SAPIENZA
UNIVERSITÀ DI ROMA

Thesis Title

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

Simone Bianco
ID number 1986936

Advisor	Co-Advisor
Prof. Nicola Galesi	Prof. Massimo Lauria

Academic Year 2023/2024

Thesis Title
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

TODO.

Abstract

TODO.

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, not all decision problems can be solved efficiently by an algorithm, i.e. be decided in a reasonable amount of finite steps.

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 question: «*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 a problem 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 into 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 subfields of complexity theory afore mentioned. These correlations give a way to study these subfields through the lens of total functional problems, making it an *unifying theory*, even though

Add stuff on parity here

currently this statement seems to be farfetched for the white-box model.

Chapter 2

Preliminaries

2.1 Turing machines

2.2 Complexity measures

Chapter 3

Search problems

3.1 Decision vs. Search

As briefly discussed in the introduction, the concept of **functional problem** rose as a generalization of the concept of decision problem. Given a set of inputs called *language*, a **decision problem** on such language is a problem that asks «*does this input have the required property?*». Each input of the language can either have a positive or negative answer to the question. In other words, the output of the problem for each input can either be «*yes*» or «*no*».

Formally, each decision problem can be described as a subset of the given language, where an input is in the subset if and only if it gives a sayyes answer to the problem.

Definition 1. Given a language Σ^*, a decision problem for a required property p is a subset $L \subseteq \Sigma^*$ such that $L = \{x \in \Sigma^* \mid p(x) = True\}$.

For example, given the language $\mathbb{N}$ corresponding to the set of natural numbers, the question «*is n prime?*» is a valid decision problem on such language and it can be described as the subset $\text{PRIMES} = \{n \in \mathbb{N} \mid n \text{ is prime}\}$.

By their own nature, decision problems are limited. In particular, in some cases we are more interested in finding what gives the required property to a specific input. For example, we may be more interested in the question «*what is the prime factorization of n?*» instead of the previous one. Since for each input the answer to this question is neither a yes nor a no, this problem is by nature *more complex* than a decision problem. In general these types of problems are referred to as **functional problems**, i.e. problems that ask questions like «*what gives this object the following property?*».

An important thing to precise is that questions like «*is y a valid output for the input x?*» are still modeled by decision problems due to it requiring a simple yes-or-no answer, while a function problem would ask the question «*what is the output for the input x?*». For example, the question «*is $(p_1, \dots, p_k)$ the prime factorization of n?*» corresponds to the decision problem $\text{FACTORIZATION}_n = \{(p_1, \dots, p_k) \in \mathbb{N}^k \mid n = p_1 \cdot \ldots \cdot p_k\}$.

Formally, functional problems are described through the concept of **relation**: given of inputs X and a set of possible outputs Y of a functional problem, the latter can be described as a relation $R \subseteq X \times Y$. For example, the question «*what is the prime factorization of n?*» is modeled by the functional problem FACTORING $= \{(n, (p_1, \dots, p_k)) \in \mathbb{N} \times \mathbb{N}^k \mid n = p_1 \cdot \dots \cdot p_k\}$.

We observe that, even thought decision problems can indeed be modeled as functional problems whose outputs are only «*yes*» and «*no*», they aren't effectively a subset of functional problems due to them being defined in a different way. For example, the PRIMES problem can be modeled as the functional problem $\{(n, a) \in \mathbb{N} \times \{0, 1\} \mid 1 \text{ if } n \text{ is prime, } 0 \text{ otherwise}\}$, but they aren't effectively the same problem even thought they answer the same question.

Another important thing to notice is that even though the name implies a correlation to mathematical functions due to the concept of input-output being involved, 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 more general term **search problems**, describing the idea of finding a valid output for the given input, better suiting the previous formal definition.

To give a more detailed definition of search problems, we assume that these problems all share the language $\{0, 1\}^k$ for some $k \in \mathbb{N}$, describing all inputs as a sequence of bits. Since each problem could have inputs of different bit-lengths, we define search problems through the use of a *sequence of relations* rather than a single relation. This also allows search problems to have different types of outputs based on the length of the inputs [LNN+95; BCE+98; RGR22; BFI23].

Definition 2. 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 very subtle thing to notice is that this definition allows search problems to be *"undefined"* for some inputs. 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 other words, a total search problem has at least an output for all possible inputs, removing partial function from the context, while multi-valued functions are still allowed. For example, FACTORING is a total search problem due to each natural number having a guaranteed prime factorization.

3.2 The complexity classes FP, FNP and TFNP

In complexity theory, decision problems are grouped in two major categories: problems that can be **solved** efficiently and problems that can be **verified** efficiently. These classes are respectively referred to as P, the class of problems solvable by a deterministic Turing machine in polynomial time, and NP, the class of problems for which a solution can be verified by a deterministic verifier in polynomial time (or equivalently, the class of problems solvable by a non-deterministic Turing machine). By the very definition of these classes, it's easy to see that $\mathsf{P} \subseteq \mathsf{NP}$ since any problem efficiently solvable can also be efficiently verified.

Include the definition of verifier either here or in preliminaries

Similarly, in the context of search problems we define FP, *functional* P, as the class of search problems solvable by an algorithm in polynomial time and FNP, *functional* NP, as the class of search problems whose solutions are verifiable by an algorithm in polynomial time.

Definition 3. We define FP as the set of search problems $R = (R_n)_{n\in\mathbb{N}}$ for which there exists a polynomial time algorithm A_n such that $A_n(x) = y$ if and only if $(x, y) \in R_n$. Likewise, we define FNP as the set of search problems $R = (R_n)_{n\in\mathbb{N}}$ for which there exists a polynomial time verifier V_n such that $\exists w \in \{0,1\}^{\mathrm{poly(n)}}$ for which $V_n(x, y, w) = \text{True}$ if and only if $(x, y) \in R_n$.

An important remark to be made is that, since they are defined on two different types of problems, it doesn't hold that $\mathsf{P} \subseteq \mathsf{FP}$ or that $\mathsf{NP} \subseteq \mathsf{FNP}$. However, each decision problem can indeed be seen as a search problem with only two possible outputs: *true* or *false*. In fact, an important result shows that $\mathsf{P} = \mathsf{NP}$ if and only if $\mathsf{FP} = \mathsf{FNP}$, meaning that even though search problems are generally harder than decision problems, if the latter are efficiently solvable then the other also is [BG94; DK14].

Theorem 1. *P = NP if and only if FP = FNP*

Proof. Since each decision problem can be translated into a search problem with only two possible outcomes, we trivially get that if $\mathsf{FP} = \mathsf{FNP}$ then $\mathsf{P} = \mathsf{NP}$.

Suppose now that $\mathsf{P} = \mathsf{NP}$. We already know that $\mathsf{FP} \subseteq \mathsf{FNP}$, so we have to show that $\mathsf{FP} \subseteq \mathsf{FNP}$. Let $R = (R_n)_{n\in\mathbb{N}} \in \mathsf{FNP}$ be a search problem verifiable in polynomial time.

For each $n \in \mathbb{N}$, let $L_n = \{(x, y) \mid \exists z \in \{0,1\}^k, k \leq n \text{ s.t. } (x, zw) \in R_n\}$, i.e. the set of pairs (x, z) such that z is the prefix of an outcome zw for the problem R_n with input x. It's easy to see that $L_n \in \mathsf{NP}$ since each pair (x, z) is certified by the string zw itself and the correctness of this certificate can be polynomially verified since $R \in \mathsf{FNP}$.

Since $L_n \in \mathsf{NP} = \mathsf{P}$, we know that there is a polynomial time algorithm Partial_n that decides L_n. Thus, for each $n \in \mathbb{N}$, we can construct the following polynomial time algorithm Solve_n which directly concludes that $R \in \mathsf{FP}$ and thus that $\mathsf{FNP} \subseteq \mathsf{FP}$.

```
function Solve$_n(x)$
    $y = \varepsilon$                                    ▷ $\varepsilon$ is the empty string
    while True do
        if Partial$_n(x, y0)$ = True then
            $y = y0$
        else if Partial$_n(x, y1)$ = True then
            $y = y1$
        else
            return $y$
        end if
    end while
end function
```

□

As discussed in the previous section, not all search problems are total, meaning that a solution could not exist for some inputs. However, total search problems have actually proven to be more important than non-total ones. In fact, a lot of interesting real worlds problems have a *"guaranteed solution"*, ranging from simple number functions to harder problems.

Definition 4. We define the class TFNP as the subset of FNP problems that are also total.

For simplicity, we assume that *each search problem in FP is also total*: since problems in FP are solvable in polynomial time, when a solution doesn't exist we can output a pre-chosen «*doesn't exist*» solution, making the problem total. This assumption easily implies that FP $\subseteq$ TFNP $\subseteq$ FNP, giving us a proper hierarchy. For obvious reasons, this assumption wouldn't work for FNP problems since the only way to polynomially verify that a solution doesn't exist would be to solve the problem itself and find that there is no solution, but this would mean that FP = FNP is trivially true.

Another less common way to view total search problems is through the lens of *polinomial disqualification*. In decisional problems, the class coNP contains all the problems whose complement is in NP. For search problems, we define the class FcoNP in the same way. If the complementary problem is polynomially verifiable, this means that there is a polynomial verifier that can decide if an input doesn't have the required property, effectively disqualifying it. In particular, the class TFNP corresponds to the class F(NP $\cap$ coNP), which contains search problems whose inputs can both be certified and disqualified in polynomial time [MP91].

Theorem 2. TFNP = F(NP $\cap$ coNP)

Proof. If $R = (R_n)_{n \in \mathbb{N}} \in$ TFNP then we know that every input x has an output y. However, this means that the complementary problem $\overline{R}$ is empty, meaning that each input is trivially verifiable in polynomial time and thus that $\overline{R} \in$ FNP. Hence, we conclude that $R \in$ F(NP $\cap$ coNP).

Vice versa, if $S = (S_n)_{n \in \mathbb{N}} \in \mathsf{F(NP \cap coNP)}$ then trivially we have that $S \in \mathsf{FNP}$. Moreover, since $S \in \mathsf{F(NP \cap coNP)}$ we know that each input x can be easily certified or disqualified in polynomial time, meaning that each input must have a solution polynomially verifiable and thus that $S \in \mathsf{TFNP}$. □

3.3 The **TFNP** hierarchy

One of the most studied properties of problems, either decisional or functional, is the ability to be **reduced** into other problems. In particular, a problem A is said to be reducible into a problem B if any instance of A can be mapped into an instance of B whose solution gives a solution to the former. If a problem A is reducible to a problem B, we say that $A \leq B$.

In decision problems, this concept is easily described trough the use of many-to-one mappings, i.e. a function that maps instances of the original problem to instances of the reduced problem.

Definition 5. A decision problem A is many-to-one reducible to a decision problem B, namely $A \leq B$, if there is a computable function f such that $x \in A$ if and only if $f(x) \in A$. If the function can be computed in polynomial time, we say that $A \leq_p B$.

Just like decision problems, search problems can also be reduced into other search problems through the use of functions, even thought the definition is slightly different.

Definition 6. A search problem $R = (R_m)_{m \in \mathbb{N}}$, where $R_m \subseteq \{0,1\}^m \times O_m$ is said to be many-to-one reducible to a search problem $S = (S_n)_{n \in \mathbb{N}}$, namely $R \leq S$, where $S_n \subseteq \{0,1\}^n \times O'_n$, if for all $m \in \mathbb{N}$ there is an $n \in \mathbb{N}$ for which there is a function $f : \{0,1\}^m \to \{0,1\}^n$ and a function $g : \{0,1\}^m \times O'_n \to O_m$ such that:

$$\forall x \in \{0,1\}^m \;\; (f(x), y) \in S \implies (x, g(x,y)) \in R$$

In other words, the function f maps inputs of R into inputs of S, while the function g maps solutions of S into solutions of R.

Differently from decisional problems, this definition doesn't require that non-solutions of S are also mapped into non-solutions of R. This weaker property is needed by the very own nature of search problems since any non-solution to a problem can be reduced into a non-solution of any other problem, which partially loses the whole sense of problem reduction.

Even though it is generally useful, reductions play a critical role in **class completeness**, that being the property of all problems of a class to be reduced into one specific problem.

State that many-to-one reductions are transitive

Definition 7. A problem B is said to be complete for a class $\mathcal{C}$ if $B \in \mathcal{C}$ and $\forall A \in \mathcal{C}$ it holds that $A \leq B$.

By definition, if a complete problem is proven to have a specific property then that property gets automatically inherited by all the problems of the class. In particular, for the classes P, NP, FP and FNP, a problem is complete only if his reductions are computable in polinomial time, meaning that for all problems A in one of these classes it holds that $A \leq_p B$. This restrictions is obvious: if the reductions weren't computable in polynomial time then there would be no way of obtaining a solution in polynomial time through the reduction.

Moreover, every problem inside the class P is P-Complete, while only a few NP problems are NP-Complete. If a single NP-Complete problem is proven to be inside the class P, i.e. it can be efficiently decided by a Turing machine, the whole NP class would collapse, meaning that P = NP. Some well-known NP-Complete problems include the SAT problem, which asks «*does this formula have an assignment that satisfies it?*», the CLIQUE problem, which asks «*does this graph have a k-clique?*», and the COLORING problem, which asks «*does this graph have a k-coloring?*».

In the search problem world, the functional versions of each of these problems, namely FSAT, FCLIQUE and FCOLORING are also FNP-complete. In fact, the functional versions can be used to prove that the decisional version is complete and vice versa. However, it is not known if there is a FNP-complete problem that is also total. For example, the problem FSAT isn't total due to some formulas being unsatisfiable, thus there is no output assignment that satisfies them.

For these reasons, in the TFNP case the concept of completeness is studied under a *different approach*: instead of considering problems that are complete for the whole class, we consider important problems who have a lot of TFNP problems reducible to them. These important problems form additional subclasses of TFNP. In other words, given a TFNP problem $R = (R_n)_{n \in \mathbb{N}}$, we define the subclass R as the set of TFNP problems reducible to R. Unexpectedly, the majority of TFNP problems can actually be characterized with very subclasses.

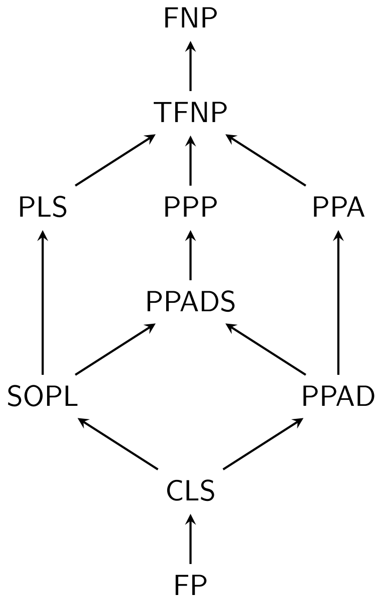

Figure 3.1. Hierarchy of the most commonly defined total search problem subclasses. An arrow from class A to class B means that $A \subseteq B$.

Each of the subclasses shown in the above hierarchy is characterized by a total search problem [RGR22; BFI23]. Such search problems are actually guaranteed to be total by **simple existence principles**. In fact, we could say that many TFNP problems are actually described by reduction to these simple principles:

- PLS (Polynomial Local Search): the class of search problems designed to model the process of finding the local optimum of a function or alteratively the class of problems whose solution is guaranteed by the «*Every directed acyclic graph has a sink*» principle. It is formally defined as the class of search problems that are polynomial-time reducible to the SINK-OF-DAG problem .

Maybe write problem names in italics

- PPP (Polynomial Pigeonhole Principle): the class of problems whose solution is guaranteed by the «*Every mapping from a set of* $n+1$ *elements to a set of* n *elements has a collision*» principle. It is defined as the class of problems that are polynomial-time reducible to the PIGEON problem.
- PPA (Polynomial Parity Argument): the class of problems whose solution is guaranteed by the «*Every undirected graph with an odd-degree node must have another odd-degree node*» principle. It is defined as the class of problems that are polynomial-time reducible to the LEAF problem
- PPADS (Polynomial Parity Argument - Directed with Sink): the class of problems whose solution is guaranteed the «*Every directed graph with a positively unbalanced node (out-degree* $>$ *in-degree) must have a negatively unbalanced node*» principle. It is defined as the class of problems that are polynomial-time reducible to the SINK-OF-LINE problem.
- SOPL (Sink of Potential Line): the class of problems that are polynomial-time reducible to the SINK-OF-POTENTIAL-LINE problem. It has been proven that $\mathsf{SOPL} = \mathsf{PLS} \cap \mathsf{PPADS}$ [GHJ+22a]
- PPAD (Polynomial Parity Argument - Directed): the class of problems whose solution is guaranteed the «*Every directed graph with an unbalanced node must have another unbalanced node*» principle. 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. It has been proven that $\mathsf{CLS} = \mathsf{EOPL} = \mathsf{PLS} \cap \mathsf{PPAD}$ [FGH+22; GHJ+22a], where EOPL is the class of search problems that are polynomial-time reducible to the END-OF-POTENTIAL-LINE problem.

The extensive study of TFNP classes has been successful in capturing the complexity of many important common problems. In fact, a lot of problems from *cryptography*, *game theory* and *economics* are actually reducible to TFNP complete problems. For example, the NASH problem relative to finding a Nash equilibrium of a given game has been shown to be PPAD-Complete.

By hardness of the question itself, any unconditional separation between these subclasses seems to be completely out of reach just like in the decisional case. However, it turns out that the TFNP model indeed has separations relative to *oracles*, i.e. the **black-box TFNP** model.

3.4 White-box TFNP

In computer science and engineering, systems and models are divided in two categories: white-box systems and black-box system. A system is said to be a **white-box** if only its inputs, its outputs and its actual internal workings are known. Contrary, a system is said to be a **black-box** if its internal workings are unknown. The idea behind a black-box model is that we only care about the result for a given input and not how that result is achieved. For example, a programmer uses both white-box and black-box systems: personal functions are white-boxes, while ready-to-go library functions are black-boxes.

Each TFNP problem can be analyzed through the lens of both white-box and black-box systems: in a **white-box TFNP** problem we're interested in how the problem gets verified (or computed if it's also in FP), while a **black-box TFNP** problem we're interested only in the verifiability (or computability) of the problem. In recent years, these two TFNP models have been formalized through the use of *protocols* and *decision trees*. In this section we will briefly discuss protocols and the white-box model, while decision trees and the black-box model will be extensively discussed in the following chapter.

A *protocol* is an algorithm that generates communications between two parties, namely Alice and Bob, who cooperate in order to achieve a common objective, like computing a function.

Definition 8 ([RYM+22])**.** Let X be Alice's input set and let Y be Bob's input set. A protocol π is a rooted directed binary tree whose leaves are associated to outputs and internal nodes are owned by either Alice or Bob, where the owner of v is noted by $\text{owner}(v)$. Each leaf is labeled with an output $o \in O$, where O is the outcome set. Each internal node v is also associated to a function $g_v : Z \to \{0, 1\}$, where $Z = X$ if $\text{owner}(v) = \text{A}$ and $Z = Y$ if $\text{owner}(v) = \text{B}$. When given the input $(x, y) \in X \times Y$, the protocol computes the associated function of the current node (starting from the root), proceeding on the left child if the output is 0 and on the right child if the output is 1. When a leaf is reached, the protocol returns the associated output. The output of the protocol for a given input (x, y) is denoted by $\pi(x, y)$.

A function f is said to be computed by the protocol π if $f = \pi$. By their own definition, protocols are also Turing complete since they are nothing more then an algorithm computed by two parties instead of one. A protocol encodes all possible messages that may be sent by the parties during any potential conversation, producing a conversation only when it is executed using a particular input. Since a protocol always returns an answer for all possible inputs, any protocol is *total*.

The complexity of protocols is measured in terms of their *size* and *depth*, that being the number of nodes of the protocol and the length of the longest directed path from the root node to a leaf. The **communication complexity** of a function f is defined as the depth of the smallest protocol that computes f, corresponding to the minimal number of bits that must be communicated by Alice and Bob to compute f for all inputs.

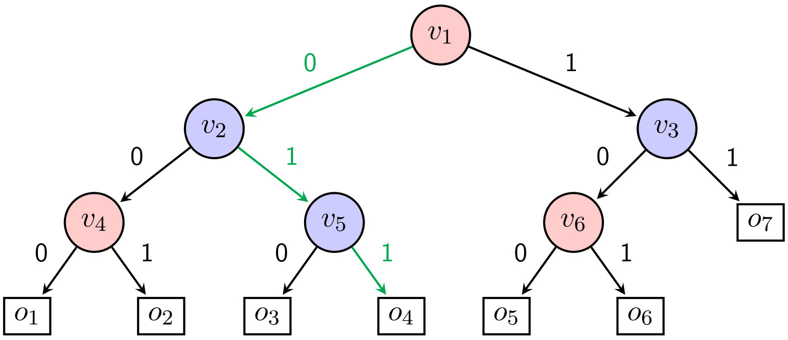

Figure 3.2. An example of a protocol of size 13 and depth 3 where the red nodes are owned by Alice and the blue nodes are owned by Bob. The computation, shown by the green path, for the inputs (x, y) is given by $f_{v_1}(x) = 0$, $f_{v_2}(y) = 1$ and $f_{v_5}(y) = 1$

Protocols are clearly white-boxes capable of solving search problems since every step of the computation is known, even though the way they are defined implies that for a given protocol any input will always return the same output due to them being a deterministic computation, losing the idea of an input being capable of having more solutions. However, they are still a valid way to solve search problems, making them a valid tool to model white-box TFNP [RGR22; BFI23].

Definition 9. A communication search problem is a sequence $R = (R_n)_{n\in\mathbb{N}}$ of relations $R_n \subseteq \{0,1\}^n \times \{0,1\}^n \times O_n$, one for each $n \in \mathbb{N}$, where each O_n is a finite set called "outcome set".

In particular, given a TFNP problem R, we denote with R^{cc} the equivalent TFNP^{cc} problem, where *cc* stands for *communication complexity.*

Definition 10. We define FP^{cc} as the set of communication search problems $R = (R_n)_{n\in\mathbb{N}}$ for which there exists a polylogarithmic depth protocol π_n such that $\pi_n(x, y) = z$ if and only if $((x, y), z) \in R_n$. Likewise, we define FNP^{cc} as the set of communication search problems $R = (R_n)_{n\in\mathbb{N}}$ for which there exists a polylogarithmic depth protocol V_n such that $V_n((x, y), z) = \text{True}$ if and only if $((x, y), z) \in R_n$.

Additionally, the concept of reduction also applies for communication search problems, even though it requires a pre-fixed value t of maximum amount of bits usable in the reduction, i.e. the maximum depth of the reduction protocol.

Definition 11. A communication search problem $R = (R_m)_{m\in\mathbb{N}}$, where $R_m \subseteq \{0,1\}^m \times \{0,1\}^m \times O_m$, is said to be reducible into a search problem $S = (S_n)_{n\in\mathbb{N}}$, namely $R \leq S$, where $S_n \subseteq \{0,1\}^n \times \{0,1\}^n \times O'_n$, if for all $m \in \mathbb{N}$ there is an $n \in \mathbb{N}$ for which there are two functions $f_X, f_Y : \{0,1\}^m \to \{0,1\}^n$ and a t-bit protocol $g : (\{0,1\}^m \times \{0,1\}^n) \times O'_n \to O_m$ such that:

$$\forall (x,y) \in \{0,1\}^m \times \{0,1\}^m \;\; (f_X(x), f_Y(y), z) \in S \implies (x, y, \pi((x,y),z)) \in R$$

In other words, the functions f_X, f_Y map inputs of R into inputs of S, while the protocol g maps solutions of S into solutions of R.

Nonetheless, this allows us to define the TFNP^{cc} hierarchy in the same way as the standard one, making communication complexity a valid system capable of characterizing white-box TFNP problems.

One of the most interesting properties of communication search problems is the ability to be reduced to be characterized by a single type of search problem: the *monotone Karchmer-Widgerson game.*

Definition 12. Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, we define the Karchmer-Wigderson game of f, denoted with $\mathrm{KW}(f)$, as the following communication problem: given the two inputs x and y, where $f(x) = 0$ and $f(y) = 1$, find an index $i \in [n]$ such that $x_i \neq y_i$.

Add description for [n] in preliminaries

If f is a monotone Boolean function, meaning that given two inputs x, y if $x \leq y$ then $f(x) \leq f(y)$, the monotone Karchmer-Widgerson game of f, denoted with $\mathrm{mKW}(f)$, finds an index $i \in [n]$ such that $x_i < y_i$.

In fact, it has been proven that any communication search problem is equivalent to the monotone KW game of some Boolean function [Gál02; GKR+19].

Lemma 1. *For any communication search problem $R = (R_n)_{n\in\mathbb{N}}$, where $R_n \subseteq \{0,1\}^n \times \{0,1\}^n \times O_n$, in t-bit TFNP^{cc}, there is a function f on $2^t\,|O_n|$ variables such that R is communication equivalent to $\mathrm{mKW}(f)$ under t-bit mapping reductions.*

This result implies that TFNP^{cc} actually coincides with the **study of the monotone Karchmer-Widgerson game**. Moreover, in 1990 Karchmer and Widgerson [KW88] showed that the complexity of these games is equal to the complexity of Boolean circuits solving the associated function, i.e. a circuit capable of computing a the function through logic gates.

Definition 13 ([RYM+22])**.** A Boolean circuit is a directed acyclic graph whose nodes, called gates, are associated with either input variables or Boolean operators. Each gate has an out-degree equal to 1 (except for the output gate who has out-degree 0) and in-degree equal to either 0 or 2. All 0 in-degree gates correspond to input variables, the negations of input variables or constant bits, while all 2 in-degree gates compute the logical AND or the logical OR of its given input variables or Boolean function. Each gate v is associated with the Boolean function f_v computed by it.

In particular, Boolean circuits have been proven to be *Turing-complete* [Sip96], meaning that they are capable of processing any computable function. In fact, every modern computer is actually just a bunch of Boolean circuits wired together. Moreover, Turing machines and circuits are capable of simulating each other up to a polynomial factor, so any polynomial complexity computation achievable through one can also be achieved through the other, making circuits a valid white-box system for analyzing TFNP problems.

The complexity of Boolean circuits is measured in terms of their *size* and *depth*, i.e. the number of gates of the circuit and the length of the longest directed path from an input gate to the output gate. The **circuit complexity** of function f is defined as the size of the smallest Boolean circuit that computes it.

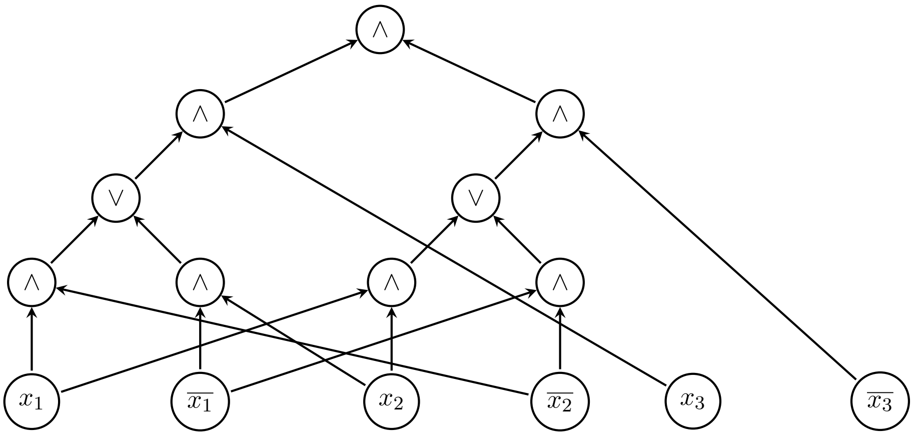

Figure 3.3. A Boolean circuit of size 15 and depth 4 computing $x_1 \oplus x_2 \oplus x_3$. The top gate is the output gate.

Theorem 3. *Given a function $f : \{0,1\}^n \to \{0,1\}$, there exists a circuit of depth d that computes f if and only if there exists a protocol of depth d that solves* KW(f)*. Moreover, if f is monotone, the circuit is monotone and the protocol solves* mKW(f)

Should be proven?

With this result, we conclude that TFNP^{cc} can be described in three ways: the study of communication search problems, the study of the monotone Karchmer-Widgerson game and the study of monotone Boolean circuits. This further extends the well-known connections between search problems, communication complexity and circuit complexity, establishing that any result obtained in one of these fields can be in some way extended to the others.

Add TFNP^{cc} hierarchy relative to circuits

Chapter 4

Black-box TFNP

4.1 Oracles and decision trees

In the previous chapter, we have discussed how TFNP subclasses are defined in terms of basic existence principles which can be converted into white-box total search problems solvable by protocols that are reducible to the Karchmer-Widgerson game. In this chapter, instead, we will extensively discuss the **black-box model**.

The difference between the white-box and black-box TFNP models can be formally described as the difference between problems verifiable (or computable) by a simple Turing machine or by a Turing machine equipped with an **oracle**.

Definition 14. An oracle for a problem A is an external device that is capable of instantaneously verifying a solution for of such problem. An oracle Turing machine is a Turing machine provided with the ability of querying an oracle. We write M^A to describe an oracle Turing machine provided with an oracle for the problem A.

By definition, it's easy to see that an oracle is nothing more than a black-box device. In particular, an oracle for a decision problem is capable of determining if an input object has the the required property, while an oracle for a search problem is capable of determining if an output is the solution for the associated problem with the given input. Any query to the oracle made by the Turing machine has a complexity of $\Theta(1)$, meaning that it doesn't affect the cost of the computation. This allows us to give the following definition of query search problem [RGR22; BFI23].

Definition 15. A query 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".

Additionally, oracles provide a simple yet effective way to generalize the concept of reduction, called **Turing reductions**: if a Turing machine provided with an oracle for the problem B is capable of resolving a problem A then the problem A can be reduced to the problem B. The idea behind this kind of reductions is simple: if M^B can solve A then any query to the oracle can be replaced with a call to a subroutine that solves B. Many-to-one reductions can be seen as restricted variants of Turing reductions where the number of calls made to the subroutine of problem B is exactly one and the value returned by the reduction is the same value as the one returned by the subroutine.

More generally, given a class $\mathcal{C}$ and an oracle for a problem A, the *relativized version* of the class $\mathcal{C}$ is the set of all problems of $\mathcal{C}$ verifiable (or solvable) with access to the oracle of A. Obviously, this definition implies that $\mathcal{C} \subseteq \mathcal{C}^A$ for all oracles A since any problem that is already in $\mathcal{C}$ can just ignore the oracle. In the particular case of TFNP, it was proven that the relation between each total search problem is strictly connected to the relation of the relativized versions of their classes [BCE+98].

Theorem 4. *Given two search problems $R, S \in \mathsf{TFNP}$ and their relative classes it holds that $R \subseteq S$ if and only if $R^A \subseteq S^A$ for all oracles A.*

This result states that proving any relativized separation is equivalent to proving a non-relativized separation, allowing us to use the intuitive nature of oracles to rule out possible collapses in TFNP subclasses. Through these relativized separations, many TFNP subclasses have been proven to be different.

Moreover, the use of oracles allows us to model total search problem through the lens of **decision trees**.

Definition 16 ([LNN+95])**.** A decision tree is a rooted directed binary tree whose nodes are associated with either an output value or an input Boolean variable. Each leaf is labeled with an output $o \in O$, where O is the outcome set. Each internal node is labeled by a variable and the two outgoing edges are labeled by the two possible values of that variable.

Decision trees can be viewed as nothing more than the *black-box version* of protocols: we don't care about who computes the next step and how they do it, we only care about the result being either a 0 or a 1 in order to proceed with the computation. In fact, like their white-box counterpart, a decision tree encodes all possible ways to obtain a result, making them *total*. Likewise, the complexity of a decision tree computing a function follows the same complexity measures as a protocol, i.e. its *size* and its *depth*.

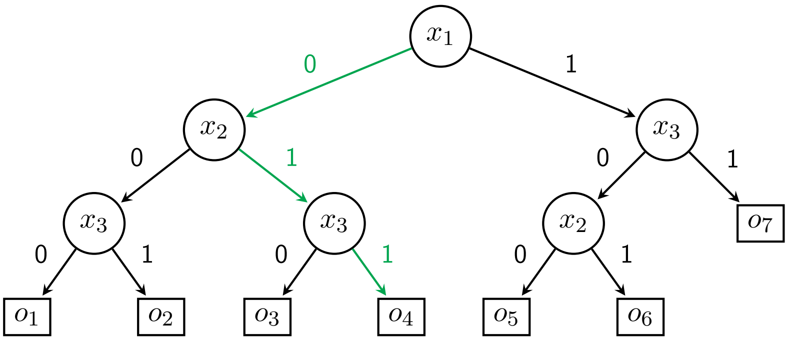

Figure 4.1. An example of decision tree of size 13 and depth 3. The computation, shown by the green path, for the input $x \in \{0,1\}^n$ is given by $x = 011$

Decision trees give an easier way to describe the computation of an oracle Turing machine: if M^B verifies (or solves) a problem A then the i-th query made by the procedure corresponds to a variable x_i for the decision tree where $x_i = 1$ if the query returns a positive result and 0 otherwise. In other words, the computation tree of an oracle Turing machine is actually a decision tree.

Proposition 1. *If there is an oracle Turing machine M^B that verifies (or solves) a problem A then there is a decision tree that verifies (or solves) A.*

Not sure if it's an if and only if, possibly yes

The above proposition gives a strong result that allows us to characterize black-box TFNP through decision trees instead of oracles: *any decision tree separation implies a relativized separation.* As in the communication complexity formulation described for the white-box model, given a TFNP problem R, we denote with R^{dt} the equivalent TFNP^{dt} problem, where dt stands for *decision tree* [RGR22; BFI23].

Definition 17. We define FP^{dt} as the set of query search problems $R = (R_n)_{n\in\mathbb{N}}$ for which there exists a polylogarithmic depth decision tree T_n such that $T_n(x) = y$ if and only if $(x, y) \in R_n$. Likewise, we define FNP^{dt} as the set of query search problems $R = (R_n)_{n\in\mathbb{N}}$ for which there exists a polylogarithmic depth decision tree T_y such that $T_y(x) = \text{True}$ if and only if $(x, y) \in R_n$.

Furthermore, we also introduce the concept of decision tree reductions. Differently from classic and communication search problems, these reductions are based on a more fine-grained definition, where the function that maps inputs of the first problem to inputs of the second problem is computed by many decision trees with output $\{0, 1\}$. This definition will allow us to prove some following results in a more convenient way. An analogous formulation can be obtained by computing this function through a simple decision tree.

Definition 18. A query search problem $R = (R_m)_{m\in\mathbb{N}}$, where $R_m \subseteq \{0,1\}^m \times O_m$ is said to be many-to-one reducible to a query search problem $S = (S_n)_{n\in\mathbb{N}}$, namely $R \leq S$, where $S_n \subseteq \{0,1\}^n \times O'_n$, if for all $m \in \mathbb{N}$ there is an $n \in \mathbb{N}$ for which there is a family of decision trees $T_i : \{0,1\}^m \to \{0,1\}$ for each $i \in [n]$ and a decision tree $T^y : \{0,1\}^m \to O_m$ for each $y \in O'_n$ such that:

$$\forall x \in \{0,1\}^m \;\; (T(x), y) \in S \implies (x, T^y(x)) \in R$$

where $T(x) := (T_1(x), \dots, T_n(x))$. In other words, the decision trees $T_1, \dots, T_n$ map inputs of R into inputs of S, while the decision tree g maps solutions of S into solutions of R.

The difference in notation between $T_1, \dots, T_n$ and T^y underlines the fact that the former give a $\{0, 1\}$ output, while the latter gives a more complex output.The *size* s of the reduction is the number of input bits to S, that being n. The *depth* d of the reduction is the maximum depth of any tree involved in the reduction, including both $T_1, \dots, T_n$ and T^y. The **complexity of the reduction** is given by $\log s + d$. 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 $S = (S_n)_{n\in\mathbb{N}}$, we define the subclass of problems reducible to S as:

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

where $R = (R_m)_{n\in\mathbb{N}}$.

4.2 Proof complexity characterization

4.3 (?) An in-depth example: FP^{dt} and TreeRes

Chapter 5

Parity in black-box **TFNP**

5.1 Parity decision trees and FP^{pdt}

5.2 Linear resolution over $\mathbb{F}_2$

5.3 Equivalence between PDTs and $\mathrm{TreeRes}(\oplus)$

5.4 Nullstallensatz over $\mathbb{F}_2$

5.5 $\mathrm{NS}-\mathbb{F}_2$ simulates $\mathrm{TreeRes}(\oplus)$

Chapter 6

Notes

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

Proposition 2. *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:

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

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 19. 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 20. 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:

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

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 2. *[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* $\mathsf{Search}(F)$

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, $\mathsf{Search}(F)$ 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 $F\lceil_{x=0}$ and $F\lceil_{x=1}$, 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 $F\lceil_{x=0}$ and $F\lceil_{x=1}$: 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 $\mathsf{Search}(F\lceil_{x=0})$ and $\mathsf{Search}(F\lceil_{x=1})$.

Finally, the search problem $\mathsf{Search}(F)$ 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 21. 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 3. *[BGL13; LNN+95] Let F be an unsatisfiable* CNF *formula. If there is a decision tree with structure T that solves* $\mathsf{Search}(F)$*, 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 $\bot$, concluding that T' is the structure of a tree-like refutation of F.

□

Theorem 5. *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 2, 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 3, 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)

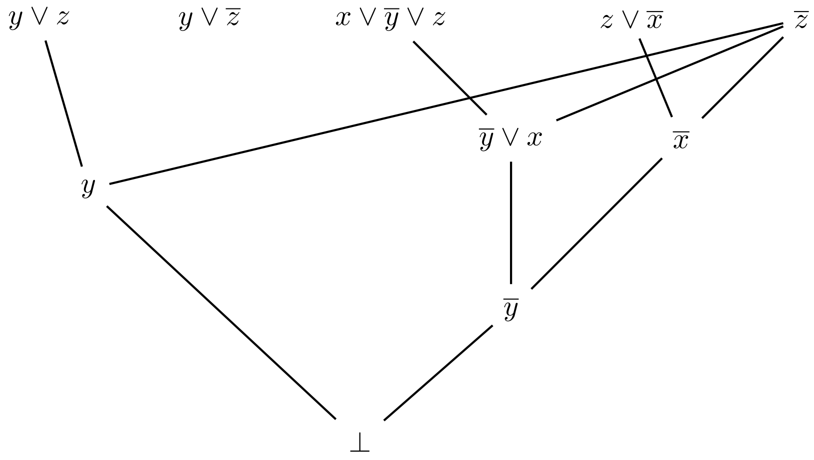

Figure 6.1. Dag-like refutation of the previous formula

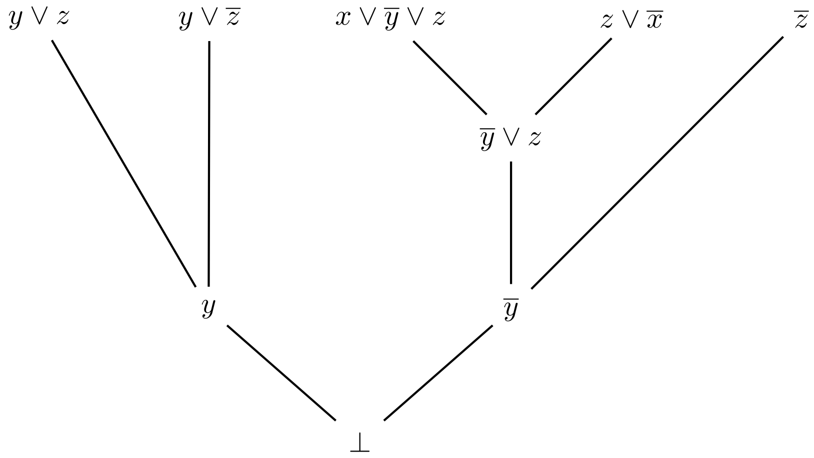

Figure 6.2. Tree-like refutation of the previous formula

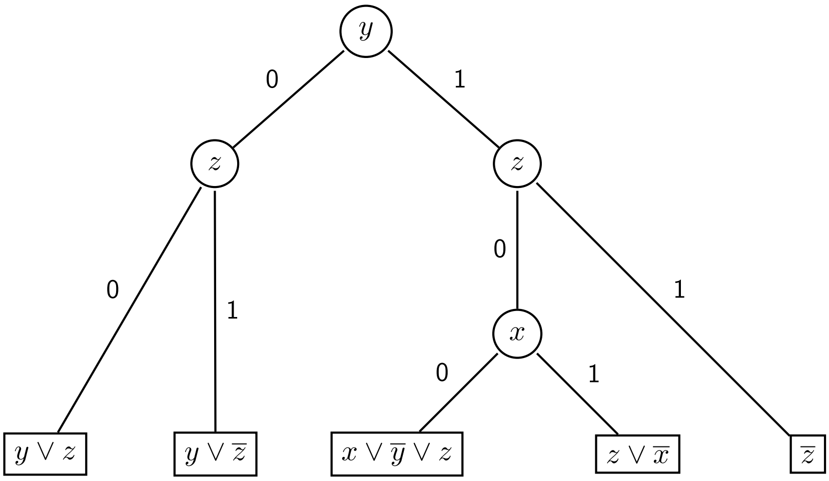

Figure 6.3. Decision tree for the previous formula

6.1 The $\frac{1}{3}, \frac{2}{3}$ lemma

Definition 22. Given a tree T and a node v, we denote as T_v the subtree of T having v as its radix.

Lemma 4 (Lewis' $\frac{1}{3}, \frac{2}{3}$ lemma [**1-3_2-3**])**.** *If T is a binary tree of size $s > 1$ then there is a node v such that the subtree T_v has size between $\left\lfloor \frac{1}{3}s \right\rfloor$ and $\left\lceil \frac{2}{3}s \right\rceil$.*

Proof. Let r be the radix of T and let ℓ be a leaf of T with the longest possible path $r \to \ell$. Let $v_1, \dots, v_k$ be the nodes of such path, where $r = v_1$ and $\ell = v_k$. For each index i such that $1 \leq i \leq k$, let $a_i b_i$ be the two children of v_i.

Claim 4.1. For any index i, if T_{v_i} has size at least $\left\lfloor \frac{1}{3}s \right\rfloor$ then for some index j, where $i \leq j \leq k$, it holds that T_{v_j} has size between $\left\lfloor \frac{1}{3}s \right\rfloor$ and $\left\lceil \frac{2}{3}s \right\rceil$.

Proof of the claim. If T_{v_i} has also size less than $\left\lceil \frac{2}{3}s \right\rceil$ then we are done. Otherwise, since $T_{v_i} = \{v_i\} \cup T_{a_i} \cup T_{b_i}$, one between the subtrees T_{a_i}, T_{b_i} must have size at least $\frac{1}{2} \lceil 2 \rceil 3s - 1$, meaning that it has size at least $\left\lfloor \frac{1}{3}s \right\rfloor$. If this subtree has also a size at most $\left\lceil \frac{2}{3}s \right\rceil$ then we are done. Instead, if this doesn't hold for both subtrees, we can repeat the process (assuming that $v_{i+1} := a_i$ without loss of generality) since we know that $T_{v_{i+1}}$ has size greater than $\left\lfloor \frac{1}{3}s \right\rfloor$.

By way of contradiction, suppose that this process never finds a subtree with size at most $\left\lceil \frac{2}{3}s \right\rceil$. Then, this would mean that it also holds for $v_k = \ell$. However, since ℓ is a leaf, we know that T_{v_ℓ} must have size 1, which is definitely at most $\left\lceil \frac{2}{3}s \right\rceil$ for any value of s, giving a contradiction. Thus, there must be a node that terminates the process.

□

Since $T_{v_1} = \{r\} \cup T_{a_1} \cup T_{b_1}$, we know that for both of these subtrees must have at least $\left\lfloor \frac{1}{3}s \right\rfloor$. Thus, assuming that $a_1 = v_2$, the claim directly concludes the proof.

□

6.2 Nullstellensatz

Definitions taken from [**Nullstellensatz**]

Definition 23 (Hilbert's Nullstellensatz)**.** Given the polynomials $p_1, \dots, p_m \in \mathbb{F}[x_1, \dots, x_n]$, the equation $p_1 = \ldots = p_m = 0$ is unsolvable if and only if $\exists g_1, \dots, g_m \in \mathbb{F}[x_1, \dots, x_n]$ such that $\sum\limits_{i=1}^{m} g_i p_i = 1$.

Hilbert's Nullstellensatz can be used to define the following proof system:

Definition 24 (Nullstellensatz Refutation)**.** Given the set of polynomial equations $P = \{p_1 = 0, \dots, p_m = 0\}$ over $\mathbb{F}[x_1, \dots, x_n]$, where $\mathbb{F}$ is any field, a Nullstellensatz refutation is a set of polynomials $\pi = \{g_1, \dots, g_n\} \subseteq \mathbb{F}[x_1, \dots, x_n]$ such that $\sum\limits_{i=1}^{m} g_i p_i = 1$.

The set of polynomials $P = \{p_1, \dots, p_n\}$ is called the axiom set and the set $\pi = \{g_1, \dots, g_n, h_1, \dots, h_m\}$ is called proof of P.

By also adding the polynomial equations $x_1^2 - x_1 = 0, \dots, x_n^2 - x_n = 0$ to the set of axioms, the NS proof system is sound and complete for the set of unsatisfiable CNF formulas. Thus, in general, given the set of axioms $P = \{p_1 = 0, \dots, p_m = 0, x_1^2 - x_1 = 0, \dots, x_n^2 - x_n = 0\}$, we say that $\pi = \{g_1, \dots, g_m, h_1, \dots, h_n\}$ is a CNF proof of P if:

$$\sum_{i=1}^{m} g_i p_i + \sum_{j=1}^{n} h_j(x_j^2 - x_j) = 1$$

For any proof $\pi = \{g_1, \dots, g_n, h_1, \dots, h_m\}$ of the axioms $P = \{p_1, \dots, p_n\}$, we define the *degree of* π as:

$$\deg(\pi) = \max\{\deg(g_i p_i), \deg(h_j) + 2 \mid 1 \leq i \leq n, 1 \leq j \leq m\}$$

If P has a proof π of degree $\deg(\pi) = d$ then we say that $P \vdash_d^{\mathsf{NS}} 1$.

Proposition 3. *Given a set of axioms P, if $P \vdash_d^{\mathsf{NS}} q$ then $P, 1 - q \vdash_d^{\mathsf{NS}} 1$*

Proof. Since $P \vdash_d^{\mathsf{NS}} q$, we know that $\exists g_1, \dots, g_m, h_1, \dots, h_n \in \mathbb{F}[x_1, \dots, x_n]$ such that:

$$\sum_{i=1}^{m} g_i p_i + \sum_{j=1}^{n} h_j(x_j^2 - x_j) = q$$

where $\deg(q) = d$.

Let $p_{m+1} := 1 - q$ and $P' = P \cup \{p_{m+1} = 0\}$. We define $g'_1, \dots, g'_m, g'_{m+1}$ as:

$$g'_i = \begin{cases} 1 & \text{if } i = m+1 \\ g_i & \text{otherwise} \end{cases}$$

With simple algebra we get that:

$$\sum_{i=1}^{m+1} g'_i p_i + \sum_{j=1}^{n} h_j(x_j^2 - x_j) = g'_{m+1} p_{m+1} + \sum_{i=1}^{m} g'_i p_i + \sum_{j=1}^{n} h_j(x_j^2 - x_j) = (1 - q) + q = 1$$

thus $\pi = \{g'_1, \dots, g'_{m+1}, h_1, \dots, h_n\}$ is a proof of P. Moreover, since $\deg(q) = d$ implies that $\deg(g'_{m+1} p_{m+1}) = d$, it's easy to see that $\deg(\pi) = d$ holds, concluding that $P, 1 - q \vdash_d^{\mathsf{NS}} 1$

□

Lemma 5. *Given two disjoint axiom sets P_1, P_2, if $P_1, p \vdash_{d_1}^{\mathsf{NS}} 1$ and $P_2, 1 - p \vdash_{d_2}^{\mathsf{NS}} 1$ then $P_1, P_2 \vdash_{d_1 + d_2}^{\mathsf{NS}} 1$.*

Proof. Suppose that $P_1 = \{p_1, \ldots, p_m\}$ and $P_2 = \{q_1, \ldots, q_k\}$. Let $p_{m+1} = p$ and let $q_{k+1} = 1 - p$. By hypothesis, we know that

$$\sum_{i=1}^{m+1} g_i p_i + \sum_{j=1}^{n} a_j (x_j^2 - x_j) = 1$$

for some $g_1, \ldots, g_{m+1}, a_1, \ldots, a_n$, implying that:

$$\sum_{i=1}^{m} g_i p_i + \sum_{j=1}^{n} a_j (x_j^2 - x_j) = 1 - g_{m+1} p_{m+1} = 1 - g_{m+1} p$$

Likewise, we know that:

$$\sum_{i=1}^{k+1} r_i p_i + \sum_{j=1}^{n} b_j (x_j^2 - x_j) = 1$$

for some $r_1, \ldots, r_{k+1}, b_1, \ldots, b_n$, implying that:

$$\sum_{i=1}^{k} r_i p_i + \sum_{j=1}^{n} b_j (x_j^2 - x_j) = 1 - r_{k+1} q_{k+1} = 1 - r_{k+1}(1 - p)$$

We notice that:

$$\begin{aligned}(1-p)\left(\sum_{i=1}^{m} g_i p_i + \sum_{j=1}^{n} a_j (x_j^2 - x_j)\right) &= (1-p)(1 - g_{m+1}p)\\ &= 1 - g_{m+1}p - p + g_{m+1}p^2\\ &= 1 - p\end{aligned}$$

In the last step, we used the fact that, due to multilinearity, it holds that $p^2 = p$. Proceeding the same way, we find that:

$$\begin{aligned}p\left(\sum_{i=1}^{k} r_i p_i + \sum_{j=1}^{n} b_j (x_j^2 - x_j)\right) &= p\,(1 - r_{k+1}(1-p))\\ &= p\,(1 - r_{k+1} + r_{k+1}p)\\ &= p - r_{k+1}p + r_{k+1}p^2\\ &= p\end{aligned}$$

Now, we define $s_1, \ldots, s_{m+k}$

$$s_i = \begin{cases} g_i \cdot (1-p) & \text{if } 1 \leq i \leq m \\ r_i \cdot p & \text{if } m+1 \leq i \leq k \end{cases}$$

and $h_1, \ldots, h_n$ as $h_j = a_j \cdot (1-p) + b_j \cdot p$.

At this point, through simple algebra we get that:

$$\sum_{i=1}^{m+k} s_i p_i + \sum_{j=1}^{n} h_j(x_j^2 - x_j) =$$

$$(1-p)\left(\sum_{i=1}^{m} g_i p_i + \sum_{j=1}^{n} a_j(x_j^2 - x_j)\right) + p\left(\sum_{i=1}^{k} r_i p_i + \sum_{j=1}^{n} b_j(x_j^2 - x_j)\right) =$$

$$(1-p)(1-g_{m+1}p) + p\,(1 - r_{k+1}(1-p)) = p + 1 - p = 1$$

concluding that $\pi_3 = \{s_1, \dots, s_{m+k}, h_1, \dots, h_n\}$ is a proof of $P_1 \cup P_2$. Furthermore, we notice that:

$$\deg((1-p)(1-g_{m+1}p)) = \deg(1-p) + \deg(1-g_{m+1}p) = d_1 + d_2$$

and that:

$$\deg(p\,(1 - r_{k+1}(1-p))) = \deg(p) + \deg(1 - r_{k+1}(1-p)) = d_2 + d_1$$

Finally, we get that:

$$\deg(\pi_3) = \max(\deg((1-p)(1-g_{m+1}p)), \deg(p\,(1 - r_{k+1}(1-p)))) = d_1 + d_2$$

concluding that $P_1, P_2 \vdash^{\mathsf{NS}}_{d_1+d_2} 1$.

□

6.3 Treelike Res and Nullstellensatz

Definition 25 ($\mathbb{F}_2$-NS encoding of Res)**.** Given a Res linear clause $C = \bigvee_{i=0}^{k_1} x_i \vee \bigvee_{j=0}^{k_2} \overline{x_j}$, the $\mathbb{F}_2$-NS encoding of C is defined as $\mathrm{enc}(C) := \prod_{i=0}^{k_1} x_i \cdot \prod_{j=0}^{k_2} (1 - x_j)$.

In general, a Res($\oplus$) formula $F = C_1 \wedge \ldots \wedge C_m$ defined on the variables $x_1, \ldots, x_n$ gets encoded in $\mathbb{F}_2$-NS as the set of axioms $P_F = \{\mathrm{enc}(C_i) = 0 \mid 1 \leq i \leq m\} \cup \{x_j^2 - x_j = 0 \mid 1 \leq j \leq n\}$.

Theorem 6. *Let F be an unsatisfiable* CNF*. If T is* Res($\oplus$) *refutation of F of size s then there is* NS *refutation of F of degree $O(\log(s))$.*

Proof. Let $F = C_1 \wedge \cdots \wedge C_n$. We proceed by strong induction on the size s.

If $s = 1$ then the T contains only the empty clause $\bot$, meaning that it also is one of the starting clauses and thus one of the axioms. We notice that $\mathrm{enc}(\bot) = 1$, which easily concludes that $\bot \vdash_0^{\mathsf{NS}} 1$.

Suppose now that $s > 1$. Let $\mathcal{L}$ be axioms of T. Since T is a binary tree, by Lemma 4 we know that there is a clause C_k, i.e. a node, of T such that T_{C_k} has size between $\left\lfloor \frac{1}{3}s \right\rfloor$ and $\left\lceil \frac{2}{3}s \right\rceil$.

Let $T' = (T - T_{C_k}) \cup \{C_k\}$. Due to the size of T_{C_k}, we get that T' has size between $\left\lfloor \frac{1}{3}s \right\rfloor + 1$ and $\left\lceil \frac{2}{3}s \right\rceil + 1$. Moreover, we notice that since T is a treelike refutation it holds that T_{C_k} and T' work with different clauses (except C_k), thus their axioms are disjoint. Let $\mathcal{L}_1, \mathcal{L}_2$ be the two sets of axioms respectively used by T_{C_k} and T'.

By construction, we notice that T_{C_k} derives the clause C_k using the axioms $\mathcal{L}_1$, while T_{C_k} derives the clause $\bot$ using the axioms $\mathcal{L}_2, C_k$. Thus, since T_{C_k} and T' have size lower than s, by induction hypothesis we get that $\mathrm{enc}(\mathcal{L}_1) \vdash_{c_1 \cdot \log s}^{\mathsf{NS}} \mathrm{enc}(C_k)$ and $\mathrm{enc}(\mathcal{L}_2), \mathrm{enc}(C_k) \vdash_{c_2 \cdot \log s}^{\mathsf{NS}} 1$ for some constants c_1, c_2. By Proposition 3 we easily conclude that $\mathrm{enc}(\mathcal{L}_1), (1 - \mathrm{enc}(C_k)) \vdash_{c_1 \cdot \log s}^{\mathsf{NS}} 1$ and, by Lemma 5, that $\mathrm{enc}(\mathcal{L}_1), \mathrm{enc}(\mathcal{L}_2) \vdash_{(c_1+c_2) \cdot \log s}^{\mathsf{NS}} 1$. Finally, since $\mathcal{L}_1 \cup \mathcal{L}_2 = \mathcal{L}$, we get that $\mathrm{enc}(\mathcal{L}) \vdash_{(c_1+c_2) \cdot \log s}^{\mathsf{NS}} 1$, meaning that $\mathcal{L}$ has a NS refutation of degree $O(\log s)$.

□

6.4 Treelike $\mathsf{Res}(\oplus)$ and Nullstellensatz

Definition 26 ($\mathbb{F}_2$-NS encoding of Res)**.** Given a $\mathsf{Res}(\oplus)$ linear clause $C = \bigvee\limits_{i=0}^{k} (\ell_i = \alpha_i)$, the $\mathbb{F}_2$-NS encoding of C is defined as $\mathrm{enc}_{\oplus}(C) := \prod\limits_{i=0}^{k} (\alpha - \ell_i)$.

In general, a $\mathsf{Res}(\oplus)$ formula $F = C_1 \wedge \ldots \wedge C_m$ defined on the variables $x_1, \ldots, x_n$ gets encoded in $\mathbb{F}_2$-NS as the set of axioms $P_F = \{\mathrm{enc}_{\oplus}(C_i) = 0 \mid 1 \leq i \leq m\} \cup \{x_j^2 - x_j = 0 \mid 1 \leq j \leq n\}$.

Theorem 7 ([**res_parity**])**.**

1. *Every tree-like* $\mathsf{Res}(\oplus)$ *proof of an unsatisfiable formula F may be translated to a parity decision tree for F without increasing the size of the tree.*

2. *Every parity decision tree for an unsatisfiable linear CNF may be translated into a tree-like* $\mathsf{Res}(\oplus)$ *proof and the size of the resulting proof is at most twice the size of the parity decision tree (and where the weakening is applied only to the axioms).*

Corollary 1. *Every tree-like* $\mathsf{Res}(\oplus)$ *proof of an unsatisfiable formula F can be converted to a tree-like* $\mathsf{Res}(\oplus)$ *proof of at most double the size and with weakening applied only to the axioms.*

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`.

[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`.

[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`.

[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`.

[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`.

[FGH+22] John Fearnley, Paul Goldberg, Alexandros Hollender, et al. "The Complexity of Gradient Descent". In: *J. ACM* 70.1 (Dec. 2022). ISSN: 0004-5411. DOI: `10.1145/3568163`.

[Gál02] Anna Gál. "A characterization of span program size and improved lower bounds for monotone span programs". In: *Comput. Complex.* 10.4 (May 2002), pp. 277–296. ISSN: 1016-3328. DOI: `10.1007/s000370100001`. URL: `https://doi.org/10.1007/s000370100001`.

[GHJ+22a] Mika Göös, Alexandros Hollender, Siddhartha Jain, et al. "Further collapses in TFNP". In: *Proceedings of the 37th Computational Complexity Conference.* CCC '22. Philadelphia, Pennsylvania: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2022. ISBN: 9783959772419. DOI: `10.4230/LIPIcs.CCC.2022.33`.

[GHJ+22b] 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.

[GKR+19] Mika Göös, Pritish Kamath, Robert Robere, et al. “Adventures in Monotone Complexity and TFNP”. In: *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Ed. by Avrim Blum. Vol. 124. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, 38:1–38:19. ISBN: 978-3-95977-095-8. DOI: 10.4230/LIPIcs.ITCS.2019.38. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITCS.2019.38.

[KW88] Mauricio Karchmer and Avi Wigderson. “Monotone circuits for connectivity require super-logarithmic depth”. In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC ’88. Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 539–550. ISBN: 0897912640. DOI: 10.1145/62212.62265. URL: https://doi.org/10.1145/62212.62265.

[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.

[MP91] Nimrod Megiddo and Christos H. Papadimitriou. “On total functions, existence theorems and computational complexity”. In: *Theoretical Computer Science* 81.2 (1991), pp. 317–324. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(91)90200-L.

[RGR22] Susanna F. de Rezende, Mika Göös, and Robert Robere. “Proofs, Circuits, and Communication”. In: *ArXiv* abs/2202.08909 (2022).

[RYM+22] Anup Rao, Amir Yehudayoff, A Mir, et al. “Communication Complexity and Applications”. In: 2022.

[Sip96] Michael Sipser. *Introduction to the Theory of Computation*. 1st. International Thomson Publishing, 1996. ISBN: 053494728X.