SAPIENZA
UNIVERSITÀ DI ROMA

IDK something about proofs, search problems, circuits, protocols and P $\neq$ NP?

Faculty of Information Engineering, Computer Science and Statistics
Bachelor's Degree in Computer Science

Simone Bianco
ID number 1986936

Advisor
Prof. Nicola Galesi
Prof. Massimo Lauria

Academic Year 2023/2024

Thesis not yet defended

IDK something about proofs, search problems, circuits, protocols and P $\neq$ NP?
Bachelor's Thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: bianco.simone@outlook.it

Please help me.

Abstract

Uhhh, idk

Contents

Chapter 1

Introduction

For many years, the study of **decision problems** has been the main focus of computability theory. These types of problems include any problem that can be described as a simple question with a yes-or-no answer, such as asking if some input object has got some kind of property or not. Decidability theory plays a core rule in math and computer science due to the subjects studied by both fields. However, even thought decision problem can be computed by an algorithm producing a solution for a given input, not all decidable problems can be solved "efficiently", meaning in a reasonable amount of time.

This very nature of decision problems has given birth to **complexity theory**, the field of computer science focused on solving the following fundamental question commonly referred to as the P vs. NP problem: «does every decision problem with an efficient way to *verify* a solution for an input also have an efficient way to *solve* the problem for that input?». The hardness of this question sparked into the establishment of many subsets of complexity theory, in particular proof complexity, communication complexity and circuit complexity. Each of these subfields revolve around solving this question for one single NP-Complete problem, that being problems for which finding an efficient algorithm would automatically imply that $\mathsf{P} = \mathsf{NP}$.

In recent years, the study of decision problems has been generalized to the study of **functional problems**, i.e. any problem where an output that is more complex than a yes-or-no answer is expected for a given input. By their very nature, functional problems are an "harder type" of problems respect to decision problems, describing any possible type of computation achievable through the concept of mathematical function and algorithm. In the same fashion of decision problems, the study of functional problems focuses on the FP vs. FNP question. In particular, solving this question for the functional case would also imply solving it for the decisional case and vice versa.

The study of functional problems has given many important results through its characterization both as a **black-box model** and a **white-box model**. These two models have been shown to be highly correlated to the previous subfields of decision problems, in particular proof complexity. These correlations give a way to study the latter through the lens of functional problems, which enables "less restrictive" methods of study. Moreover, it has been shown that so-called *natural proof systems* effectively correspond to black-box total functional problems.

Recent studies discuss if the already known relations between white-box functional problems, communication complexity and circuit complexity can be extended in order to establish a strong characterization as the one found for proof complexity. If this characterization is solid, the study of total functional problems would play an essential role in complexity theory, becoming an **unifying theory** between its subfields and thus implying that any result found in any subfield would also have impact in the others.

However, such results have not yet been found, even though a lot of process has been made. In particular, the concepts of **feasible interpolation** and **query-to-communication lifting theorems** play a very important role in finding such relations, which however are still not enough to ensure a solid "organizing theory". Nonetheless, such unifying characterizations are well-defined for highly studied proof systems and circuit models, such as Treelike Resolution and Monotone Formula Circuits.

The objective of this work is to study the relations found in proof, communication and circuit complexity, formalizing the concepts involved in each field and giving an in-depth view on how they are linked to the study of total functional problems.

Chapter 2

Preliminaries

2.1 Search Problems

As briefly discussed in the introduction, the concept of **functional problems** rose as a generalization of the concept of decision problems. Mathematically, each decision problem can be defined as a subset of possible inputs: an input object is part of the set if and only if it has the required property. For example, the decision problem «is this number prime?» can be described as the set PRIMES $= \{n \in \mathbb{N} \mid n \text{ is prime}\}$.

In a very similar way, functional problems can be described mathematically described through the concept of relation: given the set of inputs X and the set of outputs Y of the functional problem, the latter can be described as the relation $R \subseteq X \times Y$. An important thing to notice is that even though the name implies a correlation to mathematical functions, the given definition also includes partial and multi-valued functions, that being functions for which not all inputs have a corresponding output and functions for which one input can have more outputs.

For these reasons, the term *functional problem* is considered to be slightly abused. In recent years, this issue was solved by the introduction of the term **search problems**, which describes problems based on finding an output for the given input. It's easy to see that this informal definition of search problems better suits the previous definition given for functional problems. While decision problems focus on answering the question «does this object have the following property?», search problems focus on answering the question «what gives this object the following property?». Mathematically, this can be described as the difference between «is there an output?» and «what is the output?».For example, the search problem equivalent of the previous prime number problem would answer the question «what is the prime factorization of this number?».

We give a more detailed definition of each search problems through the use of a *sequence of relations* rather than a single relation in order to allow inputs of different length and different types of outputs based on the length of the inputs. This definition is a conjunction of the various definitions given in [LNN+95; BCE+98; RGR22; BFI23].

Definition 1. *A search problem is a sequence $R = (R_n)_{n\in\mathbb{N}}$ of relations $R_n \subseteq \{0,1\}^n \times O_n$, one for each $n \in \mathbb{N}$, where each O_n is a finite set called "outcome set".*

A search problem is said to be total if for each R_n in the sequence it holds that $\forall x \in \{0,1\}^n$ there is an $y \in O_n$ such that $(x, y) \in R_n$.

In particular, this work will focus on **total search problems**. Notice that this definition removes partial function from the context, while multi-valued functions are still allowed. Moreover, complexity theory focuses on the study of **decidable** decision and search problems, that being problems for which we know that there is always an computable answer.

2.2 The complexity classes FP, FNP and TFNP

In complexity theory, decidable decision problems are grouped in two major classes: problems that can be **solved** in a reasonable amount of time and problems that can be **verified** in a reasonable amount of time. These classes are respectively referred to as P, the class of problems solvable by a deterministic algorithm in polynomial time, and NP, the class of problems for which a solution can be verified by a a deterministic algorithm in polynomial time (or equivalently, the class of problems solvable by a non-deterministic algorithm). By the very definition of these classes, it's easy to see that P $\subseteq$ NP.

Similarly, in the context of search problems we define the classes FP and FNP, standing for *functional P* and *functional NP*. An important remark to be made is that, since they are defined on two different types of problems, it doesn't hold that P $\subseteq$ FP or that NP $\subseteq$ FNP. However, a decision problem can indeed be seen as a search problem with only two different outputs: *true* or *false*. In fact, as discussed in [BG94; BCE+98; DK14], an important result is that P = NP if and only if FP = FNP, meaning that even though search problems are generally harder than decision problems, even if the latter are efficiently solvable then the other also is.

Moreover, by the very own nature of search problems, it's important to distinct between *total* and *partial* search problems. Thus, for search problems we also define the class TFNP, standing for *total functional NP*.

Definition 2. *A search problem R is in FP if it is solvable by an algorithm in polynomial time. Likewise, as search problem R is in FNP if its solutions are verifiable by an algorithm in polynomial time. If the search problem is also total, it is in TFNP.*

Additionally, we will assume that each problem in FP is also total: since problems in FP are *solvable* in polynomial time, when a solution doesn't exist for a given input we can consider the solution to be a pre-chosen «doesn't exist» value, making the problem total. Obviously, this assumption implies that FP $\subseteq$ TFNP $\subseteq$ FNP.

Differently from decision problems, search problems are conjectured to not have any "normal" complete problem, that being problems for which an efficient solution would imply an efficient solution for any other search problem. The reason behind this conjecture is the nature itself of total search problems: in decision problems, any "yes" (or "no") answer can be reduced to a "yes" (or "no) answer of a complete decision problem, but total search problems imply that every input has a "yes" instance as output, meaning that there is no way for an instance to be reduced to another "no" instance.

For these reasons, for search problems we define the concept of completeness with respect to a **specific search problem**: given a search problem R, we define the subclass of all search problems reducible to R. Thus, the problem R is said to be "complete" for its own class of reducible problems. Unexpectedly, a lot of TFNP problems can be characterized with very few cases of such subclasses, giving birth to an actual hierarchy of search problems.

$$\begin{tikzcd}
 & & \mathsf{FNP} & & \\
 & & \mathsf{TFNP} \arrow[u] & & \\
\mathsf{PPP} \arrow[urr] & & \mathsf{PPA} \arrow[u] & & \mathsf{PLS} \arrow[ull] \\
 & \mathsf{PPAD} \arrow[ul] \arrow[ur] & & & \\
 & & \mathsf{CLS} \arrow[ul] \arrow[uurr] & & \\
 & & \mathsf{FP} \arrow[u] & &
\end{tikzcd}$$

Figure 2.1. Hierarchy of the most commonly defined complete search problem classes. An arrow from class A to class B means that $A \subseteq B$.

Each of the subclasses shown in the previous hierarchy is characterized by a **complete search problem**:

- PPP (Polynomial Pigeonhole Principle): the class of problems whose solution is guaranteed by the Pigeonhole principle. It is defined as the class of problems that are polynomial-time reducible to the *Pigeon problem*

- PPA (Polynomial Parity Argument): is the class of problems whose solution is guaranteed by the handshaking lemma. It is defined as the class of problems that are polynomial-time reducible to the *Odd-degree Vertices problem.*

- PLS (Polynomial Local Search): the class of search problems designed to model the process of finding a local optimum of a function. It is defined as the class of problems that are polynomial-time reducible to the *Sink-of-DAG problem.*

- PPAD (Polynomial Parity Argument): is the class of problems whose solution is guaranteed by the directed version of the handshaking lemma. It is defined as the class of problems that are polynomial-time reducible to the *End-of-Line problem.*

- CLS (Continuous Local Search): the class of search problems designed to model the process of finding a local optimum of a continuous function over a continuous domain. It is defined as the class of problems that are polynomial-time reducible to the *Continuous Localpoint problem*.

Proving any separation between one of these inclusions would directly conclude that FP $\neq$ FNP, giving an answer to the conjecture.

2.3 Proof Complexity

2.4 Communication Complexity

2.5 Circuit Complexity

Chapter 3

Black-box TFNP

3.1 The black-box model

3.2 Proof System Characterization

3.3 Natural Proof Systems

3.3.1 The Reflection Principle

3.3.2 Verification Procedures

3.4 The TFNP^{dt} hierarchy

3.5 An in-depth analysis: FP^{dt} = TreeRes

Chapter 4

White-box TFNP

4.1 The white-box model

4.2 Karchmer-Widgerson Games

4.2.1 Unsatisfiability Certificate

4.3 Circuit Characterization

4.4 The TFNP^{cc} hierarchy

4.5 An in-depth analysis: FP^{cc} = MonotoneFormulas

Chapter 5

Notes

The following definitions and proofs are a reformulation of the results shown in [BFI23; GHJ+22; RGR22]

Definition 3. *A total (query) search problem R is a sequence of relations $R = \{R_n : R_n \subseteq \{0,1\}^n \times O_n\}$, one for each $n \in \mathbb{N}$, where each O_n is a finite set called outcome set, such that $\forall x \in \{0,1\}^n$ there is an $o \in O_n$ for which $(x, o) \in R_n$.*

A total search problem R is in TFNP^{dt} if its solutions are verifiable through decision trees: for each $i \in O$ there is a decision tree $T_j : \{0,1\}^n \to \{0,1\}$ with $\text{poly}(\log \text{n})$*-depth such that $T_j(x) = 1 \iff (x, j) \in R_n$.*

While total search problems are formally defined as sequences $R = \{R_1, \ldots, R_n\}$, it will often make sense to speak of an individual search problem R_i in the sequence. Therefore, we will slightly abuse the notation and also call R_i a total search problem.

Definition 4. *Given $R \subseteq \{0,1\}^n \times O$ and $S \subseteq \{0,1\}^m \times O'$, a decision tree reduction from R to S is defined by two sets of decision trees $\{T_i \mid T_i : \{0,1\}^n \to \{0,1\}\}$ and $\{T_j^o \mid T_j^o : \{0,1\}^n \to O\}$, for each $i \in [m]$ and each $j \in O'$, such that*

$$\forall x \in \{0,1\}^n \ \ ((T_1(x), \ldots, T_m(x)), j) \in S \implies (x, T_o'(x)) \in R$$

<u>Note</u>: the o in the decision trees T_j^o is purely a notation.

To give an easier intuition of a decision tree reduction, the decision trees T_i map inputs from Q to R, while the decision trees T_o' map solutions to R back into solutions of Q. The *size* s of the reduction is the number of input bits to S, meaning that $s = m$, while the *depth* d of the reduction is the maximum depth of any tree involved in the reduction

$$d := \max(\{D(T_i) : i \in [m]\} \cup \{D(T_o') : o \in O_m\})$$

Finally, we define the *complexity* of the reduction as $\log s + d$. Moreover, we denote as $R^{dt}(S)$ the minimum complexity of any decision tree reduction from R to S. Using this definition, we can define complexity classes of total query search problems via decision tree reductions. Given a total query search problem $S = (S_n)$, we define the subclass of problems reducible to S as:

$$S^{dt} := \{R : S^{dt}(R) = \text{poly}(\log n)\}$$

where $R = (R_n)$.

Any total query search problem with solution verifiers T_o for each $o \in O$ can be *encoded* into a canonical unsatisfiable CNF formula.

Proposition 1. *Given a total query search problem $R \subseteq \{0,1\}^n \times O$, there exists an unsatisfiable CNF formula F defined on $|O|$-many variables such that $R = \mathsf{Search}(F)$. This formula is called the canonical CNF encoding of R.*

Proof. Since $R \in \mathsf{TFNP}^{dt}$, for each $o \in O$ there is a poly$(\log n)$-depth decision tree T_o that verifies R. Then, for each T_o, let C_o be the clause obtained by taking the disjunction over the conjunction of the literals along each of the accepting paths in T_o, meaning that C_o is a DNF and, by De Morgan's theorem, that $\overline{C_o}$ is a CNF.

Let $F := \bigwedge_{o \in O} \overline{C_o}$. Since each R is a total search problem, for each input there is a valid output, implying that at least one tree T_o will have an accepting path. Hence, by definition of C_o, we get that $\overline{C_o} = 0$, implying that there will always be a false clause in F and thus that F is an unsatisfiable CNF.

concluding that:

$$(x, o) \in R \iff T_o(x) = 1 \iff \overline{C_o} = 0 \iff (x, o) \in \mathsf{Search}(F)$$

□

This result implies that black-box TFNP is *exactly* the study of the false clause search problem. Then, instead of studying a total search problem R, it's sufficient to study the search problem $\mathsf{Search}(F)$ associated with the canonical CNF encoding F of R.

Given a proof system P and an unsatisfiable CNF formula F, we define the *complexity required by P to prove F*, denoted with $P(F)$, as:

$$P(F) := \min_{\Pi \ P\text{-proof of } F} (\deg(\Pi) + \log \operatorname{size}(\Pi))$$

where deg is the *degree* of the proof, which is a measure defined by the proof system itself. For example, for the Resolution proof system the degree is defined as the *width* of the proof, which is the maximum number of literals in any clause in Π.

Definition 5. *Given $R \in \mathsf{TFNP}^{dt}$, we say that R characterizes and a proof system P (and that P characterizes R) if it holds that $R^{dt} = \{\mathsf{Search}(F) : P(F) = \text{poly}(\log n)\}$.*

Many of such characterizations hold in the following stronger sense. In particular, for any of the common "natural" proof systems P, if R is the problem that characterizes P then for any unsatisfiable CNF F it holds that $P(F) = \theta(R^{dt}(\mathsf{Search}(F)))$.

Definition 6. *Given a proof system P, the reflection principle* Ref_P *states that P-proofs are sound: if Π is a P-proof of a CNF formula H then H must be unsatisfiable for any assignment α. Formally, we sat that:*

$$\mathrm{Proof}_P(H, \Pi) \implies \mathrm{Unsat}(H, \alpha)$$

Tree-like resolutions for an unsatisfiable CNF formula are strictly connected to the decision trees that solve its associated search problem. In particular, it can be proven that the smallest tree-like refutation has the exact same structure of the smallest decision tree.

Lemma 1. *[BGL13] Let F be an unsatisfiable* CNF *formula. If there is a tree-like refutation of F with structure T, there also exists a decision tree with structure T that solves* Search(F)

Proof. We procede by induction on the size s of the refutation of F.

Let $F = C_1 \wedge \ldots \wedge C_m$. If $s = 1$, then the refutation is made up of only one step that ends with the empty clause, implying that $\exists i \in [m]$ such that $F = C_i = \bot$. Hence, Search(F) can be solved by the decision tree made of only one vertex labeled with i.

We now assume that every formula with a tree-like refutation with a structure of size s there exists a decision tree with the same structure that solves the search problem associated with the formula.

Suppose now that the size s of the refutation is bigger than 1. Let x be the last variable resolved by the refutation and let T_0 and T_1 be the subtrees of T such that x is the root of T_0 and $\overline{x}$ is the root of T_1.

Consider now the formulas $F\restriction_{x=0}$ and $F\restriction_{x=1}$, respectively corresponding the formula F with the value 0 or 1 assigned to x. It's easy to see that the subtrees T_0 and T_1 are valid refutations of the formulas $F\restriction_{x=0}$ and $F\restriction_{x=1}$: if $b = 0$, then x evaluates to 0, otherwise if $b = 1$ then $\overline{x}$ evaluates to 0.

Since T_0 and T_1 have size $s - 1$, by inductive hypothesis there exist two decision tree with structure T_0 and T_1 that solve Search$(F\restriction_{x=0})$ and Search$(F\restriction_{x=1})$.

Finally, the search problem Search(F) can be solved by the decision tree that queries x and proceeds with the decision tree T_b based on the value $b \in \{0, 1\}$ such that $x = b$.

□

Definition 7. *Given two rooted trees T and T', we say that T is embeddable in T' if there exists a mapping $f : V(T) \to V(T')$ such that, for any vertices $u, v \in V(T)$, if u is a parent of v in T then $f(u)$ is an ancestor of $f(v)$ in T'.*

Lemma 2. *[BGL13; LNN+95] Let F be an unsatisfiable* CNF *formula. If there is a decision tree with structure T that solves* Search(F)*, there also exists a tree-like refutation of F with structure T' such that T' is embeddable in T.*

Proof. The main idea is to associate inductively, starting from the leaves, a clause to each vertex of T in order to transform T in a tree-like refutation of F. In particular, each vertex v gets associated to a clause $C(v)$ such that every input of the decision tree that reaches v falsifies $C(v)$.

Let $F = C_1 \wedge \ldots \wedge C_m$. For all $i \in [m]$, we associate the clause C_i to the leaf of T labeled with i. This constitutes our base case.

Consider now a vertex v that isn't a leaf. Let x be the variable that labels v and let u_0, u_1 be the vertices such that the edge (v, u_0) is taken if $x = 0$ and the edge (v, u_1) is taken if $x = 1$. By induction, assume that u_0 and u_1 have already been associated with the clauses C_0 and C_1.

By way of contradiction, suppose that C_0 contains the literal $\overline{x}$. Then, since in a decision tree each variable can be queried only once in every path, there will always be an input with $x = 0$ that reaches v. Since $x = 0$ and since C_0 contains $\overline{x}$, this input would satisfy C_0, contradicting the fact that C_0 was associated to u_0 in a way that it is falsified by every input.

Thus, the only possibility is that C_0 can't contain the literal $\overline{x}$. Similarly, we can show that C_1 can't contain the literal x. This leaves us with only two possibilities: either $C_0 = x \vee \alpha$ and $C_1 = \overline{x} \vee \beta$ or one of C_0, C_1 doesn't contain $x, \overline{x}$.

In the first case, we can simply associate to v the clause $C = \alpha \vee \beta$. In the second case, we associate to v the clause that doesn't contain $x, \overline{x}$ (chose any of them if both clauses do not contain $x, \overline{x}$).

In particular, we notice that the first case directly emulates the resolution rule, while the second case essentially represent "redundant steps". By "skipping" these redundant steps, we can obtain a tree T' that is embeddable in T and that contains only nodes on which the first case was applied. Finally, it's easy to deduce that the root node of T' will always be associated with the empty clause $\perp$, concluding that T' is the structure of a tree-like refutation of F.

□

Theorem 1. *Let F be an unsatisfiable* CNF *formula. The smallest tree-like refutation of F has size s and depth d if and only if the smallest decision tree solving* $\mathsf{Search}(F)$ *has size s and depth d.*

Proof. Let s and d be the size and depth of the smallest tree-like refutation of F. Likewise, let x and y be the size and depth of the smallest decision tree solving $\mathsf{Search}(F)$.

Then, by Lemma 1, we know that there exists a decision tree that solved $\mathsf{Search}(F)$ with the same structure of the smallest refutation. Let α and β be the size and depth of this decision tree. It's easy to see that $s = \alpha \geq x$ and $d = \beta \geq y$.

Viceversa, by Lemma 2, we know that there exists a tree-like refutation of F such that its structure is embeddable in the one of the smallest decision tree. Let γ and β be the size and depth of this tree-like refutation. Since the latter is embedded in the smallest decision tree, it's structure must be smaller or equal. Hence, it's easy to see that $x \geq \gamma \geq s$ and $y \geq \delta \geq d$. Thus, we can conclude that $s = x$ and $d = y$.

□

Note: [RGR22] says that this theorem should be generalizable to each tree and not only for the smallest trees (doubt this is true)

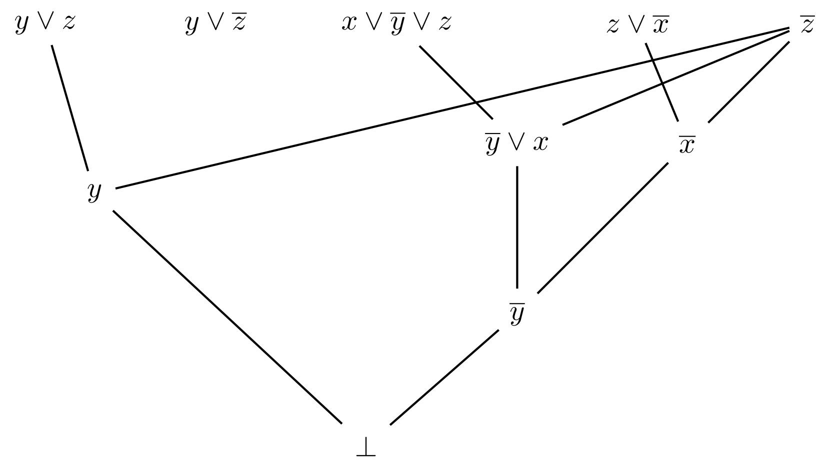

Figure 5.1. Dag-like refutation of the previous formula

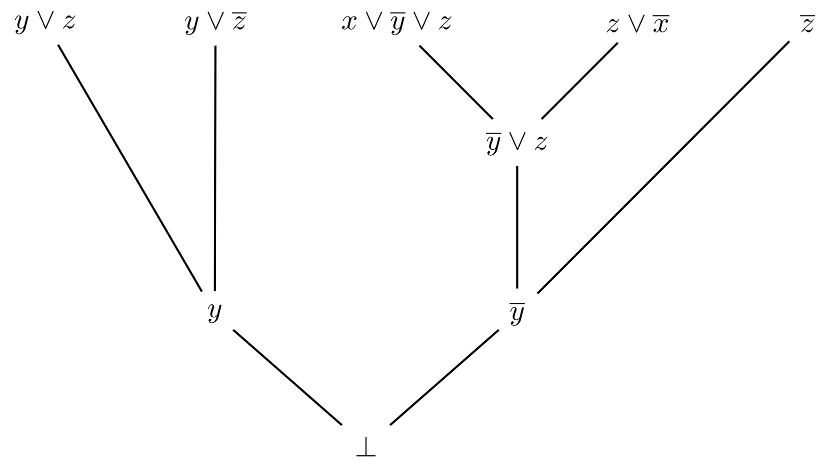

Figure 5.2. Tree-like refutation of the previous formula

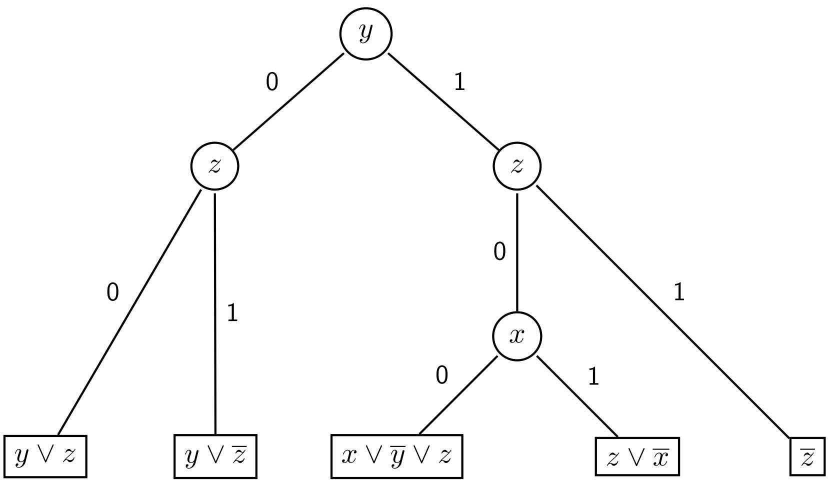

Figure 5.3. Decision tree for the previous formula

Acknowledgements

this is sad, alexa play despacito

Bibliography

[BCE+98] Paul Beame, Stephen Cook, Jeff Edmonds, et al. “The Relative Complexity of NP Search Problems”. In: *Journal of Computer and System Sciences* 57.1 (1998), pp. 3–19. ISSN: 0022-0000. DOI: `https://doi.org/10.1006/jcss.1998.1575`. URL: `https://www.sciencedirect.com/science/article/pii/S0022000098915756`.

[BFI23] Sam Buss, Noah Fleming, and Russell Impagliazzo. “TFNP Characterizations of Proof Systems and Monotone Circuits”. In: *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*. Ed. by Yael Tauman Kalai. Vol. 251. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 30:1–30:40. ISBN: 978-3-95977-263-1. DOI: `10.4230/LIPIcs.ITCS.2023.30`. URL: `https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.ITCS.2023.30`.

[BG94] Mihir Bellare and Shafi Goldwasser. “The Complexity of Decision Versus Search”. In: *SIAM Journal on Computing* 23.1 (1994), pp. 97–119. DOI: `10.1137/S0097539792228289`. URL: `https://doi.org/10.1137/S0097539792228289`.

[BGL13] Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. “A characterization of tree-like Resolution size”. In: *Information Processing Letters* 113.18 (2013), pp. 666–671. ISSN: 0020-0190. DOI: `https://doi.org/10.1016/j.ipl.2013.06.002`. URL: `https://www.sciencedirect.com/science/article/pii/S0020019013001592`.

[DK14] Ding-Zhu Du and Ker-I Ko. “Models of Computation and Complexity Classes”. In: *Theory of Computational Complexity*. 2014. Chap. 1, pp. 1–44. ISBN: 9781118595091. DOI: `https://doi.org/10.1002/9781118595091.ch1`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118595091.ch1`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118595091.ch1`.

[GHJ+22] Mika Göös, Alexandros Hollender, Siddhartha Jain, et al. “Separations in Proof Complexity and TFNP”. In: *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. 2022, pp. 1150–1161. DOI: `10.1109/FOCS54457.2022.00111`.

[LNN+95] László Lovász, Moni Naor, Ilan Newman, et al. “Search Problems in the Decision Tree Model”. In: *SIAM J. Discret. Math.* 8.1 (Feb. 1995), pp. 119–132. ISSN: 0895-4801. DOI: 10.1137/S0895480192233867. URL: https://doi.org/10.1137/S0895480192233867.

[RGR22] Susanna F. de Rezende, Mika Göös, and Robert Robere. "Proofs, Circuits, and Communication". In: *ArXiv* abs/2202.08909 (2022). URL: https://api.semanticscholar.org/CorpusID:246996726.