ABSTRACT

Title of Thesis: A COMPARATIVE RESEARCH BETWEEN THE RELATION OF VARIOUS SATISFIABILITY PROBLEMS AND COMPLEXITY CLASSES

Georgios Tsimos
Master of Science, 2019

Thesis directed by: Assistant Professor Eugenie Foustoucos

In this study we present a chronological comparison of multiple results, each of which proves the Completeness of a Satisfiability problem for a corresponding Complexity Class. We examine several important results that all seem to be the result of using the notion of encoding the behavior of Turing Machines using Logic. Via this idea, we give an alternative and to our best of knowledge novel proof for one of these results, a proof that does not rely immediately on any prior result. Specifically, we prove again that 2-SAT is NL-Complete and do so without reducing it to any other NL-Complete problems. Besides this novel result, we survey the proofs of several similar, already well-established results. We present Trakhtenbrot's Theorem, the NP-Completeness of 3-SAT and the PSPACE-Completeness of TQBF. A comparison of the similarities and natural differences between the techniques behind these results follows and some interesting conclusions and directions for further analysis are being drawn.

A COMPARATIVE STUDY OF THE RELATIONSHIP BETWEEN SATISFIABILITY PROBLEMS AND COMPLEXITY CLASSES

by

Georgios Tsimos

Thesis submitted to the Faculty of the Master's Program of the
Athens University of Economics and Business in partial fulfillment
of the requirements for the degree of
Masters of Computer Science
2019

Acknowledgments

I would like to thank several people who helped me conclude this work.

First and foremost, I owe my gratitude to my advisor, Professor Eugenie Foustoucos for sacrificing a lot of her personal time in order to help me in my efforts of understanding small, new parts of a vast field. She has given me the opportunity to better understand how to properly conduct research and continuously encouraged me to question and challenge my self, while at the same time she always remained patient and helpful.

I would also like to thank Professor Evangelo Markaki and Professor Iordani Koutsopoulo for agreeing to serve on my thesis committee and for using their time resources for reading and understanding this work.

Many thanks are also due to all the members of the faculty of the Master's Program of Computer Science at AUEB, for their tremendous work and for the opportunity they have given to all of us students to enrich our knowledge over so many important fields of Computer Science.

A special thanks is due to my colleagues in the Program, since they have made a very difficult year seem like it run by just in days. We managed to create a great,tight nucleus of friends and colleagues and that is truly rare for a, seemingly, random group of people.

Thank you all!

Table of Contents

List of Abbreviations

CNF	Conjunctive Normal Form
FO	First Order logic
L	Linear space complexity class
NL	Nondeterministic Linear space complexity class
NP	Nondeterministic Polynomial time complexity class
P	Polynomial time complexity class
PSPACE	Polynomial SPACE complexity class
SAT	SATisfiability of formula problem
SO	Second Order logic
TM	Turing Machine
TQBF	Totally Quantified Boolean Formula satisfiability problem

Chapter 1: Basic Definitions

1.1 Introduction

The studies of Complexity Theory have been of great importance through the last six decades. However, while the core of this Theory has been thoroughly researched, various questions remain unanswered. During the efforts for unearthing the most celebrated results, researchers have tried different approaches. One of the most promising of them -after it was initially presented by Ronald Fagin in his Ph.D. Thesis [1] is the field of Descriptive Complexity, where each Complexity Class is equivalently defined by the type of logic required for the Languages in the Class to be expressed.
Fagin's work, however, wasn't the first instance in which the idea of encoding Turing Machines using Logic has been deployed. In fact, through out the years, this notion has been central to several results regarding Complexity Theory.

In the present work, we examine several important results that have occurred by use of this notion, of encoding the behavior of Turing Machines using Logic. We use this notion to give an alternative proof for a Completeness result regarding a well-known problem, i.e. that 2-SAT is NL-Complete. Our proof differs from the existing proof, which uses a logarithmic-space reduction from an already established NL-Complete problem, the Reachability problem. Instead, we prove the result by constructing a method of reducing the decidability of any problem in NL to the satisfiability of a 2-CNF, Boolean formula.

Through this procedure, we give another valid example of a Complexity Class in which a specific problem of Logic is Complete for the Class.

We compare the several proofs and try to understand the common traits behind all these results. At the same time, we use this opportunity in order to better understand some other fundamental results of Descriptive and Computational Complexity that relevant to our work.

1.2 Logic

We now briefly review some standard definitions from mathematical logic. For further readings regarding Logic, Enderton's [2] is a Classic textbook.

Definition 1.2.1. A **vocabulary** σ is a collection of constant symbols $(c_1, c_2, \dots, c_n, \dots)$, relation symbols or predicates $(R_1, R_2, \dots, R_n, \dots)$ and function symbols $(f_1, f_2, \dots, f_n, \dots)$. Every predicate and every function symbol has an associated **arity**, which corresponds to the dimension of its domain.

A **σ-structure** or model $\mathcal{U} = \Big(\mathcal{A}, \{c_i^{\mathcal{U}}\}, \{R_i^{\mathcal{U}}\}, \{f_i^{\mathcal{U}}\}\Big)$ consists of a universe $\mathcal{A}$ and an interpretation of each c_i, R_i, f_i from σ as $c_i^{\mathcal{U}} \in \mathcal{A}, R_i^{\mathcal{U}} \subseteq \mathcal{A}^k, f_i^{\mathcal{U}} : \mathcal{A}^l \to \mathcal{A}$, respectively.

A structure $\mathcal{U}$ is called **finite** if its universe $\mathcal{A}$ is a finite set. Vocabularies that contain only relation symbols and constants are called **relational** vocabularies. Throughout this work, we assume that every vocabulary is finite and relational. If σ is a relational vocabulary, then STRUCT[σ] denotes the Class of all finite σ-structures.

Definition 1.2.2. We assume a countably infinite set of variables. Variables will be typically denoted by $x, y, z, \ldots$, with subscripts and superscripts. We inductively define terms and formulae of the first-order predicate calculus over vocabulary σ as follows:

- Each variable x is a term.
- Each constant symbol c is a term.
- If $t_1, \ldots, t_k$ are terms and f is a k-ary function symbol, then $f(t_1, \ldots, t_k)$ is a term.
- If t_1, t_2 are terms, then $t_1 = t_2$ is an atomic formula.
- If $t_1, \ldots, t_k$ are terms and R is a k-ary relation symbol, then $R(t_1, \ldots, t_k)$ is an atomic formula.
- If ϕ_1, ϕ_2 are formulae, then $\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$, and $\neg\phi_1$ are formulae.

  (We keep both $\wedge, \vee$, although one suffices, so as to be able to use the notion of CNF. However, we omit $\rightarrow, \leftrightarrow$, since they can be expressed by use of the other connectives.)
- If ϕ is a formula, then $\exists x \phi$ and $\forall x \phi$ are formulae.

If we only use Propositional (i.e. Boolean) variables and the Boolean connectives $\vee, \wedge, \neg$, then the resulting formulae are **Boolean formulae**.

A Boolean formula is in **k-CNF** (Conjunctive Normal Form), if it contains only conjunctions($\wedge$) of clauses, each of which is a disjunction($\vee$) of k or less Boolean literals(variables and their negations). A Boolean formula is **satisfiable** if there exists a satisfying assignment for its variables, i.e. if there exists an allocation of TRUE-FALSE between its variables, such that the complete formula is TRUE.

If we extend the Propositional Logic in a way such that Boolean variables accept quantification ($\forall, \exists$) over a Boolean formula, then the resulting formula is a **Quantified Boolean formula** (QBF).

Note however that the resulting Logic is an extended version of the definition of Propositional Logic. A Quantified Boolean formula where every variable is quantified is called a **sentence** (or fully Quantified Boolean Formula) and is either true or false. This might not be straightforward in first sight, so we will clarify it with an example.

Suppose we have a formula of the form: $(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3)$. This is a Boolean formula. If we use quantification, then we are no longer in Propositional Logic. Instead, we will have a Quantified Boolean formula over the extended (quantified) definition of Propositional Logic. Suppose we have a new formula:

$\forall x_1 \exists x_2 \forall x_3 (x_1 \vee x_2) \wedge (x_2 \vee \neg x_3)$. This formula is a fully Quantified Boolean Formula. This formula suggests that for any value we give to x_1 (True or False), there must exist a value we can give to x_2 such that for any value we give to x_3 (True or False), the initial formula is satisfied.

The above sentence however is a statement, that will always be either True or False, and since it is logically encoded by our latest formula, we define such formulas as sentences and they are always either True or False. For example, the above sentence is always True, since if we set x_2 to be true, both clauses of the initial formula are satisfied independently of the values of x_1 and x_3.

1.3 Complexity Theory

Definition 1.3.1. A Turing Machine (T.M.) say M, is a tuple $(Q, \Sigma, \Delta, \delta, q_0, Q_a, Q_r)$, such that:

- Q is a finite set of states.
- Σ is the finite input alphabet.
- Δ is the finite tape alphabet; it contains Σ and a specific blank symbol $\sqcup$.
- $\delta : Q \times \Delta \to 2^{Q\times\Delta\times\{left,right\}}$is the transition function.
- $q_0 \in Q$ is the initial state of M.
- $Q_a, Q_r \subseteq Q$ s.t. $Q_a \cap Q_r = \emptyset$ are the set of accepting and rejecting states of the machine respectively. They are all halting states for the machine.

.

A T.M. is called **deterministic** if δ is actually a function $Q \times \Delta \to Q \times \Delta \times \{left, right\}$. In any other case, the machine is **non-deterministic**.

A T.M. might have one or more tapes. In general we can assume that any T.M.

has one tape and one head. When otherwise (as it will be the case for problems in NL) we will specifically explain so. No matter what, a T.M. with one tape is asymptotically as expressible as any T.M. with a finite amount of tapes.

A **configuration** of a T.M. encodes an instance of the machine at a given time while running on a specific input. It specifies the contents of the tape, the state, and the position of the head as follows.

Let the tape contain the word $w = w_1, w_2, ..., w_n$, where $w_i \in \Delta$ is the symbol in the i^{th} position of the word and consequently, the tape. Assume that the head is in position j. If the T.M. is in state q, we denote this configuration by $|q|j|w_1|w_2|\dots|w_n|$. A configuration $C_2 = (q_2, h_2, w_2)$ immediately follows another configuration $C_1 = (q_1, h_1, w_1)$ iff $(q_2, w_2^{h_1}, h_2) \in \delta(q_1, w_1^{h_1})$. A configuration C_i is an **accepting** configuration if $q_i \in Q_a$.

We will now remind the definitions of the several Complexity Classes we are going to encounter throughout the remaining chapters.

Definition 1.3.2. L is the Class of Languages decidable in *logarithmic space* on a two-tape, deterministic T.M. , or else, L= SPACE($\log n$).

NL is the Class of Languages decidable in *logarithmic space* on a two-tape, non-deterministic T.M. , or else, NL= NSPACE($\log n$)

Definition 1.3.3. P is the Class of Languages decidable in *polynomial time* on a single-tape, deterministic T.M. , or else, P= $\bigcup_k$ TIME(n^k).

Definition 1.3.4. A verifier for a Language A is an algorithm V, where

A $= \{w|\text{V accepts } (w,c), \text{for some string } c\}$.

The time Complexity of a verifier takes into consideration only the size of w.

A Language A is **polynomially verifiable** if it has a polynomial verifier.

Definition 1.3.5. NP is the Class of polynomially verifiable Languages ,or else, NP= $\bigcup_k$ NTIME(n^k).

Definition 1.3.6. PSPACE is the Class of Languages decidable in *polynomial space* on a singe-tape, deterministic T.M. , or else, PSPACE= $\bigcup_k$ SPACE(n^k)

Definition 1.3.7. Consider a logarithmic-space T.M. with three tapes: one read-only input tape, one read/write work tape and one write-only output tape, where the work tape must contain at most $\mathcal{O}(\log n)$ symbols. Such a T.M. computes a function $f : \Delta^* \to \Delta^*$.

Then, Language A is **logarithmic-space reducible to** Language B ($A \leq_{\log} B$) if A is a mapping reducible to B by use of a logarithmic-space computable function f, computed by a T.M. as the above.

Definition 1.3.8. Consider a polynomial-time T.M. that computes a function $f : \Delta^* \to \Delta^*$.

Then, Language A is **polynomial-time reducible to** Language B ($A \leq_p B$) if A is a mapping reducible to B by use of a polynomial-time computable function f, computed by a T.M. as the above.

Definition 1.3.9. Problem A is **C-Hard** for a Complexity Class C if A is at least as hard as any problem in C, or else, every problem in C is appropriately reduced to A. Problem A is C-Complete for a Complexity Class C, if A is in C and A is C-Hard.

We consider the readers to be familiar with the Big-Oh notation. For most part of this work, we will use the theoretical definitions of Complexity Classes, following mostly the definitions of [3]. However, most cases of asymptotic, Big-Oh complexity have an immediate correspondence to some Complexity Class. Specifically, the Complexity Class of time $TIME(t(n))$ is the set of languages decidable from a deterministic T.M. of asymptotic running time $O(t(n))$.

With all these in mind, we are ready to proceed with the several proofs and results.

1.4 Outline of the Thesis

In Chap. 2, we present Trakhtenbrot's Theorem, since it is the first theorem which integrates the important notion, namely that one can prove equivalence between the (un)satisfiability of a type of logical sentences and the halting (or acceptance/rejection) of a Turing Machine. We show the alternative proof presented in [4], since it gives a better example of a more general method for proving the equivalence between T.M.s and logical sentences(formulas).

In Chap. 3, we continue by giving the proofs of the Cook-Levin Theorem, stating that SAT is NP-Complete and of the theorem which states that TQBF is PSPACE-Complete. These two proofs are presented in the same chapter since during the time we were trying to extract their “essence”, we came to the conclusion that they shared many similar notions. Thus, it seems better in order to understand the techniques used in the proofs, as well as their results, it is preferable to study both of them together.

In Chap. 4, we present our result. We follow Chap. 2 and 3 by using similar ideas and building upon prior knowledge for proving that 2-SAT is NL-Complete. This result is known and the existing proof uses a log-space reduction of the problem 2-SAT from another, already known NL-Complete problem. Here, we present a different and novel (to the best of our knowledge) proof. Our proof uses properties inherent to the problems in the Class of NL and by explaining a way to reduce any such problem to an appropriate 2-CNF Boolean formula. To conclude the proof, we make use of the Immerman-Szelepcsényi Theorem, which will later be further discussed. Our proof is simpler, in the sense that it is a direct proof that does not assume previous knowledge of some other NL-complete problem.

In Chap. 5, we explain their similarities and differences between the proofs of all previous four results. We show that all four proofs share some common traits despite the fact that the first one corresponds to an unsatisfiability problem for sentences of FO while the latter three correspond to satisfiability problems of Boolean formulas. We present some other, important Theorems and interesting, relevant topics. We present our general observations derived from this work regarding all the discussed topics as well as some ideas for future work.

1.5 Template

This work has been written in LaTeX by using the open template provided by the University of Maryland, College Park.

Chapter 2: Trakhtenbrot's Theorem

2.1 Overview

The importance of this theorem lies in many aspects.

The theorem proves that Completeness fails over finite models, giving the negative result that fueled novel results over the ways to prove the definability of queries over finite structures. An example of such a way are the Ehrenfeucht -Fraïssé Games, to which topic my work in my Bachelor's Thesis is related.

At the same time, this result was chronologically the first that deployed the idea of expressing a T.M. as an equivalent logic formula and relating the halting of the machine with the satisfiability of the formula. This same idea deployed for different problems offered great results later on, some of which we will explore in the current work.

Even the Theorem that introduced the field of Descriptive Complexity, (Fagin's Theorem in [1]) follows an idea and proof close to that of Trakhtenbrot's Theorem, specified for the class NP.

2.2 Proof of Theorem

Lets now prove Trakhtenbrot's Theorem, following the Proof in [4]. The Theorem states:

Theorem 2.2.1. For every relational vocabulary σ with at least one binary relation symbol, it is undecidable whether a sentence Φ of vocabulary σ is finitely satisfiable.

Proof. Let $M = (Q, \Sigma, \Delta, \delta, q_0, Q_a, Q_r)$ be a deterministic T.M. .

Since we are coding the problem of halting on the empty input, we can assume without loss of generality that $\Delta = 0, 1$, with 0 playing the role of the blank symbol.

We define a vocabulary σ so that its structures represent computations of M. More precisely, $\sigma = \{<, min, T_0(\cdot, \cdot), T_1(\cdot, \cdot), (H_q(\cdot, \cdot))$ for $q \in Q\}$. Relation $<$ is a linear order, while min is a constant symbol for the minimal element with respect to $<$ [1] . T_0 and T_1 are tape predicates; $T_i(p, t)$ indicates that position p at time t writes i, for $i = 0, 1$. H_q's are head predicates; $H_q(p, t)$ expresses that at time t, M is in state q, and the tape's head is in position p.

We will define Φ_M to be the conjunction of specific sentences.

The first sentence will be expressing that $<$ is a linear order and min is its minimal element.

The next sentence will be expressing the initial configuration of M. Thus, it must state that M is in state q_0, the head is in the starting position for the starting time

[1]Therefore, the finite universe can be associated with an initial segment of natural numbers.

moment and the tape contains 0's(blank):

$$H_{q_0}(min, min) \wedge (\forall p T_0(p, min))$$

Another sentence must state that at any time the machine is in exactly one state:

$$\forall t\Big(\exists p\big(\bigvee_{q \in Q} H_q(p,t)\big) \wedge \neg\exists p\big(\bigvee_{q,q' \in Q, q \neq q'} H_q(p,t) \wedge H_{q'}(p,t)\big)\Big)$$

A set of sentences must also express the well-functioning of M. This set depends on the machine and its transition relation δ. We can give an example of such a sentence. Let $\delta(q_a, 1, q_b, 1, r)$, which means that if M is in state q_a and if the current position of the tape head contains 1, then M proceeds to state q_b, writes in the current position 1 and moves the head one position to the right. This will be expressed as the following sentence:

$\forall p \forall t\Big(H_{q_a}(p,t) \wedge T_1(p,t)\Big) \rightarrow$

$\Big(H_{q_b}(p+1, t+1) \wedge T_1(p, t+1) \wedge \forall p' \neq p\big(\bigwedge_{k=0,1} T_k(p',t) \leftrightarrow T_k(p', t+1)\big)\Big)$

Another sentence must be expressing that in every configuration of M, each cell of the tape contains exactly one element of the tape alphabet Δ :

$$\forall p \forall t\Big(T_0(p,t) \leftrightarrow \neg T_1(p,t)\Big)$$

Finally, there must exist a sentence which will imply that M halts:

$$\exists p \exists t \bigvee_{q \in Q_a \bigcup Q_r} H_q(p,t)$$

The conjunction of all the above sentences produces Φ_M. What these sentences together express must be clear by now. Φ_M states that $<, min, T_k$'s, and H_q's are interpreted as indicated above, and that the machine eventually halts. If the machine halts, then $H_q(p,t)$ is true for some p, t, and $q \in Q_a \bigcup Q_r$, and after that the configuration of the machine does not change, so it can be omitted. Due to this, all the configurations of a halting computation can be represented by a finite σ-structure. If Φ_M has a finite model, then (by definition) this model defines a halting, valid computation of M on empty input. Reversely, if M halts on empty input, then the set of all the configurations that M follows in order to halt coded as these relations stated above create a finite model of Φ_M. Therefore, M halts on empty input iff Φ_M has a finite model. However, the Halting problem is undecidable and thus the problem of finite satisfiability for Φ_M is also undecidable. $\square$

Chapter 3: Satisfiability problems Complete for the Classes NP and PSPACE

3.1 Overview

To better understand the importance of the method of expressing Turing Machines with equivalent logical formulas, we will view this through a different angle. In this Chapter, we will present two distinct and very important results, both of which became possible by showing that the Turing Machine for any problem of each class is accepting any input if and only if a respective Boolean formula (or sentence) produced given the specific input is satisfiable. The first such result is the well-known Cook-Levin Theorem which proved that SAT is NP-Complete.

3.2 Cook-Levin Theorem

In this part, we will reproduce the proof for the classical Theorem of Cook and Levin, following the proof presented in [3].

Theorem 3.2.1. The Boolean satisfiability problem (SAT) is Complete for the Complexity Class NP.

Proof. Theorem 3.2.1 specifically states that SAT is NP-Complete, which, as any

Completeness Theorem for a Complexity class, requires two different statements to be proven. First we need to prove that SAT is in NP. Then, we ought to prove that SAT is NP-Hard.

To prove that SAT is in NP is rather straightforward. Let ϕ be a Boolean formula. Then a non-deterministic Turing Machine M_ϕ can non-deterministically produce every assignment of all variables of ϕ in polynomial time regarding the number of variables in ϕ. The formula is satisfiable if at least one such assignment satisfies it.

Now let's prove that any language, say L, in NP can be polynomially reduced to the satisfiability problem of a Boolean formula.

Let M_L be the non-deterministic, polynomial-time T.M. that decides L and let w be an input to M_L. Then, we build a Boolean formula $\phi_{M_L,w}$ that will encode the accepting computation path of M_L for input w.

The construction of $\phi_{M_L,w}$ shall follow the same ideas shown in Theorem 2.2.1, modified to create a Boolean formula, instead of an FO sentence.

First, we present the notion of a *configuration table* (or else *tableau*) for M_L. Since M_L is a non-deterministic, polynomial time T.M. its tableau will be an $\mathcal{O}(\text{poly}(n)) \times \mathcal{O}(\text{poly}(n))$ table, where each row will be a configuration of a computation branch of M_L for input w, as depicted in figure 3.1.

We can assume for practical reasons that no branch of M_L surpasses a time threshold of n^k for some k, so each tableau will be an $n^k \times n^k$ table. So, each i,j cell of a tableau contains the j-th element in the tape for the i-th configuration of this specific computational branch.

Therefore, M_L accepts input w iff there exists an accepting tableau, i.e. a tableau

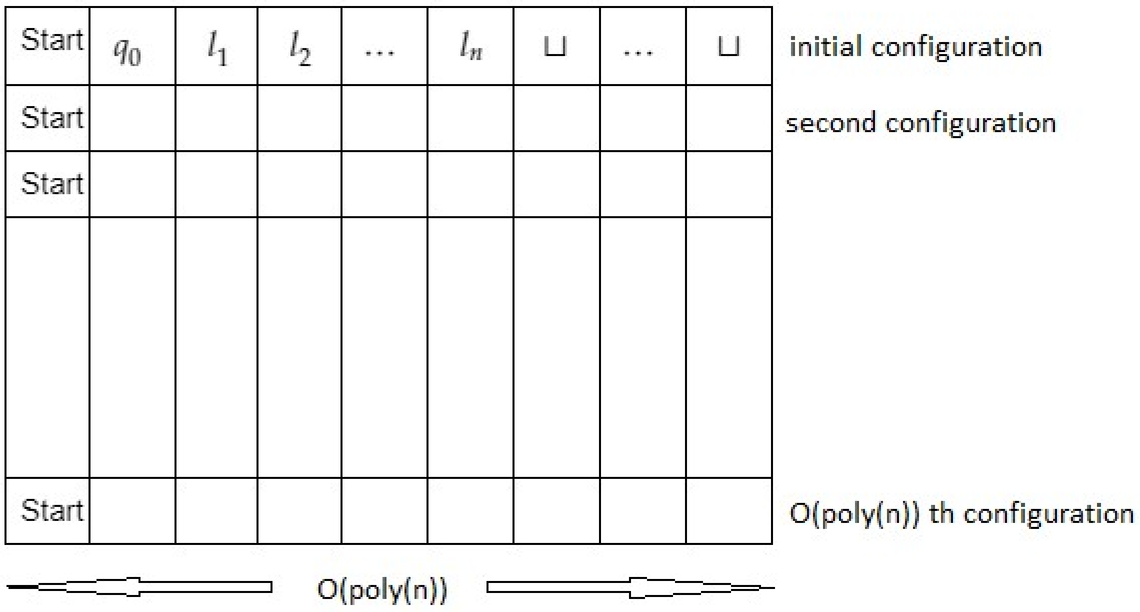


Figure 3.1: The typical representation of a tableau for a computation path of a non-deterministic polynomial-time T.M.

corresponding to an accepting computational branch. For each cell i,j and each symbol s of $A = Q \bigcup \Delta$ (State and Tape Alphabet sets), we shall have an according variable $x_{i,j,s}$, which will be 1 only if the i^{th} configuration of the machine contains symbol s in its j^{th} position.

We will encode such a tableau's properties into $\phi_{M_L,w}$.

$\phi_{M_L,w}$ will be the conjunction of several formulas. First, we will need a formula stating that the initial configuration must be valid:

$$\phi_{start} = x_{1,1,start} \wedge x_{1,2,q_0} \wedge x_{1,3,l_1} \wedge \ldots \wedge x_{1,n+2,l_n} \wedge x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k,\sqcup}$$

Then, we will need a formula that will express that every cell contains exactly one valid symbol:

$$\phi_{cell} = \bigwedge_{i,j \in \{1,2,\ldots,n^k\}} \Big(\big(\bigvee_{s \in A} x_{i,j,s} \big) \wedge \neg \big(\bigvee_{s \neq t \in A} x_{i,j,s} \wedge x_{i,j,t} \big) \Big)$$

We will also need a formula that will express that each row of the tableau follows the previous in a way compatible to the machine's transition function. This can be efficiently checked, by viewing that each 2×3 window follows the rules of the transition function. For the proof, in [3] it is proven as a claim that if the top row of the table is the initial configuration and every window in the table is valid, then each configuration correctly follows the previous. From the machine's transition function, we can produce all instances of valid 2×3 windows as ordered sets of 6 symbols. Let V be the set of such valid windows, expressed as mentioned. Then, this formula will be as follows:

$$\phi_{transit} = \bigvee_{\{c_{1,1},\ldots,c_{2,3}\} \in V} \big(x_{i,j-1,c_{1,1}} \wedge x_{i,j,c_{1,2}} \wedge x_{i,j+1,c_{1,3}} \wedge x_{i+1,j-1,c_{2,1}} \wedge x_{i+1,j,c_{2,2}} \wedge x_{i+1,j+1,c_{2,3}} \big)$$

Finally, we will need a formula that will state that the tableau contains an accepting configuration:

$$\phi_{accept} = \bigvee_{i,j \in \{1,2,\ldots,n^k\}} x_{i,j,q_a}$$

Then, $\phi_{M_L,w}$ will be the conjunction of all the above formulas. We will first show that the reduction is of polynomial complexity. We examine the size of $\phi_{M_L,w}$. First of all, $\phi_{M_L,w}$ contains a number of variables of $\mathcal{O}(n^{2k})$. This holds since the tableau has n^{2k} cells and for each cell we use $|A| = |Q \bigcup \Delta|$ variables, which is a constant regarding M_L.

In addition, each formula we already described is of polynomial size.

Formula ϕ_{start} is created only for the first row and contains some information about each cell of the first row, so is of size $\mathcal{O}(n^k)$.

Formulas ϕ_{cell}, $\phi_{transit}$ and ϕ_{accept} contain information for each cell of the tableau, so they are of length $\mathcal{O}(n^{2k})$.

Therefore, the complete formula $\phi_{M_L,w}$ is of size $\mathcal{O}(n^{2k})$, which is polynomial in n. Because of that, we can produce $\phi_{M_L,w}$ in polynomial time using a repetitive procedure which can be well-defined.

The last part of the proof is to prove that $\phi_{M_L,w}$ is satisfiable if and only if M_L accepts w. This is trivially proven, since $\phi_{M_L,w}$ always expresses a tableau of M_L on input w. Thus, if M_L accepts w, then there exists an accepting computation path and by definition, this path's tableau would give the guidelines as to the satisfying assignment for $\phi_{M_L,w}$.

Reversely, if $\phi_{M_L,w}$ is satisfiable, then the tableau that $\phi_{M_L,w}$ expresses will be a tableau that corresponds to an accepting computation path, so M_L accepts w and the proof is complete. $\square$

3.3 TQBF is PSPACE-Complete

As we have previously stated, the TQBF Problem (True fully Quantified Boolean Formula) is the problem of determining whether a fully quantified Boolean formula (i.e. a Boolean sentence) is true or false.

Theorem 3.3.1. The True fully Quantified Boolean Formula problem (TQBF) is Complete for the Complexity Class PSPACE.

Proof. We will now present the proof that this problem is in fact PSPACE-Complete. We will follow the proof given in [3]. In order to do so, we will construct a proof different but similar to this of Theorem 3.2.1 that we previously proved. We first prove that TQBF is in PSPACE. This we can do by giving an algorithmic procedure that uses polynomial space in order to decide the TQBF problem.

Given as input a sentence ϕ, the procedure does one of the following:

1. If ϕ doesn't has any quantification, then it only contains constants. Therefore, it evaluates its value and accepts if it is true or reject if it is false.

2. If $\phi = \exists x \psi$, then the procedure is recursively repeated for ψ but where previously there was x, now it is substituted once with 0 and once with 1. If at least one of the times we get accepting results, then we accept, else we reject.

3. If $\phi = \forall x \psi$, then the procedure is recursively repeated for ψ but where previously there was x, now it is substituted once with 0 and once with 1. If both of the times we get accepting results, then we accept, else we reject.

The above procedure decides TQBF in polynomial space. This occurs because the recursion depth is at most as much as the number of variables in the initial sentence.

Now we will prove that TQBF is PSPACE-Hard. Let a language $\Lambda \in$ PSPACE., decided by a T.M. M_Λ in space at most n^k for some constant k. We will reduce M_Λ on input w to a Boolean sentence $\phi_{M_\Lambda,w}$, such that $\phi_{M_\Lambda,w}$ is satisfiable iff M_Λ accepts w.

We consider two collections of variables, let c_1 and c_2 such that each corresponds to a configuration of M_Λ on input w and we also consider a number $t > 0$. Then, we construct a Boolean formula $\phi_{c_1,c_2,t}$, such that it is true iff M_Λ can go from configuration c_1 to configuration c_2 in less that t steps.
If we can correctly build such formulas, then we have $\phi_{M_\Lambda,w}$ be the formula $\phi_{c_{start},c_a,T}$, where c_{start} is the initial configuration of the machine, c_a is an accepting configuration and T is a time interval such that M never uses more time than T (e.g. T is $2^{\mathcal{O}(n^k)}$ and we will w.l.o.g. we will assume that T is a power of 2). In order to create the formula, we use the same technique as in the Cook-Levin Theorem's proof for expressing the contents of each cell.

For one time step ($t = 1$), constructing the formula is easy. The formula $\phi_{c_1,c_2,t}$ will be such as to express that either $c_1 = c_2$ or that c_2 immediately follows the configuration c_1.
The first possibility is expressed by creating a Boolean formula stating that each of the variables that represent c_1 have the same value as to the variables that represent c_2.
The second case is expressed by following the technique of $\phi_{transit}$ in the proof of the Cook-Levin Theorem.

For any other time interval $t > 1$, we create our formula by using recursion.

An initial idea is the following:

$$\phi_{c_1,c_2,t} = \exists \kappa_1 \Big(\phi_{c_1,\kappa_1,\frac{t}{2}} \wedge \phi_{\kappa_1,c_2,\frac{t}{2}} \Big)$$

However, this idea cannot work, since the size of the sentence doubles in each recursive step and thus becomes exponentially big. To reduce the resulting size, we use the $\forall$ quantifier as well in order to have only one occurrence of the sentence, instead of two as above. So, we have:

$$\phi_{c_1,c_2,t} = \exists \kappa_1 \forall (c_3, c_4) \in \{(c_1, \kappa_1), (\kappa_1, c_2)\} \Big(\phi_{c_3,c_4,\frac{t}{2}} \Big)$$

Due to repeating the use of the subformula by changing the new variables c_3, c_4, we do not get recursively a doubled formula. The size of the complete formula is $\mathcal{O}(n^{2k})$, since in each recursive step a linear portion of the formula is added and we have $\log(2^{\mathcal{O}(n^k)}) = \mathcal{O}(n^k)$ steps. Hence, the size of the final sentence is polynomial to the size of the input and the proof is complete. $\square$

Although the proofs presented so far share several common traits, these similarities will become clearer as we continue to find them between all the proofs we will present.

Chapter 4: 2-SAT is Complete for the class NL

4.1 Overview

After the work done in order to deeply understand the topics already mentioned in the previous chapters, a question had arisen. Could we use this seemingly common method -or way of thinking about the problem- for proving the Completeness of more variations of satisfiability problems for other Complexity Classes? If so, how would the already existing proofs be useful to such tasks? In order to acquire more insight on the whole topic and at the same time test and better understand the method of reducing specific Machines to formulas, we attempted to come up with a proof about the Completeness of the 2-CNF satisfiability problem for the class NL. The result is all but new, since it is known for at least the last 15 years. However, all the proofs found in the literature (see [5], [6], [3]) uses reductions from already proven NL-Complete problems to 2-SAT. The proof that we came up with follows the "norm" as it reduces the decidability of a language in NL to the satisfiability of a 2-CNF formula by encoding a T.M. to such a formula.

However, notice that our current reduction will have to be of logarithmic space which is by far stricter a constraint than that of a polynomial time (or space) reduction.

Also, an important Theorem that we will use in our proof is the Immerman-Szelepcsényi Theorem, which states that NL is closed under complements (NL=co-NL).

Theorem 4.1.1. For any function $s(n) \geq \log n$ it holds that NSPACE($s(n)$) = co-NSPACE($s(n)$). As a special case, when $s(n) = \log n$, it holds that NL = co-NL.

Therefore if one proves that a problem is co-NL complete, then it is also NL-complete, since the two classes are actually the same one.

4.2 A new Proof of Theorem

Let's now prove that 2-SAT is NL-Complete.

Theorem 4.2.1. The 2-satisfiability problem (2-SAT) is Complete for the Complexity Class NL.

Proof. First of all, we will prove that 2-SAT is in NL. For a given 2-CNF formula f, we will prove that it's unsatisfiability can be proven by a non-deterministic, logarithmic-space T.M. and thus due to Theorem 4.1.1, the satisfiability of such a formula can also be proven by such a machine.

Let f a 2-CNF formula that consists of $|c|$ clauses. Let ν be the set of all positive literals over f. Let's define the following possible computation paths:

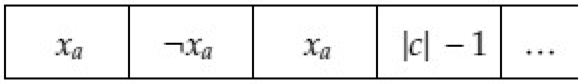
- First, non-deterministically, some $x_a \in \nu$ is chosen and is written in the logarithmic tape as the initial literal.

- In the second step, we choose non-deterministically a clause in which the initial literal participates. We choose the other literal of the clause and we write the initial literal, the negation of the second literal and the step count (now 2, in general is of length $\mathcal{O}(n)$) in the tape.

- Non-deterministically in every step i, we choose a clause in which the previously written in the tape literal is in and we write the initial literal, the negation of the newly chosen literal and the step count in the tape. This is always of length $\mathcal{O}(n)$.

- If in some step $s < |c| + 1$, the tape contains $|x_a|\neg x_a|k|$, then we set the step count to 1 and we repeat the process considering as initial literal the negation of x_a. Now every time in the tape we first write $|x_a|\neg x_a|$ and then the new negation of a literal and the step count.

- If in some step $m < |c|+1$ the tape looks like in Figure 4.1, then the procedure is accepting for this computation path.



x_a	$\neg x_a$	x_a	$\|c\| - 1$	…

Figure 4.1: An example of a state of the writing logarithmic tape for which the procedure will accept. Here $m = |c| - 1$.

Then, f is unsatisfiable if any of these computation paths is an accepting one.

This is because, if we find such a path, we have managed to find a variable x_a which has a circular relation with its negation in f. In exact, if we set x_a to false, then the computation path that we accepted, shows us the path of evaluation of literals through which we will end up to have $\neg x_a$ to be also set to false. This is the first part of our procedure. Therefore, x_a cannot be set false and must be set true for f to be satisfiable. In this case however, $\neg x_a$ is set to false. So, we follow the path of evaluation of literals indicated by the second part of our procedure and we end up to have x_a set to false, which is a contradiction. So, for any evaluation of x_a f is unsatisfiable.
Reversely, due to the nature of this property, any unsatisfiable, 2-CNF formula ought to have such a circular relation between a literal and its negation, otherwise, it would be easily satisfiable by setting each literal to an appropriate value in a linear fashion.
Thus, 2-SAT $\in$ co-NL and so, 2-SAT $\in$ NL.

Now we will prove that 2-SAT is co-NL-Hard.

Let B be a language in NL. Let B be decided by a non-deterministic T.M. of logarithmic space, say M_B.

Let's assume w.l.o.g. that M_B has a single accepting state, say q_α (if it has more than one we can easily adjust the machine to have exactly one).

We will transform the accepting run of M_B given input w to an equivalent instance of a 2-CNF sentence.

First of all, given input w, M_B will have a specific configuration table for each computation path, consisting of all possible configurations of the machine, while working on that path. Every configuration will look like in Figure 4.2.

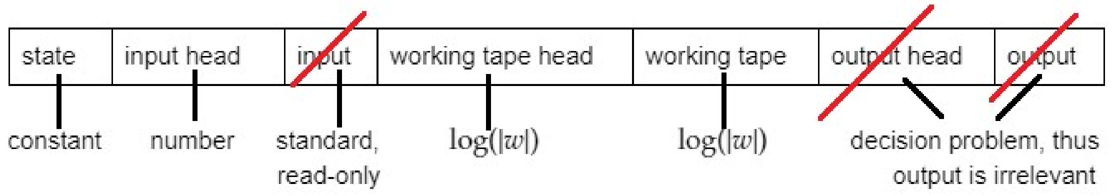


Figure 4.2: Any configuration of the machine will have the above form.

The first entry can be one of the machine's states, therefore might be $|Q|$ different states. The second entry denotes where the input tape's head lies and since this tape contains w, then it can be in $(n+1)$ different positions. The working tape is by definition of length $\log(n)$ and contains symbols of the alphabet $\mathcal{A}$. Due to that, its head is in a position from 1 to $\mathcal{O}(\log(n))$. So, the set of all possible configurations is of size:

$$|Q| \times (n+1) \times |\mathcal{A}^{\log(n)}| \times \mathcal{O}(\log(n)) \leq \mathcal{O}(n^k)$$

A successful computation of M_B given input w, would begin with an initial configuration $C_0 := (s, 0, \triangleright, \epsilon)$, where s is the starting state of the machine. It would accept if ending in a configuration C^i_{accept} of the form $(q_{accept}, \alpha, \beta, i)$ but instead of keeping track of all accepting configurations, we can think of a virtual configuration

C_{accept}, which is a configuration immediately after every accepting configuration.

In order for the computation of w by M_B to be successful, we should have a specific set of configurations, each immediately preceding the previous one in a total order, which begin from C_0 and end with C_{accept}. In order for two configurations to be consecutive (C_a is followed by C_b), the second must arrive from the first one via following the rules of M_B, $(C_a \rightarrow_{(M_B,w)} C_b)$, or in the special case of C_{accept}, where we assume that every accepting configuration is followed by C_{accept}.

We will model these requirements with a 2-CNF formula, which will be *unsatisfiable if and only if M_B accepts w.*

Our formula, let $f_{M_B,w}$ will contain clauses of the following forms:

- First, a clause (X_0), where X_0 is a Boolean variable corresponding to the initial configuration C_0.

- For each configuration C_a and for every configuration C_b successive to C_a we will have a clause $(\neg X_a \vee X_b)$.

- For each accepting configuration C^i_{accept} we will have a clause $(\neg X^i_{accept} \vee X_{accept})$.

- Finally, we will add a clause $(\neg X_{accept})$.

Before we prove the equivalence of the initial problem to the unsatisfiability of $f_{M_B,w}$, let's prove that our sentence can be produced in logarithmic space. In other words, we will prove that there exists a deterministic, logarithmic-space T.M. that, given as inputs the encoding of M_B and w, it produces $f_{M_B,w}$.

As we have already shown, the number of all possible configurations is polynomial in the size of the input. This means that the number of pairs of configurations is also polynomial in the size of the input. Thus, our T.M. can work as follows:
It will linearly produce each valid pair of consecutive configurations by producing all valid words of size equal to the size of two configurations (which is logarithmic to the size of the input, since each configuration is of logarithmic size) and checking for each word if it actually ensembles two distinct and consecutive instances of M_B while working on input w. This can be achieved in polynomial time and logarithmic space for each pair (by checking the rules of M_B and whether the first configuration can lead to the second) and thus -by reusing space- can be achieved in polynomial time and logarithmic space for all pairs of configurations. Each configuration can be attributed a specific variable and therefore, for each valid pair of configurations C_a, C_b we produce a clause $(\neg X_a \vee X_b)$.

With a similar procedure we find all accepting configurations and produce the clauses $(\neg X^i_{accept} \vee X_{accept})$. In the end we add the two distinct clauses (X_0), $(\neg X_{accept})$ and our formula is ready.

Now let's prove the equivalence between 2-SAT and B.

Let w be accepted by M_B. This means that there exists a sequence of configurations of M_B with given input w, that begins from C_0, followed by $C_1 \ldots$ etc $\ldots$ followed by C_n, followed by C^i_{accept}, followed by C_{accept}. Based on the way $f_{M_B,w}$ was produced, our formula is going to contain a subformula of the form:

$$(X_0) \wedge (\neg X_0 \vee X_1) \wedge (\neg X_1 \vee \ldots) \wedge \ldots \wedge (\neg X_n \vee X^i_{accept}) \wedge (\neg X^i_{accept} \vee X_{accept}) \wedge (\neg X_{accept})$$

It is straightforward to verify that this part of $f_{M_B,w}$ is always unsatisfiable and thus, $f_{M_B,w}$ is unsatisfiable in this case.

So, if w is accepted by M_B, then $f_{M_B,w}$ is unsatisfiable.

Reversely, let w not be accepted by M_B. This means that no accepting configurations exist in the computation of M_B with input w. Respectively, no clauses of the form $(\neg X^i_{accept} \vee X_{accept})$ are part of $f_{M_B,w}$. Due to that, we can assign satisfying values to the variables of $f_{M_B,w}$ as follows:

- Set $X_0 = 1$ and $X_{accept} = 0$.

- For each clause, where the literal $\neg X_0$ participates in, set the other literal (by definition of the formula, positive) to 1.

- Repeat for each such variable the same procedure, until no clause is left unsatisfied.

  (This can be systematically done by following each computation path of M_B

with input w and setting the respective variables to 1)

This evaluation satisfies the formula, which means that if w is not accepted by M_B, then $f_{M_B,w}$ is satisfiable and the proof is complete.

Notice that, as we previously stated, we have proven that $\overline{\text{2-SAT}}$ is NL-Complete. However, due to the Immerman-Szelepcs$\acute{e}$nyi Theorem, we know that NL is closed under complements. We established that 2-SAT is in NL and therefore, $\overline{\text{2-SAT}}$ is NL-Complete means that 2-SAT is NL-Complete. $\square$

Now we are ready to continue by comparing the several proofs we already presented.

Chapter 5: Observations, Relevant Results and Further Work

5.1 Comparison of the several proofs

This is one of the most interesting and difficult parts of this work. We will attempt to present several similarities and differences between the proof techniques and make some interesting observations in a way as clear as possible.

5.1.1 General observations regarding the proofs

First of all, let us present a interesting detail in Trakhtenbrot's Theorem. If a sentence ϕ of FO does not have a finite model (is not finitely satisfiable), then one of two things occurs.
Either ϕ does not have a finite model (e.g. $\phi = (v \wedge \neg v)$), or ϕ has an infinite model. Notice that, the sentence we created in the proof is of the first case. This occurs due to the addition of the halting clause:

$$\exists p \exists t \bigvee_{q \in Q_a \bigcup Q_r} H_q(p,t)$$

If we did not include this clause, then our sentence would be of the second case. This is a rather interesting observation because it seems like this small detail somewhat captures a logical difference between decidability and recognizability for a T.M. Specifically, It might be useful to research this idea further in future work.

Next, let's focus at the comparison between the proofs.

5.1.2 Comparing the proofs of theorems 2.2.1 and 3.2.1

Another useful observation lies between the proofs of Trakhtenbrot's and Cook-Levin's Theorems. As we can observe by just reading the two proofs, the one for Theorem 3.2.1 follows the same steps as that of Theorem 2.2.1. However, since it has to create a Boolean formula and not an FO sentence, the proof cannot use the more expressive tools of FO, such as relations and variables that take values over a set. To overcome that problem in a comely manner in this proof, instead of leaving the position on the tape and the time-step to be considered variables for FO relations, the tableau is created and appropriate Boolean variables are defined for each tableau position and each possible tape symbol. For example, a relation instance $T_0(p,t)$ from Theorem 2.2.1, would be encoded as a variable $c_{p,t,0}$ in Cook-Levin's proof. Similarly, we would encode other relations used in Theorem 2.2.1 (e.g. $H_{q_i}(p,t)$ will be encoded using a respective variable c_{p,t,q_i}).

Therefore, it seems like Theorem 3.2.1 uses exactly the same idea as Theorem 2.2.1 and simply manages to express the setting of an accepting path of configurations using only propositional logic and polynomially many Boolean variables.

5.1.3 Comparing the proofs of theorems 3.2.1 and 4.2.1

Let's present a comparison between the proofs regarding SAT and 2-SAT. At first, it seems rather counter-intuitive that both SAT and 2-SAT can express the encoding of T.M.s in respect to the classes for which each problem is Complete. Since, in order to reduce the problem of acceptance from a non-deterministic polynomial-time T.M. in a Boolean formula without specific structure we need a polynomial-time reduction, it seems like it ought to be impossible to do the same but with a logarithmic-space reduction. What this observation fails to include to its line of reasoning though, is that the T.M.s we are in each case required to encode, differ. In fact, as a non-deterministic T.M. deciding a Language in NP, requires polynomial time to compute any computation branch, similarly, a non-deterministic T.M. deciding a Language in NL, requires logarithmic space.

So, while the proof of Cook-Levin manages to encode an accepting path of configurations with a detailed logical representation of the various configurations involved, our proof uses a more high-level approach, where each configuration in fact corresponds to a Boolean variable and the only relations our formula encodes are those of a configuration being the initial one, of a configuration being an accepting one and of a pair of configurations consisting of configurations where the first is immediately followed by the second one by complying to the rules of the transition relation of the machine.

Even though this approach is less detailed, it suffices for the class of problems it refers to.

5.1.4 A fitting cross-observation

In Theorem 2.2.1, we have any T.M. which, depending on whether it halts on empty input or not, can be reduced to an FO sentence, which is finitely satisfiable or not. If we tried to apply the idea of using a tableau in Theorem 2.2.1, in several cases, the tableau would be infinite, so the produced sentence, would not have a finite model.

Similarly, the idea of encoding the acceptance of a word from a T.M. with the satisfiability of a Boolean formula, if always implemented in the same way, would lead to wrong results. For example, if we use the same proof from SAT for 2-SAT, the resulting formula could have never be produced from a logarithmic-space reduction. Or, if we used the exact same idea of SAT for TQBF, we would end up with a formula of exponential size.

This means that even though the same idea lies behind every one of the proofs, in order to be implemented correctly, one has to get a good understanding of the actual complexity bound of the T.M. participating in the proof. This is actually a very intriguing notion, since it means that just by following and understanding these proofs, we enhance our intuition regarding the respective complexity classes.

5.2 Relevant Results

We will now present briefly some other, relevant and important results.

First of all, Fagin in [1] proved that the Existential Second Order logic($\exists$SO) captures NP. We say that a logic L captures a complexity class C if:

1. The data complexity of L is C. This means that for every L-sentence Φ, testing for any finite structure $\mathcal{U}$ if it models Φ is a problem in C.

2. For every property $\mathcal{P}$ of finite structures that can be tested with complexity C, there exists an L-sentence $\Phi_{\mathcal{P}}$ such that for any finite structure $\mathcal{U}$, it is true that $\mathcal{U}$ models $\Phi_{\mathcal{P}}$ if and only if $\mathcal{U}$ has the property $\mathcal{P}$.

Fagin's Theorem basically explains that exactly every property that requires a non-deterministic, polynomial-time T.M. in order to be computed, can also be expressed as a sentence of $\exists$SO. This result, has set the foundations for Descriptive Complexity, a field that studies Complexity classes via the expressive power of logic required in order to express properties that belong in each class.

The other two results that we will mention are both proven by use of the Reachability method, which is in general closely related to the classes of non-deterministic space. For more regarding this method and for studying the proofs of those two results, one can read Chapter 7 of [5].

The first result is Savitch's Theorem, which states that for any proper complexity function $f(n) \geq \log(n)$, NSPACE$(f(n)) \subseteq$ SPACE$(f^2(n))$.

This result directly implies that PSPACE= NPSPACE. It also makes it clear that non-determinism in respect of space resources is less powerful than it is in respect of time.

Another theorem that supports this idea, is the Immerman-Szelepcsényi Theorem, which states that for any proper complexity function $f(n) \geq \log(n)$, NSPACE$(f(n)) =$ coNSPACE$(f(n))$.

Again, an immediate result from the Immerman-Szelepcsényi Theorem, is that NL= coNL, a result that we used for our proof in Chapter 4.2.

Of course, there exist many more exciting results regarding those topics but we will not refer to any more, since we haven't studied it as extensively.

5.3 Future Work

Many interesting ideas have come while working on the parts of this Thesis. An important next step would be to try and better understand whether this "General SAT Idea" we presented can actually become a more exact approach, or even an algorithmic one for finding which Boolean satisfiability problem is complete for any complexity class. In order to do so, it would be useful to gather more insight in those methods, perhaps by re-proving some other similar results. A very appealing such case, would be to give a new proof that Horn-Sat is P-complete using the ideas presented in this work. Moreover, a lot of exciting work lies in the fields of Descriptive Complexity and Finite Model Theory. It is important to strongly understand the proof of Fagin's Theorem and compare it with the proofs we already saw. Due to the previous work done for my Bachelor's Thesis, it also seems only right to try and question which the relationship is between several games of logic and important complexity classes.

Bibliography

[1] Ronald Fagin. *Contributions to the model theory of finite structures.* University of California, Berkeley, 1973.

[2] Herbert B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 2001.

[3] Michael Sipser et al. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

[4] Leonid Libkin. *Elements of finite model theory.* Springer Science & Business Media, 2013.

[5] Christos H Papadimitriou. *Computational complexity.* John Wiley and Sons Ltd., 2003.

[6] Neil Immerman. *Descriptive complexity.* Springer Science & Business Media, 2012.