cartesian power A^k denotes the cartesian product of A with itself repeated k times

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 equality($=$), an ordering relation $\leq$, and $BIT(x, y)$, which means that the y^{th} bit of x is set in binary notation, to exist.

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 (We omit functions as they can be simulated by a relation in our case)

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 the truth of the relations have a truth value for each a_i-tuple from $|\mathcal{A}|^{a_i}$

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

conjuction If φ and ψ are formulas, $(\varphi \wedge \psi)$ is a formula

disjuction If φ and ψ are formulas, $(\varphi \vee \psi)$ is a formula

negation If φ is a formula, $\neg\varphi$ is a formula

Existencial Quantification If φ is a formula, $\exists x\varphi$ is a formula

Universal Quantification If φ is a formula, $\forall x\varphi$ is a formula

Semantics For any structure, we can assign a truth value to any formula (by assigning values from the universe to free variables if they exist in the formula). We say $\mathcal{A}$ satisfies ϕ (where ϕ is taken over the vocabulary of $\mathcal{A}$), denoted $\mathcal{A} \models \phi$ if and only if ϕ is true under the interpretation of the constant and relations of $\mathcal{A}$. This is inductively defined as follow:

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

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

disjuction 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 $\mathcal{A} \not\models \varphi$

Existencial Quantification We have $\mathcal{A} \models \exists x\varphi$ if and only if there exists a $y \in |\mathcal{A}|$ such that $\mathcal{A} \models \varphi(y)$ (where $\varphi(y)$ denotes φ 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 $\mathcal{A} \models \varphi(y)$

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 abreviate 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 Existencial Quantification If φ is a formula, then $\exists V\varphi$ is a formula

SO Universal Quantification If φ is a formula, then $\forall V\varphi$ is a formula

SO Semantics Here we also need to extend the FO semantics

SO Existencial Quantification We have $\mathcal{A} \models \exists V\varphi$ if and only if there exists a relation U over $|\mathcal{A}|$ such that $\mathcal{A} \models \varphi(U)$ (where $\varphi(U)$ denotes φ 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 $\mathcal{A} \models \varphi(U)$

A.4 Turing Machines

Turing machines are the most common model of computation. We abreviate Turing Machines as TM

Informal definition A turing machine is a 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

Σ is the set representing the symbols of which the input word on the tape can consist

Γ is the set of symbols which can be written or read on the tape

δ is the transition function, with $\delta : \Gamma \times Q \to \Gamma \times Q \times \{L, R\}$. So when a TM is in state n and reads a on the tape, δ 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 accept state

q_{reject} the reject state

Turing computation At the beginning, the TM is in the start state, the input is written in a consecutive way on the tape and the read/write head is on the first character of the input word. In consecutive steps, the machine state then changes according to the transition function. If at some point the machine enters the accept or the reject state, the computation halts, and the TM is said to have accepted / rejected the input. In this work we will ignore the tape content after the computation and focus on decision problems.

Decidability If a TM 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.

Nondetermenistic TM (NTM) We can extend the transition function δ to allow multiple transitions from a given state. If there exists any computational path which leads to an accept state, the NTM accepts. This is not analog to how real sequential computers work, but allows interesting results, and is as powerfull as a normal deterministic TM.

Church-Turing Thesis According to the Church-Turing Thesis, this formalism is equivalent to what any computer can compute.

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
