

# The Complexity of Finding Tarski Fixed Points

Master Thesis

July 22, 2024

Nils Jensen

ADVISED BY

PROF. DR. BERND GÄRTNER

SEBASTIAN HASLEBACHER



# Abstract

Insert the abstract here.

# Acknowledgements

Insert the acknowledgements here.

# Contents

Contents	v
<b>1 Introduction</b>	<b>1</b>
1.1 Total Search Problems	1
1.2 The TFNP landscape	2
1.3 The TARSKI problem	2
1.4 Current algorithms for solving TARSKI	2
1.5 Location of TARSKI in <b>TFNP</b>	3
1.6 Thesis Outline	3
<b>2 Preliminaries</b>	<b>4</b>
2.1 Total search problems	4
2.1.1 Search problems	4
2.1.2 Reductions	5
2.1.3 Promise Problems	6
2.2 Representation of functions and sets	7
2.2.1 Representation of sets	7
2.2.2 Representation of functions	7
2.2.3 Complexity of boolean circuits	9
2.3 Subclasses of <b>TFNP</b>	10
2.3.1 Polynomial Local Search ( <b>PLS</b> )	10
2.3.2 Polynomial Parity Argument on Directed Graphs ( <b>PPAD</b> )	12
2.3.3 End of Potential Line ( <b>EOPL</b> )	13
2.4 The TARSKI Problem	14
2.4.1 Definition of the TARSKI Problem	14
2.4.2 Two algorithms for solving TARSKI	16
2.4.3 Lower bounds for TARSKI	19
2.4.4 Location of TARSKI in <b>TFNP</b>	19
<b>3 Reducing TARSKI to PPAD</b>	<b>21</b>
3.1 Presentation of the known reduction of TARSKI to <b>PPAD</b>	21
3.2 Introducing TARSKI*	22
3.3 Sperner's Lemma	24
3.3.1 Sperner's Lemma for Simplices	25
3.3.2 Sperner's Lemma for an integer lattice	26
3.4 Reducing TARSKI* to SPERNER	27
<b>4 Reducing TARSKI to EOPL</b>	<b>30</b>
4.1 Choosing a simplicial decomposition of the lattice — Freudenthal's Simplicial Decomposition	30
4.2 Orientation of a the simplicial decomposition	34
4.2.1 Orientation of a simplex	35

4.2.2	Orientation of a simplicial complex . . . . .	36
4.3	Sequences of simplices . . . . .	40
4.4	Properties of the coloring of TARSKI instances . . . . .	41
4.4.1	General properties of the coloring . . . . .	41
4.4.2	Properties of sequences of simplices . . . . .	42
4.5	No cycles in the ENDOFLINE instance . . . . .	44
<b>APPENDIX</b>		<b>47</b>
<b>Bibliography</b>		<b>48</b>
<b>Alphabetical Index</b>		<b>50</b>

# List of Figures

2.1	Example of a Boolean Circuit . . . . .	9
2.2	Computing a function with circuits . . . . .	10
2.3	Example of a LOCALOPT Problem . . . . .	11
2.4	Example of an END-OF-LINE Problem . . . . .	12
2.5	Example of an <b>EOPL</b> Problem . . . . .	13
2.6	Example of a TARSKI instance . . . . .	15
3.1	Setup for SPERNER'S LEMMA . . . . .	25
3.2	Example of SPERNER'S LEMMA . . . . .	26
3.3	Example of a simplicial decomposition of a lattice . . . . .	27
4.1	Orientation of a simplex . . . . .	37
4.2	Orientation of a simplicial complex . . . . .	38

# List of Tables

## 1.1 Total Search Problems

The study of computational complexity is central to computer science, its primary goal is to establish lower bounds on the complexity of various problems. Specifically, complexity theory attempts to prove that certain problems cannot be solved faster than a given time, as a function of the input size. This endeavor has proven to be particularly challenging for many problems where there is a significant gap between the best-known lower bounds and the upper bounds determined by existing algorithms.

A fundamental tool in complexity theory is the concept of reduction, which makes it possible to compare the difficulty of two problems. We say that a problem  $P_1$  is reducible to another problem  $P_2$  if  $P_1$  can be solved efficiently by solving  $P_2$ . This concept underlies the classification of problems into complexity classes-groups of problems that are mutually reducible.

Traditionally, complexity theory has focused on decision problems, which involve determining whether a given object has a given property. Examples include determining whether a graph contains a  $k$  clique, or whether a number is prime. These problems typically require a decision about whether an object belongs to a set of objects—a language—defined by a particular property.

However, real-world challenges often extend beyond simple decision making into the realm of search problems. In practical scenarios, the existence of a solution is typically assumed, and the task is not just to verify its existence, but to compute the solution itself. For example, instead of just detecting the presence of a  $k$  clique in a graph, one may need to explicitly find this clique or confirm its absence. Similarly, instead of just recognizing a number as prime, one might need to determine its prime factors.

Within this broader category of search problems lies a special subclass known as *total search problems*. These are characterized by the guaranteed existence of a solution, often proved by mathematical theorems. A notable example within this subclass is the problem of identifying a sink in a directed acyclic graph, where the existence of such a sink is guaranteed.

1.1 Total Search Problems . . .	1
1.2 The TFNP landscape . . .	2
1.3 The TARSKI problem . . .	2
1.4 Current algorithms for solving TARSKI . . . . .	2
1.5 Location of TARSKI in TFNP	3
1.6 Thesis Outline . . . . .	3

Here, *efficiently* generally means in polynomial time. We will define this and related concepts more strictly later.



## 1.2 The TFNP landscape

The class of **TFNP**, is the pendant of **NP**, in the sense that it is the class of all total search problems, where a solution can be checked for validity in polynomial time. The study of this complexity class has been a active research subject in the last years and has given rise to many interesting results.

Because it is unexpected that we can find **TFNP**-complete problems, the class has been studied using other tools. The main method which has been established is the use of syntactic subclasses. The idea is to build subclasses of **TFNP** which are created by using very classical and almost obvious existence result. Three of these subclasses are particular relevant to this thesis.

The first is the class **PPAD**, which is the class of total search problems where the existence of a solution is guaranteed by *Brouwer's fixed point theorem*. The problems which lie in **PPAD** are those that can be solved by walking along a directed graph, starting from an unbalanced vertex, and ending at an other unbalanced vertex [1].

The second class of interest is **PLS**, which is the class of total search problems which can be expressed as starting at a vertex of a directed acyclic graph, and finding a sink of this graph [2].

Finally the class **EOPL**, is the class of total search problems which can be expressed as starting at a source of a directed acyclic graph, and finding a vertex which is a sink [3]

[1]: Papadimitriou (1994), *On the complexity of the parity argument and other inefficient proofs of existence*

[2]: Johnson et al. (1988), *How easy is local search?*

[3]: Fearnley et al. (2018), *End of Potential Line*

## 1.3 The TARSKI problem

The main problem we study in this thesis is the TARSKI problem. The name sake of the TARSKI problem is TARSKI'S FIXED POINT THEOREM, which states that every monotone function on a complete lattice has a fixed point[4]. The TARSKI problem is the problem of finding such a fixed point for a given function  $f$  on a complete lattice  $L$ , or to find a violation of monotonicity of this function [5]. By Tarski's theorem, this problem is guaranteed to have a solution, and hence is a total search problem.

[4]: Tarski (1955), *A lattice-theoretical fixpoint theorem and its applications.*

[5]: Etessami et al. (2020), *Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria*

## 1.4 Current algorithms for solving TARSKI

TODO: Write this section.

## 1.5 Location of TARSKI in TFNP

It is known that the TARSKI problem lies in **PPAD** and in **PLS**. A very recent breakthrough has shown that the class  $\mathbf{PPAD} \cap \mathbf{PLS} = \mathbf{EOPL}$  [6]. This result immediately implies that the TARSKI problem is in **EOPL**, which in turn means that there must be a reduction from TARSKI to **EOPL**-complete problems, in particular to the **ENDOFPOTENTIALLINE** problem. The main goal of this thesis is to understand why TARSKI lies in **EOPL**, and to construct a reduction from TARSKI to the **ENDOFPOTENTIALLINE** problem, which is **EOPL**-complete.

[6]: Goos et al. (2022), *Further Collapses in TFNP*

## 1.6 Thesis Outline

TODO: Write this section.

This Chapter aims to establish the framework used throughout this thesis to study the TARSKI problem. It formally introduces the concept of total search problems, the complexity class **TFNP**, and its subclasses **PLS**, **PPAD**, and **EOPL**. In addition, in this Chapter, we will describe how we represent sets and functions in this framework and how their complexity is measured. Finally, we give a formal introduction to the TARSKI problem and a presentation of the known algorithms for solving it and its location in the **TFNP** landscape.

## 2.1 Total search problems

The study of complexity classes has traditionally focused on *decision problems*, which involve determining whether an object belongs to a set, also called a *language*. Notable examples include determining whether a Boolean formula is satisfiable or whether a  $k$ -clique exists in a given graph [7]. However, real-world questions often require explicit answers rather than existence results. For example, while deciding whether a function has a global minimum is a decision problem, the practical interest lies in identifying that minimum, which goes beyond mere existence.

Here, so-called *search problems* come into play.

### 2.1.1 Search problems

#### Definition 2.1 — Search Problem.

A *search problem* is given by a relation  $R \subset \{0,1\}^* \times \{0,1\}^*$ . For a given *instance*  $I \in \{0,1\}^*$  the computational problem, to find a *solution*  $s \in \{0,1\}^*$ , that satisfies:  $(I, s) \in R$  or output “No” if no such  $s$  exists.

We can view these search problems as decision problems by looking at the corresponding decision problem given by the language:

$$\mathcal{L}_R = \{I \in \{0,1\}^* \mid \exists s \in \{0,1\}^* : (I, s) \in R\}$$

This perspective leads us to ask classical complexity questions about search problems: Are these problems in **P**? in **NP**? Are they

2.1	Total search problems	4
2.1.1	Search problems . . .	4
2.1.2	Reductions . . . . .	5
2.1.3	Promise Problems . . .	6
2.2	Representation of objects . . . . .	7
2.2.1	Representing sets . . .	7
2.2.2	Representing functions	7
2.2.3	Complexity of circuits .	9
2.3	Subclasses of TFNP . .	10
2.3.1	PLS . . . . .	10
2.3.2	PPAD . . . . .	12
2.3.3	EOPL . . . . .	13
2.4	TARSKI Problem . . . .	14
2.4.1	TARSKI Definition . . .	14
2.4.2	TARSKI Algorithms . . .	16
2.4.3	Lower bounds for TARSKI	19
2.4.4	TARSKI in TFNP . . . .	19

[7]: Arora et al. (2009), *Computational complexity : a modern approach*

The “No” case can be encoded as some special binary string.

Here, we have rephrased the valid language as the pair of a problem instance and a valid solution.

**NP-hard?** It is readily apparent that search problems are at least as complex as their decision counterparts since solving a search problem inherently solves the associated decision question. This observation leads to an intriguing question: what if we remove the decision component from the problem? This scenario is achieved by ensuring that “No” is never a valid solution. Such problems, where every instance is guaranteed to admit a solution, are called total search problems.

**Definition 2.2 — Total search problems.**

A total search problem is a search problem given by a relations  $R \subset \{0,1\}^* \times \{0,1\}^*$ , such that for every given instance  $I \in \{0,1\}^*$  there is a solution  $s \in \{0,1\}^*$ , that satisfies:  $(I, s) \in R$ .

The complexity class **TFNP** as introduced in [1] is simply the class of all total search problems that lie in **NP**. Examples of **TFNP** problems are:

- ▶ **FACTORING**, the problem of finding the prime factors of a number. Every number admits a factorization into prime numbers, which can be checked in polynomial time.
- ▶ **NASH**, the problem of finding a Nash equilibrium in a bimatrix game [8];
- ▶ **MINIMIZE**, the problem of finding the global minimum of a convex function [9].

## 2.1.2 Reductions

Similarly to decision problems, we can also define reductions inside **TFNP**.

**Definition 2.3 — Many-to-one Reduction.**

For two problem  $R, S \in \mathbf{TFNP}$ , we say that  $R$  reduces (many to one) to  $S$  if there exist polynomial time computable functions  $f : \{0,1\}^* \rightarrow \{0,1\}^*$  and  $g : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$  such that for  $I, s \in \{0,1\}^*$ :

$$\text{If } (f(I), s) \in S \text{ then } (I, g(I, s)) \in R.$$

This means that if  $s$  is a solution to an instance  $f(I)$  in  $S$ , we can compute  $g(I, s)$  a solution to an instance  $I$  in  $R$

We also introduce the notion of Turing reduction in **TFNP**, analogously to the classical Turing reduction.

**Definition 2.4 — Turing Reduction.**

For two problems  $R, S \in \mathbf{TFNP}$ , we say that  $R$  Turing reduces

[1]: Papadimitriou (1994), *On the complexity of the parity argument and other inefficient proofs of existence*

This means that **TFNP** can be seen as an intermediate class between **P** and **NP**, containing all search problems where a solution is guaranteed to exist, and where one can efficiently check the feasibility of a candidate solution.

[8]: Daskalakis et al. (2009), *The Complexity of Computing a Nash Equilibrium*

[9]: Daskalakis et al. (2011), *Continuous Local Search*

Saying one can reduce  $R$  onto  $S$  can be understood as saying if one can solve  $S$  efficiently then I can solve  $R$  efficiently.

to  $S$  if a polynomial-time oracle Turing machine that solves  $R$  given access to an oracle for  $S$  exists.

### 2.1.3 Promise Problems

We have defined total search problems as problems where a solution always exists for *any* input in  $\{0, 1\}^*$ . However, in practice, we often study problems where a solution is guaranteed to exist only for a subset of the inputs. For instance, every convex function has a global minimum, but this existence result relies on the fact that we are given a convex function. This leads us to the notion of *promise problems* as introduced in [10]. Formally, we restrict the instance space to some subset  $\mathcal{X} \subset \{0, 1\}^*$ . We only require our algorithm to solve the problem for instances in  $\mathcal{X}$ , and it can behave arbitrarily on instances outside of  $\mathcal{X}$ .

[10]: Hollender (2021), *Structural Results for Total Search Complexity Classes with Applications to Game Theory and Optimisation*

We highlight that formally **TFNP** does not contain promise problems where  $\mathcal{X} \neq \{0, 1\}^*$ . However, in practice, we can still compute various instances of **TFNP** problems by restricting the input space to some subset  $\mathcal{X}$ . This is, for instance, the case when we can check in polynomial time whether an instance is in  $\mathcal{X}$ . Assume that we have a total problem  $R$  on  $X \subset \{0, 1\}^*$ , then we can define the promise problem  $R'$  on  $\{0, 1\}^*$  by adding a solution  $(I, \star)$  to  $R$  for all  $I \in \{0, 1\}^* \setminus R$ , where  $\star$  is some special binary string. Because it can be decided in polynomial time whether an instance is in  $\mathcal{X}$ , we can solve  $R'$  by checking whether the instance is in  $\mathcal{X}$  and then solving  $R$ , hence obtaining a problem in **TFNP**.

Another way to formally obtain a **TFNP** problem is by checking the validity of the input syntactically. For instance, if instances are supposed to be functions or boolean circuits, then it can be checked whether an instance is a valid encoding of a function or boolean circuit in polynomial time. For example, this is the case for the **TARSKI** problem, where the instances are boolean circuits, and the validity of the instances can be checked in polynomial time.

Finally, there is one final way to obtain a **TFNP** problem, which we want to highlight. We can add *violations* to the solution space. For example, if we are interested in finding the global minima of convex functions, we can construct a total search problem by:

- (1) Checking syntactically that the input defines a function;
- (2) Adding a violation of convexity to the solution space.

This means we can often construct a **TFNP** problem by adding violations to the solution space and checking the validity of the

A violation of convexity is given by a  $x, y \in \mathcal{D}_f$ , and  $t \in \{0, 1\}$  such that  $tf(x) + (1-t)f(y) < f(tx + (1-t)y)$ .

input syntactically when given a promise problem. However, it is essential to note that this is only sometimes the case and that constructing a **TFNP** problem from a promise problem can be a non-trivial task.

## 2.2 Representation of functions and sets

As we will see, the problems we will work with are given by questions of the form “find an  $x \in S$  such that  $f(x)$  has some property”. A natural question we will answer next is how we represent the input, that is, the set  $S$  and the function  $f$ . We start by describing how we represent sets.

### 2.2.1 Representation of sets

In this thesis, we will work with sets of the form  $S = \{0, \dots, 2^n - 1\}$ , which we will denote by  $[2^n]$ . Notice that this set can be identified with the set of binary strings of length  $n$ . We will denote the set of binary strings of length  $n$  by  $\{0, 1\}^n$ . Formally, the functions and the model we will use to represent the functions will use the underlying binary strings in  $\{0, 1\}^n$ . We often denote the integer  $x \in [2^n]$  instead of the binary string for notational convenience.

Similarly, when considering the  $d$ -dimensional case, we can represent the set  $L = [2^n]^d$ , which corresponds to a  $d$ -dimensional lattice with side length  $2^n$ , as the set of binary strings of length  $n \cdot d$ , i.e.  $\{0, 1\}^{nd}$ . Again, for simplicity, while the underlying functions rely on the binary strings, we often only denote the point  $(x_1, \dots, x_d) \in [2^n]^d$ , instead of its binary representation.

### 2.2.2 Representation of functions

Now that we have described the sets, we can describe how we represent the functions. We will represent the functions by using so-called boolean circuits. In this section, we will rely on the presentation of boolean circuits described in [11] and refer an interested reader to this source for a more detailed description.

[11]: Greenlaw et al. (1998), *Chapter 9*  
- *Circuit Complexity*

On a high level, a boolean circuit is a directed acyclic graph, where the nodes are called *gates*, and the edges are called *wires*. The sinks of the graphs are the output gates, and the sources are the input gates. We want to start by defining a gate formally.

**Definition 2.5 — Gate.**

A gate is a function  $g : \{0, 1\}^k \rightarrow \{0, 1\}$ , where  $k$  is the number of input wires of the gate.

This corresponds to the gate node, having  $k$  incoming edges, and one outgoing edge.

In this thesis, we will only consider the following types of gates:

- **AND-gate:**  $g(x_1, x_2) = x_1 \wedge x_2$ ,
- **OR-gate:**  $g(x_1, x_2) = x_1 \vee x_2$ ,
- **NOT-gate:**  $g(x) = \neg x$ .

Notice that we only consider gates with at most two inputs, as we can always represent a gate with  $k$  inputs as a composition of gates with at most two inputs.

Now, we can describe a boolean circuit formally as follows:

**Definition 2.6 — Boolean circuit.**

A boolean circuit  $C$  is a labeled finite directed acyclic graph, where each vertex has a *type*  $\tau$ , with

$$\tau(v) \in \{\text{INPUT}\} \cup \{\text{OUTPUT}\} \cup \{\text{AND}, \text{OR}, \text{NOT}\}$$

and with the following properties:

- If  $\tau(v) = \text{INPUT}$ , then  $v$  has no incoming edges. We call these vertices the *inputs gates*.
- If  $\tau(v) = \text{OUTPUT}$ , then  $v$  has one incoming edge. We call these vertices the *output gates*.
- If  $\tau(v) = \text{AND}$ , then  $v$  has two incoming edges. We call these vertices the *AND-gates*.
- If  $\tau(v) = \text{OR}$ , then  $v$  has two incoming edges. We call these vertices the *OR-gates*.
- If  $\tau(v) = \text{NOT}$ , then  $v$  has one incoming edge. We call these vertices the *NOT-gates*.

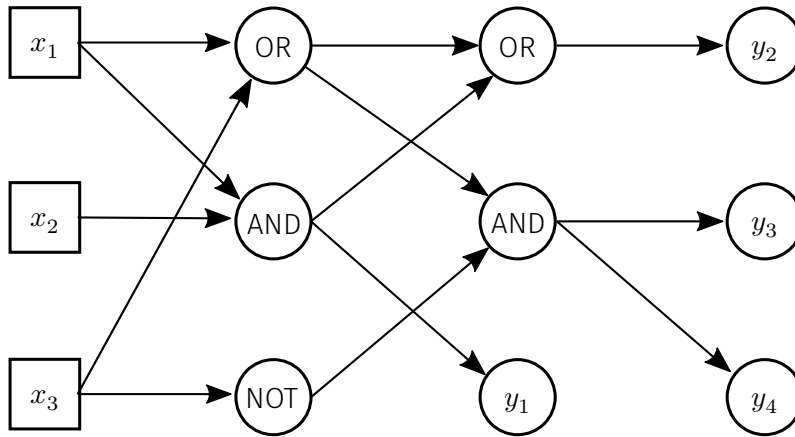
The inputs of  $C$  are given by a tuple  $(x_1, \dots, x_k)$  of distinct input gates. The output of  $C$  is given by a tuple  $(y_1, \dots, y_l)$  of distinct output gates.

We give an example of a boolean circuit in Figure 2.1. Of course, we now want to use a boolean circuit to represent a function. To do this, we need to define the function computed by a boolean circuit formally.

**Definition 2.7 — Computed function of a boolean circuit.**

A boolean circuit  $C$  with inputs  $x_1, \dots, x_n$  and outputs  $y_1, \dots, y_m$  computes a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  as follows:

- The input  $x_i$  is assigned the value of the  $i$ -th bit of the argument to the function.
- Every other vertex  $v$  is assigned the value of the gate  $g$  of the vertex, applied to the values of the incoming edges of  $v$ .
- The  $i$ -th bit of the output of the function is the value of the output gate  $y_i$ .



**Figure 2.1:** example of a Boolean circuit with three input and four output gates.

In Figure 2.2, we give an example of using a boolean circuit to compute a function, in particular for a function that is a TARSKI instance. From now on, we will formally represent all functions used in problems by boolean circuits.

### 2.2.3 Complexity of boolean circuits

Of course, formally, the complexity of a problem is defined in terms of the *size* of the input. This means we also need to define what we mean by the size of a boolean circuit. We will use the following definition:

**Definition 2.8 — Size of a boolean circuit.**

The size of a boolean circuit  $C$  is the number of gates in the circuit.

Size is a measure of the complexity of the input, i.e. it gives us an indication of how many bits we need to represent the input, it also tells us how many computations are made when computing the function output. We also define the depth of a boolean circuit as follows:

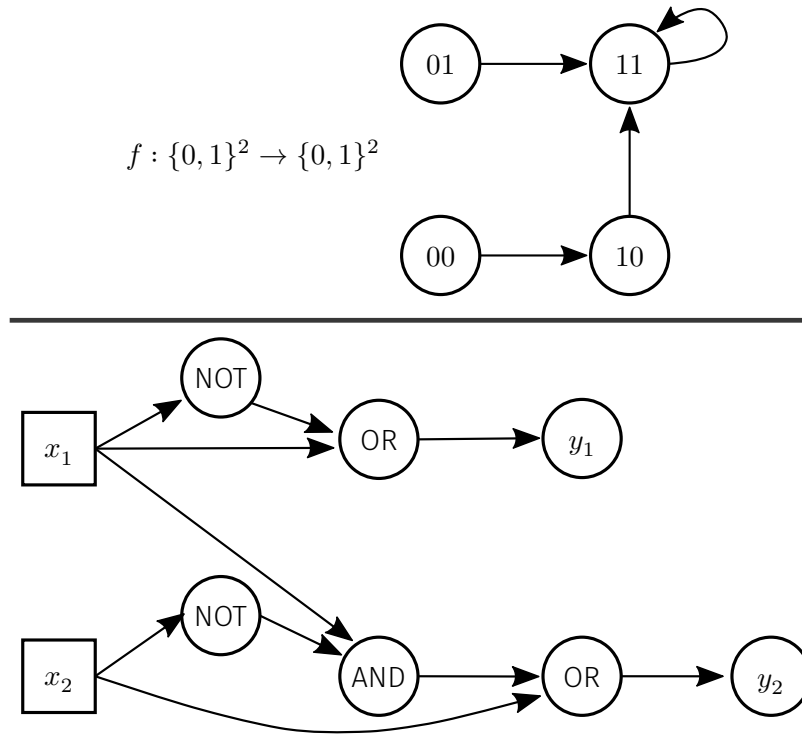
**Definition 2.9 — Depth of a boolean circuit.**

The depth of a boolean circuit  $C$  is the length of the longest path from an input gate to an output gate.

The depth of a boolean circuit is a measure of the time complexity of the computation, i.e. it tells us how many time steps are needed to compute the output of the function. This is especially true in a parallel setting, where all gates can be seen as setting off at the same time (exactly as in a CPU).

It can be shown that  $\text{poly}(\text{size}(n))$  bits suffice to encode any boolean circuit.





**Figure 2.2:** example of how a function  $f : \{0,1\}^2 \rightarrow \{0,1\}^2$  (on the top), can be computed using boolean circuits (on the bottom).

## 2.3 Subclasses of TFNP

Complete **FNP** problems within **TFNP** would imply that **NP** = **coNP** [12], a highly unlikely scenario. Consequently, complete problems are not expected within **TFNP**, necessitating alternative approaches to investigate its structure.

[12]: Megiddo et al. (1991), *On total functions, existence theorems and computational complexity*

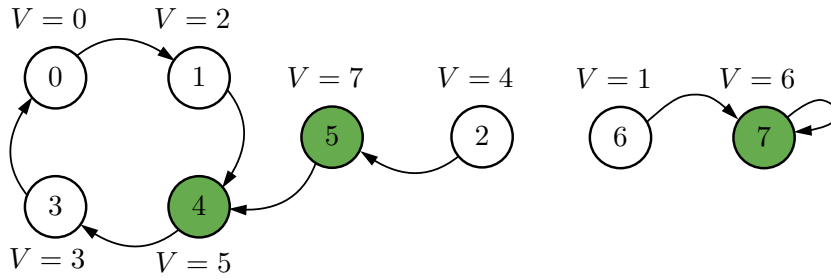
**TFNP** is characterized as a *semantic* class, which means it is difficult to verify whether a Turing machine defines a language within this class. In contrast, *syntactic* classes such as **P** and **NP** are characterized by the ease with which one can confirm that a Turing machine's accepted language belongs to the class. We refer the reader to Papadimitriou's work [13] for a more detailed discussion of these terms.

[13]: Papadimitriou (1994), *Computational complexity*

We want to explore syntactic subclasses of **TFNP** to address these challenges. One approach, proposed by Papadimitriou [13], categorizes search problems based on existence proofs confirming their totalness. This basic strategy leads to the detailed study of specific complexity classes discussed in the following sections.

### 2.3.1 Polynomial Local Search (PLS)

The existence result which gives rise to **PLS** is:



**Figure 2.3:** Example of a LOCALOPT Problem with  $n = 3$  (8 vertices). Solid lines represent the circuit  $S$ . The valid solutions are colored green.

*“Every directed acyclic graph has a sink.”*

We can then construct the class **PLS** by defining it as all problems which reduce to finding the sink of a directed acyclic graph (DAG). Formally we first define the problem LOCALOPT as in [2]:

**LOCALOPT**

**Input:** Two boolean circuits  $S, V : [2^n] \rightarrow [2^n]$ .

**Output:** A vertex  $v \in [2^n]$  such that  $P(S(v)) \geq P(v)$ .

[2]: Johnson et al. (1988), *How easy is local search?*

$S$  can be seen as a proposed successor, and  $V$  as a potential. The goal is to find a local minima  $v$  of the potential.

One might ask why this is equivalent to finding the sink of a DAG? The circuit  $S$  defines a directed graph, which might contain cycles. Only keeping the edges on which the potential decreases (strictly) leads to a DAG, with as sinks exactly the  $v$  such that  $P(S(v)) \geq P(v)$ . We give an example of a LOCALOPT instance in Figure 2.3. Now we can define **PLS**:

**Definition 2.10 — Polynomial Local Search (PLS).**

The class **PLS** is the set of all **TFNP** problems that reduce to LOCALOPT.

Studying “easy” problems such as PLS is particularly insightful because we strongly believe that these problems cannot be solved by any method more efficiently than simply traversing the graph.. Hence, given a graph of exponentially large size, it appears highly improbable that an efficient solution can be found. Therefore, all problems in **PLS** inherently embody the fundamental challenge of not being able to surpass the basic strategy of navigating through the directed acyclic graph. Of course — and here lies the difficulty of complexity theory — we cannot prove this statement; it could be that some very clever analysis of the boolean circuits could lead to an efficient algorithm for finding sinks of exponentially large directed acyclic graphs.

By “easy”, we mean that the problem can be solved by simply walking through the graph and checking whether every vertex is a local minimum.

### 2.3.2 Polynomial Parity Argument on Directed Graphs (PPAD)

Now we want to discuss the complexity class **PPAD**, introduced by Papadimitriou as one of the first syntactic subclasses of **TFNP** in [1]. The existence result giving rise to this class is:

*“If a directed graph has an unbalanced vertex, then it has at least one other unbalanced vertex.”*

**PPAD** can be defined using the problem **END-OF-LINE** as introduced in [8].

#### END-OF-LINE

**Input:** Boolean circuit  $S, P : \{0, 1\}^n \rightarrow \{0, 1\}^n$  such that  $P(0^n) = 0^n \neq S(0^n)$  ( $0^n$  is a source.)

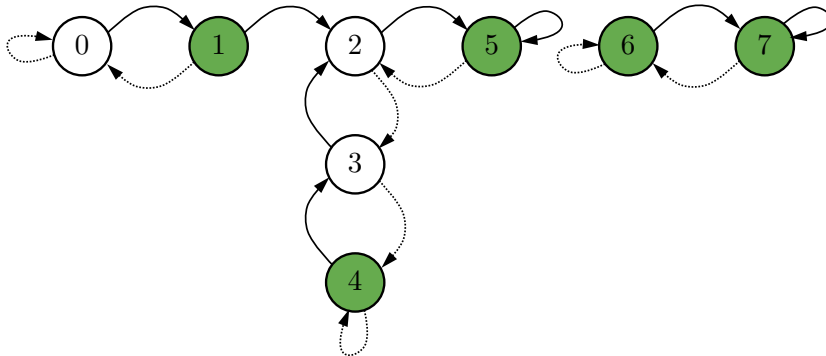
**Output:** An  $x \in \{0, 1\}^n$  such that either:

- ▶  $P(S(x)) \neq x$  ( $x$  is a sink) or
- ▶  $S(P(x)) \neq x \neq 0^n$  ( $x$  is a non non-standard source)

[1]: Papadimitriou (1994), *On the complexity of the parity argument and other inefficient proofs of existence*

[8]: Daskalakis et al. (2009), *The Complexity of Computing a Nash Equilibrium*

Here,  $S$  can be thought of as giving the successor of a vertex, and  $P$  as giving the predecessor of a vertex.



**Figure 2.4:** Example of an **END-OF-LINE** Problem with  $n = 3$  (8 vertices). Solid lines represent the circuit  $S$ , and dashed lines represent the circuit  $P$ . The solutions are the sinks  $x = 5$ ,  $x = 7$  and  $x = 1$ , as well as the sources  $x = 4$  and  $x = 6$ .

These boolean circuits represent a directed graph with maximal in and out-degree one by having an edge from  $x$  to  $y$  if and only if  $S(x) = y$  and  $P(y) = x$ . The goal is to find a sink in the graph or another source. It can be shown that the general case of finding a second imbalanced vertex in a directed graph (a problem called **IMBALANCE**) can be reduced to **END-OF-LINE** [14]. Now we can define the complexity class **PPAD** as follows:

#### Definition 2.11 — PPAD.

The class **PPAD** is the set of all **TFNP** problems that reduce to **END-OF-LINE**.

Notice that **END-OF-LINE** allows cycles and that these do not induce solutions.

[14]: Goldberg et al. (2021), *The Hairy Ball problem is PPAD-complete*

### 2.3.3 End of Potential Line (EOPL)

Next, we want to discuss the complexity class **EOPL** introduced in [3]. The existence results giving rise to **EOPL** is:

*“In a directed acyclic graph, there must be at least two unbalanced vertices.”*

[3]: Fearnley et al. (2018), *End of Potential Line*

Similarly to **PLS** acyclicity will be enforced using a potential.

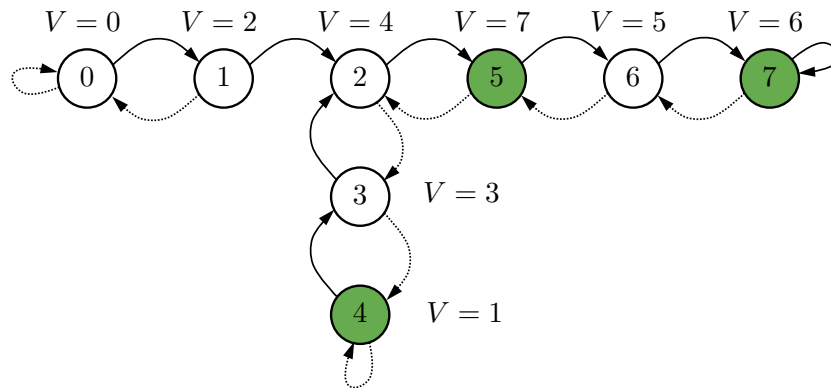
#### END OF POTENTIAL LINE

**Input:** Two boolean circuits  $S, P : \{0, 1\}^n \rightarrow \{0, 1\}^n$ , and a boolean circuit  $V : \{0, 1\}^n \rightarrow [2^n - 1]$ , such that  $0^n$  is a source, (i.e.  $P(0^n) = 0^n \neq S(0^n)$ ).

**Output:** An  $x \in \{0, 1\}^n$  such that either:

- ▶  $P(S(x)) \neq x$  ( $x$  is a sink)
- ▶  $S(P(x)) \neq x \neq 0^n$  ( $x$  is a non-standard source)
- ▶  $S(x) \neq x$ ,  $P(S(x)) = x$  and  $V(S(x)) \leq V(x)$  (violation of the monotonicity of the potential)

Here,  $S$  can be thought of as giving the successor of a vertex, and  $P$  as giving the predecessor of a vertex.  $V$  is a potential that is supposed to increase monotonously along the line.



**Figure 2.5:** Example of an **EOPL** Problem with  $n = 3$  (8 vertices). Solid lines represent the circuit  $S$ , and dashed lines represent the circuit  $P$ . The solutions are the sink  $x = 7$ , the violation of potential at  $x = 5$ , and the non-standard source  $x = 4$ .

$S$  and  $P$  can be thought of as representing a directed line. Finding another source (a non-standard source) is a violation, as a directed line only has one source. The potential serves as a guarantee of acyclicity. Now, we can define the complexity class **EOPL**.

#### Definition 2.12 — EOPL.

The class **EOPL** is the set of all **TFNP** problems that reduce to **END OF POTENTIAL LINE**.

## 2.4 The TARSKI Problem

### 2.4.1 Definition of the TARSKI Problem

Next, we want to introduce the TARSKI Problem. Before we do this, we recall that there is a partial order on the  $d$  dimensional lattice  $[N]^d$ , given by  $x \leq y$  if and only if  $x_i \leq y_i$  for all  $i \in \{1, \dots, d\}$ . We can now define functions on this lattice, and in particular, we can define monotone functions.

**Definition 2.13 — Monotone function.**

A function  $f : [N]^d \rightarrow [N]^d$  is *monotone* if for all  $x, y \in [N]^d$  we have  $x \leq y$  implies  $f(x) \leq f(y)$ .

Notice that  $x \not\leq y$  does not imply  $x \geq y$ . In particular, two points are not always comparable.

Such functions are also called *order preserving* functions in the literature.

The name of the TARSKI problem originates from Tarski's fixed point Theorem, first introduced in [4], which we remind the reader of below:

**Theorem 2.14 — Tarski's fixed point Theorem.**

Let  $f : [N]^d \rightarrow [N]^d$  a function on the  $d$ -dimensional lattice. If  $f$  is monotonous (for the previously discussed partial order), then  $f$  has a fixed point, i.e. there is an  $x \in [N]^d$  such that  $f(x) = x$ .

[4]: Tarski (1955), *A lattice-theoretical fixpoint theorem and its applications*.

This Theorem is also known as the Knaster–Tarski Theorem in the literature.

A proof of this Theorem can be found in the previously mentioned work [4]. Without surprise, the TARSKI problem, defined in [5], is now to find such a fixed point. Formally, we define the problem as follows:

[5]: Etessami et al. (2020), *Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria*

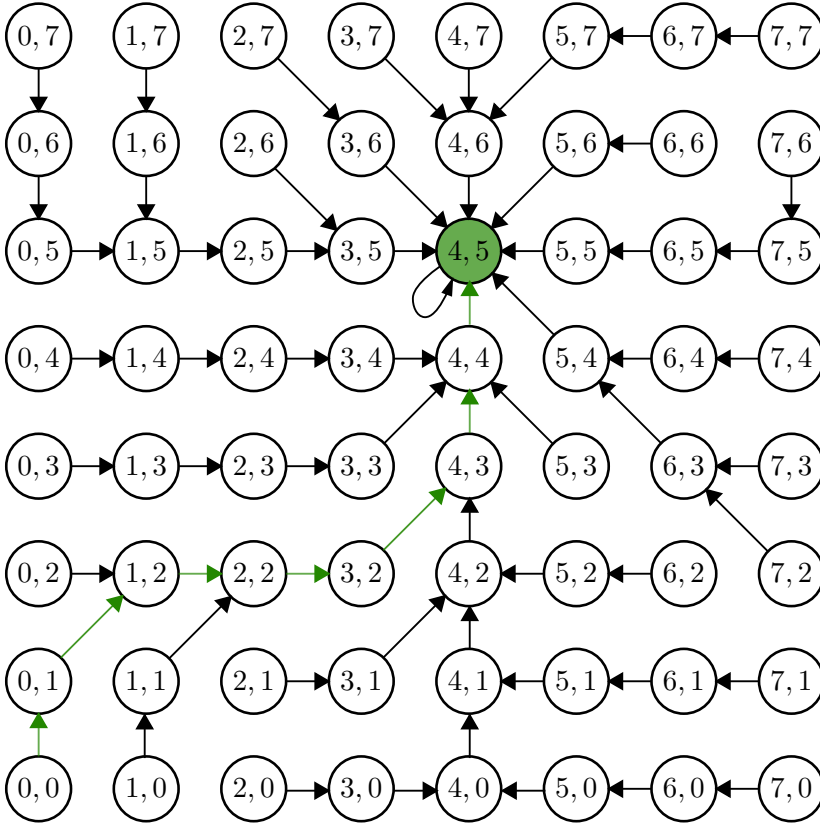
**TARSKI**

**Input:** A boolean circuit  $f : [N]^d \rightarrow [N]^d$ .

**Output:** Either:

- An  $x \in [N]^d$  such that  $f(x) = x$  (fixed point) or
- $x, y \in [N]^d$  such that  $x \leq y$  and  $f(x) \not\leq f(y)$  (violation of monotonicity).

This is a total search problem, as there will always either be a fixed point or a point violating monotonicity. We give an example of a two-dimensional TARSKI instance in Figure 2.6. Before we discuss the location of TARSKI in the TFNP landscape and two known algorithms for solving TARSKI, we want to discuss a useful Lemma, which allows us to simplify the study of Tarski instances. The definition of TARSKI instances allows for the image of a point to be located anywhere in the lattice; we will show that we can reduce to the cases where the image of a point is in the immediate neighborhood of the point.



**Figure 2.6:** Example of a 2 dimensional TARSKI instance. A fixed is located at  $x = (4, 5)$ . The path to the fixed point which Algorithm 1 finds is colored in green.

#### Lemma 2.15 – Simplifying TARSKI.

Let  $f : L \rightarrow L$  be a TARSKI instance on a complete lattice  $L$ . Consider  $\tilde{f} : L \rightarrow L$  given by:

$$\tilde{f}(x)[i] = \begin{cases} x[i] + 1 & \text{if } f(x)[i] > x[i], \\ x[i] & \text{if } f(x)[i] = x[i], \\ x[i] - 1 & \text{if } f(x)[i] < x[i]. \end{cases} \quad \text{for all } i \in \{1, \dots, d\}$$

Then, for any two points  $x, y \in L$ ,  $f(x) \leq f(y)$  if and only if  $\tilde{f}(x) \leq \tilde{f}(y)$ .

*Proof.* The lemma follows directly by observing that for all  $i \in \{1, \dots, d\}$  we have:  $f(x)[i] \leq f(y)[i]$  if and only if  $\tilde{f}(x)[i] \leq \tilde{f}(y)[i]$ .  $\square$

This means that in the whole thesis, we can consider the simplified version of the TARSKI problem, where for every  $x \in L$ , we have  $\|x - f(x)\|_\infty \leq 1$ , which we will implicitly assume from now on.

Notice that given a circuit  $C$  which computes  $f$ , we can construct a circuit  $\tilde{C}$  which computes  $\tilde{f}$  by adding  $\mathcal{O}(d)$  gates to  $C$ . This means that both problems are equivalent in terms of complexity.

### 2.4.2 Two algorithms for solving TARSKI

We briefly discuss the most common algorithms for solving TARSKI instances. We begin with a straightforward algorithm, which is based on the following observation:

**Remark 2.16**

Let  $f$  be a TARSKI instance on a complete lattice  $L$ . If  $f$  is monotonous, and for some  $x \in L$  we have  $f(x) \geq x$ , then  $f(f(x)) \geq f(x)$ .

Now, by noticing that by starting with the point  $\mathbf{0} = 0^d$  and iterating the function  $f$ , we will eventually reach a fixed point, we can construct an iterative Algorithm for solving TARSKI, described in Algorithm 1.

---

**Algorithm 1:** Iterative Algorithm for TARSKI

---

**Data:** A boolean circuit  $f : L \rightarrow L$

**Result:** A fixed point of  $f$

```

 $x \leftarrow \mathbf{0}$ ;
while  $f(x) \neq x$  do
  if  $f(x) \not\geq x$  then
    return " $x, f(x)$  are a violation of monotonicity.";
  else
     $x \leftarrow f(x)$ ;
return  $x$ ;

```

---

The path which Algorithm 1 takes to solve a TARSKI instance is colored green in Figure 2.6. While Algorithm 1 might not be very efficient — it runs in worst-case time  $\mathcal{O}(d \cdot N)$  for  $L = [N]^d$  — it does have some theoretical applications for locating TARSKI inside **TFNP**. Previous work [5] showed that TARSKI lies in **PLS** by considering the set of possible states of the previously described algorithm, together with a potential function given by  $V(x) = \sum_{i=1}^d x[i]$ , and showing that this potential is monotonous along the states of the algorithm. The circuit  $S$  associates to the state of the algorithm the next state it will be in.

Next, we want to describe a more advanced algorithm for solving TARSKI instances. The algorithm we will present is due to [15]. We will give an alternative presentation and simplified proof of the correctness of the algorithm here. Before this, we want to introduce some notations to make the argument as clear as possible. For a given complete lattice  $L = [N_1] \times \dots \times [N_d]$  and

[5]: Etessami et al. (2020), *Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria*

[15]: Dang et al. (2020), *Computations and Complexities of Tarski's Fixed Points and Supermodular Games*

some dimension  $x \in L$  we define the following sublattices:

$$L_{\leq x} = [x[1] + 1] \times \cdots \times [x[d] + 1],$$

$$L_{\geq x} = \llbracket x[1], N_1 \rrbracket \times \cdots \times \llbracket x[d], N_d \rrbracket$$

We denote by  $\llbracket a, b \rrbracket$  the set of whole numbers  $\{a, a + 1, \dots, b\}$ .

and for a given dimension  $k \in \{1, \dots, d\}$  and  $K \in [N_k]$ , we define the following sublattices:

$$L_{k < K} = [N_1] \times \cdots \times [N_{k-1}] \times [K] \times [N_{k+1}] \times \cdots \times [N_d],$$

$$L_{k=K} = [N_1] \times \cdots \times [N_{k-1}] \times \{K\} \times [N_{k+1}] \times \cdots \times [N_d],$$

$$L_{k > K} = [N_1] \times \cdots \times [N_{k-1}] \times \{K + 1, \dots, N_k\} \times [N_{k+1}] \times \cdots \times [N_d].$$

The algorithm — and in particular, our proof of the correctness — is based on the following observation:

**Remark 2.17**

Let  $L = [N_1] \times \cdots \times [N_d]$  be a complete lattice and  $f : L \rightarrow L$  a monotonous function. Then:

- (1) If for some  $x \in L$  we have  $f(x) \leq x$ , then  $f$  has a fixed point in  $L_{\leq x}$ .
- (2) If for some  $x \in L$  we have  $f(x) \geq x$ , then  $f$  has a fixed point in  $L_{\geq x}$ .

*Proof.* Let  $x \in L$  such that  $f(x) \leq x$ . Then for all  $y \in L_{\leq x}$  we have  $y \leq x$  and hence  $f(y) \leq f(x) \leq x$ , which shows that  $f$  is a TARSKI instance on  $L_{\leq x}$ . By Tarski's fixed point Theorem,  $f$  has a fixed point in  $L_{\leq x}$ . The proof for the second point is analogous.  $\square$

Hence, points with these properties seem particularly interesting when searching for fixed points of  $f$ . Hence, we want to give them a name:

**Definition 2.18 — Progress point.**

Let  $f : L \rightarrow L$  a TARSKI function. We call a point  $x \in L$  a *progress point* if  $f(x) \leq x$  or  $f(x) \geq x$ .

The lattice's smallest vertex and largest vertex are always progress points.

This means that if we have a progress point, we can reduce the area where we need to search for a fixed point. The question now becomes: how do we find such an  $x$ ? The algorithm we will present is based on the following observation:

**Remark 2.19**

Let  $f : L \rightarrow L$  on a complete lattice  $L = [N_1] \times \cdots \times [N_d]$ , for a monotonous function  $f$ , be a TARSKI instance. By fixing some dimension  $k \in \{1, \dots, d\}$ , we can define the function



$f_{k=K} : L_{k=K} \rightarrow L_{K=k}$  as follows:

$$f_{k=K}(x)[i] = \begin{cases} f(x)[i] & \text{if } i \neq k, \\ K & \text{if } i = k. \end{cases} \quad \text{for all } i \in \{1, \dots, d\}$$

Then  $f_{k=K}$  is a monotone TARSKI instance on  $L_{k=K}$ , and if  $x^*$  is a fixed point of  $f_{k=K}$ , then  $x^*$  is a progress point of  $f$ .

If we can solve a  $d - 1$  dimensional TARSKI instance, we can find a point  $x$  such that  $f(x) \geq x$  or  $f(x) \leq x$ .

*Proof.* The monotonicity of  $f_{k=K}$  follows directly from the monotonicity of  $f$ .

The fact that  $x^*$  is a progress point follows from the fact that if  $x^*$  is a fixed point of  $f_{k=K}$ , then  $f(x^*)[i] = x^*[i]$ , for all  $i \neq k$ . This means that if  $f(x^*)[k] \leq x^*[k]$ , then  $f(x^*) \leq x^*[k]$  and if  $f(x^*)[k] \geq x^*[k]$ , then  $f(x^*) \geq x^*[k]$ .  $\square$

By choosing  $K = \lfloor \frac{N_k}{2} \rfloor$  we can find a progress point  $x$  such that both  $L_{\leq x}$  and  $L_{\geq x}$  have at most half the size of  $L$ . This means we can reduce the search space by a factor of at least two by solving a  $d - 1$  dimensional TARSKI instance. We can solve a  $d$  dimensional TARSKI instance by repeatedly solving  $d - 1$  dimensional TARSKI instances and reducing the search space size by a factor of at least 2 in each step. This means we can solve a  $d$  dimensional TARSKI instance by combining a  $d - 1$  dimensional TARSKI solver and a binary search. The  $d - 1$  dimensional instances can be solved recursively. We give the recursive algorithm for solving TARSKI instances in Algorithm 2.

---

**Algorithm 2:** Recursive Algorithm for TARSKI

---

**Function** RecursiveTarskiSolver( $f : L \rightarrow L, d$ ):

```

/* Binary search in the d-th dimension */
l ← 0, r ← Nd;          /* The search space is [l, r] */
while r - l > 1 do
  m ← ⌊l+r/2⌋;          /* Middle of the interval */
  if d - 1 = 0 then
    ⊢ x* ← m
  else
    /* Solve the d-1 dimensional instance */
    ⊢ x* ← RecursiveTarskiSolver(fd=m, d - 1);
  if f(x*)[d] ≤ x*[d] then
    ⊢ r ← m
  else
    ⊢ l ← m
⊢ return x*
```

---

A simple analysis shows that this algorithm runs in  $\mathcal{O}(\log^d N)$  for  $L = [N]^d$ . It was conjectured by Etessami et al. that this is an optimal algorithm for TARSKI [5]. This turned out not to be true, as a better algorithm was developed, which mostly relies on a smarter way of finding progress points, and a smarter way of dividing the problem into sub-instances [16]. We will not discuss this algorithm in detail, as it is not relevant for the rest of the thesis, but we want to mention that it achieves a runtime of  $\mathcal{O}(\log^{2\lceil \frac{d}{3} \rceil} N)$  for  $L = [N]^d$ . This is, to date, the best upper bound for solving TARSKI instances.

[5]: Etessami et al. (2020), *Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria*

[16]: Fearnley et al. (2022), *A Faster Algorithm for Finding Tarski Fixed Points*

### 2.4.3 Lower bounds for TARSKI

We want to discuss the lower bounds for solving TARSKI instances. The best-known lower bounds for TARSKI are given by [5]. They showed that in the black-box model, where the only way to access the function  $f$  is by querying it, solving a  $d$ -dimensional TARSKI requires solving at least  $\Omega(\log^N)$  one-dimensional TARSKI instances, which are as difficult as binary search, hence this means that solving a  $d$ -dimensional TARSKI instance requires at least  $\Omega(\log^2 N)$  queries. This means that the upper and lower bounds are equal in the 2-dimensional case, but in all other cases, there remains a gap. In particular, the best-known lower bound for solving TARSKI does not depend on the dimension  $d$ , which seems somewhat unexpected.

[5]: Etessami et al. (2020), *Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria*

This gives us reason to study TARSKI under the lens of complexity theory, in particular to understand where TARSKI lies in the **TFNP** landscape.

### 2.4.4 Location of TARSKI in TFNP

We now want to summarize where TARSKI lies inside of **TFNP**. It has been shown in [5] that TARSKI lies in **PLS** as we discussed when presenting Algorithm 1. The same paper showed that TARSKI lies in  $\mathbf{P}^{\text{PPAD}}$ . We will provide an alternative proof of this second fact in Chapter 3. Previous work [17] showed that many-to-one reductions and Turing-reduction onto **PPAD** are equivalent. In particular this means that  $\mathbf{P}^{\text{PPAD}} = \text{PPAD}$ , and that TARSKI lies in **PPAD**.

[17]: Buss et al. (2012), *Propositional proofs and reductions between NP search problems*

Now that we have established that TARSKI lies inside  $\text{PLS} \cap \text{PPAD}$ , we want to discuss the structure of  $\text{PLS} \cap \text{PPAD}$  and describe recent advances in the study of this class. There have been two surprising advances in the study of  $\text{PLS} \cap \text{PPAD}$  in the last few years. The first is that  $\text{CLS} = \text{PLS} \cap \text{PPAD}$  [18]. **CLS**

[18]: Fearnley et al. (2023), *The Complexity of Gradient Descent: CLS = PPAD  $\cap$  PLS*

(Continuous Local Search) was first introduced by Daskalakis and Papadimitriou in [9] and can be informally thought of as the class of all problems that can be solved by finding the local optimum of a potential in a discrete space equipped with an adjacency relation. This result shows that the problems in  $\text{PLS} \cap \text{PPAD}$  are exactly those that gradient descent algorithms can solve.

[9]: Daskalakis et al. (2011), *Continuous Local Search*

A further notable collapse is the result  $\text{PLS} \cap \text{PPAD} = \text{EOPL}$ , which was only recently shown in [6]. This, of course, means that, in particular, TARSKI lies in **EOPL**. A question that then arises, and which this thesis will try to answer, is whether we can construct an explicit reduction of TARSKI to **ENDOFPOTENTIALLINE**.

[6]: Goos et al. (2022), *Further Collapses in TFNP*

# Reducing TARSKI to PPAD

This chapter explores the membership of TARSKI in the complexity class **PPAD**. We begin by presenting an established proof of the reduction of this problem to BROUWER [5], focusing on a high-level overview. Subsequently, we introduce a novel problem, TARSKI\*, which facilitates a divide and conquer approach to solving TARSKI by leveraging the structure of the function  $f$ . This new formulation allows us to provide an alternative proof of Tarski's membership in PPAD using *Sperner's Lemma* instead of the traditional *Brouwer's Fixed Point Theorem*. This approach simplifies the proof and sets the stage for further reduction of TARSKI\* to EOPL in the subsequent chapter.

3.1	Known reduction to	
	PPAD . . . . .	21
3.2	Introducing TARSKI* . .	22
3.3	Sperner's Lemma . . .	24
3.3.1	on Simplices . . . . .	25
3.3.2	on Lattices . . . . .	26
3.4	Reducing TARSKI* to	
	SPERNER . . . . .	27

[5]: Etessami et al. (2020), *Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria*

## 3.1 Presentation of the known reduction of TARSKI to PPAD

We want to give a high-level presentation of the proof of TARSKI membership in **PPAD** from [5], which will help us motivate the introduction of TARSKI\* and the subsequent use of *Sperner's Lemma*. The proof given by Etessami et al. relies on *Brouwer's fixed point theorem*, which we introduce below.

### Theorem 3.1 — Brouwer's fixed point theorem.

Let  $K \subset \mathbb{R}^d$  be a compact, convex set. Then every continuous function  $f : K \rightarrow K$  has a fixed point  $x^* \in K$ , i.e.  $f(x^*) = x^*$ .

The original proof can be found in [19], and a more straightforward proof relying on SPERNER'S LEMMA can be found in [20]. This Theorem gives rise to a total search problem which we call BROUWER:

#### BROUWER

**Input:** A continuous function  $f : K \rightarrow K$ .

**Output:** A fixed point  $x^* \in K$  such that  $f(x^*) = x^*$ .

The problem BROUWER was first introduced and shown to be **PPAD**-complete in [21], meaning that it suffices to reduce TARSKI to BROUWER in order to show that TARSKI is in **PPAD**. We will reduce TARSKI to, at most polynomially, many instances of BROUWER, which will allow us to show that TARSKI is in  $\mathbf{P}^{\text{PPAD}}$ . Overall, we will construct a Turing reduction of TARSKI to BROUWER, which suffice as **PPAD** is closed under Turing reductions [17].

[5]: Etessami et al. (2020), *Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria*

[19]: Brouwer (1911), *Über Abbildung von Mannigfaltigkeiten*

[20]: Aigner et al. (2018), *Proofs from THE BOOK*

We leave out the technical detail of how this function is given using boolean circuits and how precise the output needs to be, as it is irrelevant for this high-level presentation.

[21]: Papadimitriou (1994), *On the complexity of the parity argument and other inefficient proofs of existence*

[17]: Buss et al. (2012), *Propositional proofs and reductions between NP search problems*

The idea of the reduction is to extend the discrete function  $f$  to a function  $\tilde{f} : [0, 2^n - 1]^d \rightarrow [0, 2^n - 1]^d$ , such that  $\tilde{f}$  interpolates the lattice function  $f$ , is continuous and piecewise linear between lattice points, and hence continuous. The authors achieve this by using a simplicial decomposition of each cell of the lattice. Now we have an instance of BROUWER, and hence, we can find a fixed point  $x^*$  of  $\tilde{f}$ . Of course, this fixed point does not need to be *integral*. The critical insight is that we can use this fixed point to reduce the search area for an integral fixed point by at least half or find a violation of monotonicity. In particular, either there is a fixed point in both  $\{x \in [2^n]^d : x \geq x^*\}$  and  $\{x \in [2^n]^d : x \leq x^*\}$ , or there is a violation of monotonicity in the cell containing  $x^*$ . We can repeat this procedure, always halving the search area, which allows us to solve a TARSKI instance using at most  $\mathcal{O}(d \cdot n)$  calls to BROUWER.

We call a point *integral* if it belongs to the original lattice.

### 3.2 Introducing TARSKI\*

In the previous section, we have seen that TARSKI can be reduced to a polynomial number of BROUWER instances. We want to study a single such reduction to give an alternative proof that TARSKI is in PPAD. In order to do this, we introduce a new problem, TARSKI\*. This problem can be thought of as a subproblem towards solving TARSKI. A standard strategy to solve TARSKI is to use a *divide and conquer* strategy, for instance, used in [5], and presented previously in Algorithm 2, of Subsection 2.4.2. We want to construct a problem that allows us to divide the TARSKI problem into two smaller problems, where solving the smaller of the two leads to a solution.

[5]: Etessami et al. (2020), *Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria*

For the sake of generality and in order to achieve more precise proofs in the following, we introduce the problem on the integer lattice  $L = N_1 \times \dots \times N_d$ , such that  $N_i \leq 2^n$  for all  $i \in \{1, \dots, d\}$ . We propose the following problem:

#### TARSKI\*

**Input:** A boolean circuit  $f : L \rightarrow L$ .

**Output:** Either:

- Two points  $x, y \in L$  such that  $\|x - y\|_\infty \leq 1$ ,  $x \leq f(x)$  and  $y \geq f(y)$ , or;
- A violation of monotonicity: Two points  $x, y \in L$  such that  $x \leq y$  and  $f(x) \not\leq f(y)$ .

We want to show that TARSKI\* is, in a sense, a subproblem of TARSKI.

**Claim 3.2**

An instance of TARSKI can be solved using  $\mathcal{O}(d \cdot n)$  calls to TARSKI\* and up to  $\mathcal{O}(d)$  additional function evaluations.

*Proof.* We will show that we can use a single call of TARSKI\* to either find a violation of monotonicity, a fixpoint, or an instance of TARSKI which has at most half as many points and must contain a solution. Let  $x, y$  be the two points which a Turing machine solving TARSKI\* on a function  $f$  outputs. We proceed by case distinction:

**Case 1:** We are done if  $f(x) = x$  or  $f(y) = y$  because we have found a fixpoint.

**Case 2.1:** If  $x < y$  and  $f(x) \not\leq f(y)$ , we have a violation of monotonicity, which solves the given TARSKI instance.

**Case 2.2:** If  $x < y$  and  $f(x) \leq f(y)$ , we claim that we can solve the TARSKI instance in  $\mathcal{O}(\|x - y\|_1)$  additional function calls. Notice that we have  $\|x - y\|_\infty \leq 1$ . Now notice that because  $f(x) > x$  (if not, see case 1), there is at least one dimension  $i \in \{1, \dots, d\}$  such that  $f(x)[i] > x[i]$ . Also notice that in this dimension  $i$  if  $f(y)[i] < y[i]$ , then because  $|x[i] - y[i]| \leq \|x - y\|_\infty \leq 1$ , we would have a violation of the monotonicity of  $f$  in this dimension. Therefore, we must have  $f(y)[i] = y[i]$ . The same argument shows that if in any dimension  $j$  we have  $f(y)[j] < y[j]$ , then  $f(x)[j] = x[j]$ . Therefore, we know that because there must be at least one such dimension  $i$  and  $j$ , we have:

$$\|f(x) - f(y)\|_\infty \leq \|x - y\|_\infty \leq 1 \text{ and } \|f(x) - f(y)\|_1 \leq \|x - y\|_1 - 2$$

Hence, we can now repeat the same argumentation with  $f(x)$  and  $f(y)$ , and we can do this at most  $\mathcal{O}(\|x - y\|_1)$  times until we find a violation of monotonicity or a fixpoint. Because  $\|x - y\|_1 \leq d$ , this will take at most  $\mathcal{O}(d)$  additional steps.

**Case 3:** If  $x \not\leq y$ , then we can partition the set of lattice points into two sets  $S_x$  and  $S_y$ , as follows:

$$S_x = \{z \in L : z \geq x\} \quad \text{and} \quad S_y = \{z \in L : z \leq y\}.$$

These two sets are disjoint: if there was a  $z \in S_x \cap S_y$ , then  $x \leq z \leq y$ , which would imply  $x \leq y$ , which is a contradiction. We will show that  $S_x$  must contain a solution to the TARSKI instance. If for some  $z \in S_x$ , we have  $f(z) \notin S_x$ , then we have  $f(z) \not\leq f(x)$ , which means that  $z$  and  $x$  form a violation of monotonicity. This means that  $S_x$  forms a new valid instance of TARSKI. By the same argumentation,  $S_y$  also forms a valid instance of TARSKI and hence, it suffices to solve the smaller of the two instances recursively. In particular, because they are disjoint, one of the instances  $S_x$  or  $S_y$  contains less than half of the lattice points of  $L$ , and hence we can solve the instance in  $\mathcal{O}(\log 2^{dn}) = \mathcal{O}(d \cdot n)$  calls of TARSKI\*.

□

Now that we know that TARSKI\* is a good stepping stone towards solving TARSKI, we want to investigate why TARSKI\* lies in **PPAD**.

### 3.3 Sperner's Lemma

The preceding discussion hinges on the assumption that TARSKI\* is a total problem, implying that every instance of the problem is guaranteed a solution. This section will substantiate this claim, establishing Tarskistar's classification within **TFNP**. Rather than employing *Brouwer's fixed point Theorem* — a cornerstone of continuous topology — we pivot to its discrete analog, *Sperner's Lemma*, a foundational result in combinatorial topology. This approach is particularly apt for two main reasons:

- We are working on a discrete lattice, so it seems more natural to use a discrete tool.
- Papadimitriou proved that BROUWER is **PPAD**-complete by reducing BROUWER to SPERNER [21]. Hence, by reducing to BROUWER, we introduce continuity into the problem, which is unnecessary, as it gets removed again behind the scenes.

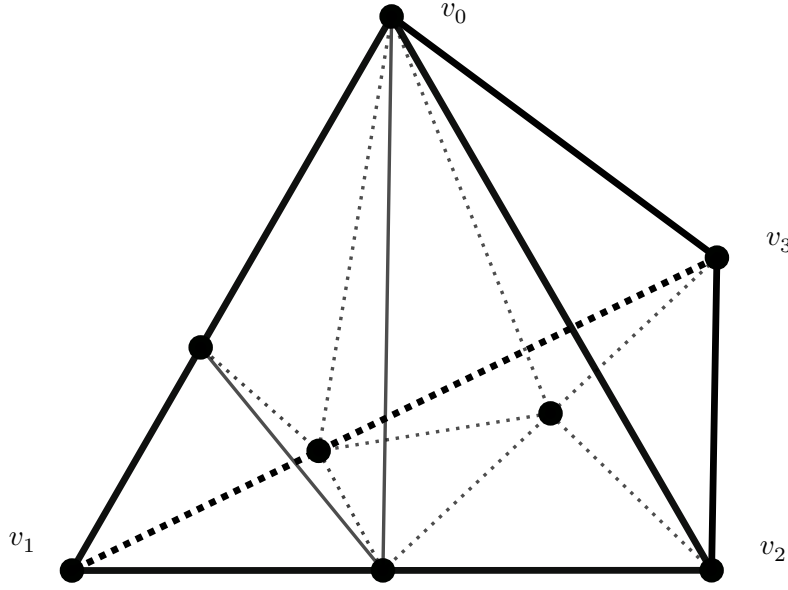
[21]: Papadimitriou (1994), *On the complexity of the parity argument and other inefficient proofs of existence*

We aim to apply *Sperner's Lemma* on the integer lattice. Using this tool is not directly possible, as *Sperner's Lemma* is defined on a simplicial decomposition of a simplex. Hence, we will first introduce *Sperner's Lemma* for simplices and then show how it can be adapted to work on an integer lattice.

### 3.3.1 Sperner's Lemma for Simplices

Before we introduce the Lemma itself, we want to define the setting of the result. We consider a  $d$ -dimensional simplex with vertices  $v_0, v_1, \dots, v_d$ . We now consider a *simplicial subdivision* of this simplex, meaning that we partition the simplex into smaller simplices. We give an example of such a partition in Figure 3.1 in the 3-dimensional case.

By  $d$  dimensional simplex we mean the convex Hull of these  $d+1$  points in  $\mathbb{R}^d$



**Figure 3.1:** Setup for SPERNER'S LEMMA in the 3-dimensional case. The large simplex spanned by  $v_0, v_1, v_2, v_3$  is subdivided into smaller simplices.

Now we introduce a coloring  $c$  of the vertices of this subdivision with colors  $\{0, 1, \dots, d\}$ . We want to enforce that the vertices  $v_i$  of the large simplex are colored with color  $i$  and that the vertices on a subsimplex  $\{v_{i_0}, \dots, v_{i_k}\}$  are colored with colors  $i_0, \dots, i_k$ . We give an example of such a coloring in 2 dimensions in Figure 3.2.

We now introduce Sperner's Lemma, which was first proven in [22], and for which a more modern proof can be found in [20].

#### Theorem 3.3 — Sperner's Lemma.

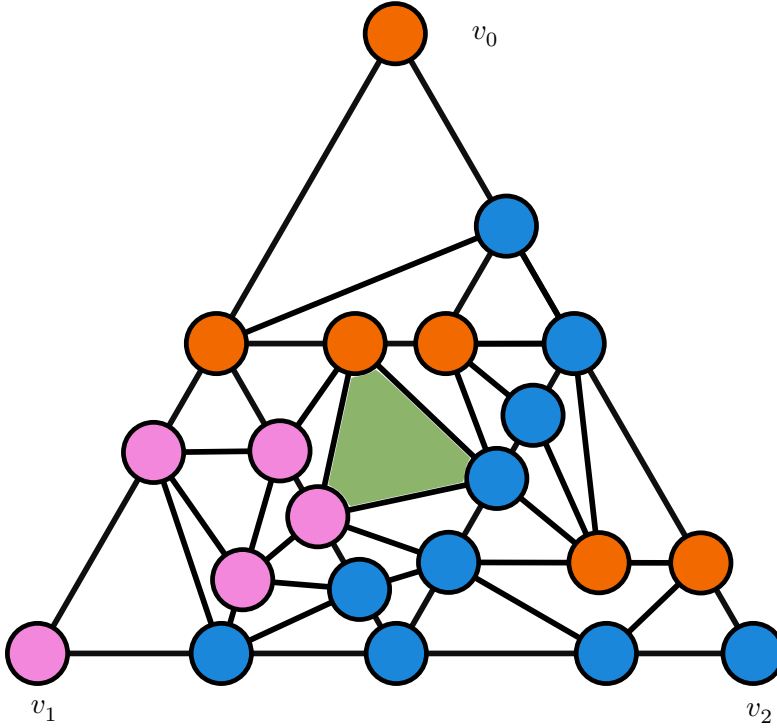
Suppose that a  $d$ -dimensional simplex with vertices  $v_0, \dots, v_d$  is subdivided into smaller simplices. Now color every vertex with a color  $\{0, \dots, d\}$  such that  $v_i$  is colored  $i$ , and the vertices on a subsimplex  $\{v_{i_0}, \dots, v_{i_k}\}$  are colored with colors  $i_0, \dots, i_k$ . Then, there is a subsimplex with vertices of every color.

[22]: Sperner (1928), *Neuer beweis für die invarianz der dimension-zahl und des gebietes*

[20]: Aigner et al. (2018), *Proofs from THE BOOK*

We give an example of a 2-dimensional simplex, subdivided into smaller simplices, and colored according to *Sperner's Lemma* in Figure 3.2.





**Figure 3.2:** Example of SPERNER'S LEMMA in the two-dimensional case, with three colors: orange (0), purple (1), and blue (2). The subsimplex spanned by  $v_0$  and  $v_1$  only contains blue and purple vertices, the subsimplex spanned by  $v_1$  and  $v_2$  contains only purple and blue vertices, and the subsimplex spanned by  $v_0$  and  $v_2$  contains only orange and blue vertices. *Sperner's Lemma* implies that there must be a subsimplex (colored in green) containing all colors.

### 3.3.2 Sperner's Lemma for an integer lattice

We have introduced *Sperner's Lemma* for an integer lattice. The motivation is to find a region of a colored lattice that contains all colors under certain conditions. Instead of a subsimplex, we will look for a *cell*<sup>1</sup> of the lattice, which contains all colors.

In order to do this, we will proceed as follows: We take the  $d$ -dimensional lattice  $L = [N_1] \times \dots \times [N_d]$ , we subdivide each cell into simplices<sup>2</sup>. We set  $v_0 = (0, \dots, 0)$ ,  $v_1 = (N_1 - 1, 0, \dots, 0)$ ,  $\dots$ ,  $v_d = (0, \dots, 0, N_d - 1)$ . We give an example of such a subdivision in the 3-dimensional case in Figure 3.3. Notice that we can deform the lattice and obtain an equivalent simplex and a simplicial decomposition of this simplex.

This means that under the appropriate conditions — which we will detail next — we can apply *Sperner's Lemma* to the lattice. Assume that we color all vertices of the lattice with colors  $\{0, \dots, d\}$ , such that  $v_i$  is colored  $i$ , and every vertex  $x$  with  $x[i] = 0$ , is *not* colored  $i$  for  $i \in \{1, \dots, d\}$ . Then, we can apply *Sperner's Lemma* to this simplicial decomposition of the lattice, and we will find a simplex that contains all colors. Of course, because every subsimplex is included in exactly one cell by construction, there must be a cell that contains all colors. This motivates the definition of the total problem SPERNER, which was introduced and shown to be PPAD-complete in [21]. We introduce

1: By cell, we mean a unit hypercube of the integer lattice

2: How this we do this is not relevant in this chapter but will be discussed in the next chapter.

[21]: Papadimitriou (1994), *On the complexity of the parity argument and other inefficient proofs of existence*

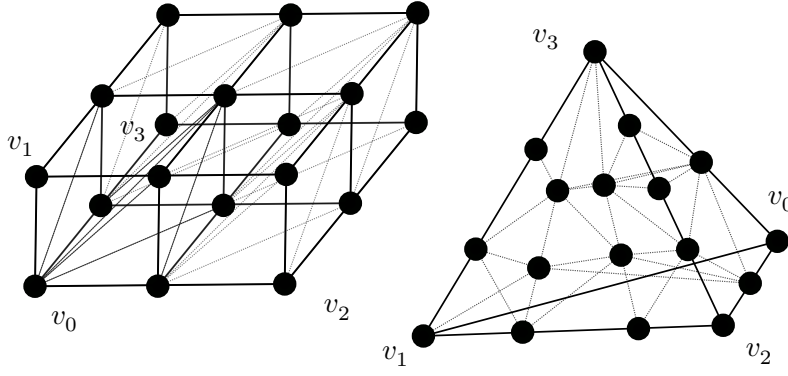


Figure 3.3: Example of the simplicial decomposition of a lattice in the three-dimensional case on the left, and the equivalent simplicial decomposition on the right of a simplex  $v_0, v_1, v_2, v_3$ .

the problem for a general lattice  $L = N_1 \times \dots \times N_d$ , such that  $N_i \leq 2^n$ .

#### SPERNER

**Input:** A coloring  $c : L \rightarrow \{0, \dots, d\}$  of the vertices of  $L$ , such that for every  $i \in \{0, \dots, d\}$  the vertices  $\{x \in L : x[i] = 0\}$  are not colored  $i$ .

**Output:** A cell  $C$  such that for all  $i \in \{0, \dots, d\}$  there is a vertex  $x \in C$  such that  $c(x) = i$ .

Next, we will use this problem to show that TARSKI\* is a total search problem and hence lies in PPAD.

### 3.4 Reducing TARSKI\* to SPERNER

For us to be able to use SPERNER'S Lemma on our TARSKI\* instances, we need to define a coloring of the vertices of  $L$ . We propose the following coloring  $c : L \rightarrow \{0, \dots, d\}$ :

$$c(x) = \begin{cases} 0 & \text{if } x \leq f(x) \\ 1 & \text{else if } x[1] > f(x)[1] \\ \vdots & \\ d & \text{else if } x[d] > f(x)[d] \end{cases}$$

A vertex colored 0 indicates that the function points *weakly forwards* in all dimensions, a vertex colored  $i$  for  $i \geq 1$  indicates that the function points *backwards* in at least the  $i$ -th dimension.

We now need two results. First, we need to show that a cell with all colors always exists, allowing us to show that TARSKI\* is a total search problem. Second, we need to show that finding a cell with all colors yields a solution to TARSKI\* in polynomial time.

#### Claim 3.4

For any TARSKI\* instance with vertices colored as above, there is always a cell with all colors.

*Proof.* This claim follows directly from SPERNER's Lemma and the coloring we have defined. There can never be a vertex colored  $i$  with  $x[i] = 0$  because this would imply that  $f(x)[i] < x[i]$ , which is a contradiction to the construction of the function. Hence, by dividing each cell of the lattice into simplices, we can apply SPERNER's Lemma to show that a cell with all colors always exists. The vertices we use as the vertices of the large simplex are  $\{(0, \dots, 0), (2^n - 1, 0, \dots, 0), \dots, (0, \dots, 2^n - 1)\}$ .  $\square$

### Claim 3.5

Finding a cell with all colors yields a solution to TARSKI\*, in  $\mathcal{O}(d)$  additional steps.

*Proof.* Assume we have found a simplex, with vertices colored  $0, \dots, d$ . Let us denote  $x_i$  the vertex colored  $i$ , for  $i \in \{0, \dots, d\}$ . Notice that all of these vertices are by construction contained in some cell (hypercube of length 1); let  $\mathbf{0}$  be the smallest vertex of this hypercube and  $\mathbf{1}$  the largest. In particular, this means that for all  $i$ , we have:

$$\mathbf{0} \leq x_i \leq \mathbf{1} \quad \text{and} \quad f(x_i)[i] < x_i[i] \quad \text{for } i > 0$$

We now proceed by case distinction:

**Case 1:** If  $x_0$  is a fixed point, then  $x = y = x_0$  is a solution to TARSKI\*.

**Case 2:** If  $x_0 \neq f(x_0)$  and  $x_0 = \mathbf{0}$ . Then there is an  $i$  such that  $f(x_0)[i] > x_0[i]$ , which means that  $f(x_0)[i] - x_0[i] \geq 1$ . At the same time we must have  $f(x_i)[i] < x_i[i]$  and  $x_0[i] - x_i[i] \leq 0$  because  $x_0 = \mathbf{0}$ , and hence  $x_i[i] - f(x_i)[i] \geq 1$ . Now we get:

$$\begin{aligned} f(x_0)[i] - f(x_i)[i] &= \underbrace{f(x_0)[i] - x_0[i]}_{\geq 1} + \underbrace{x_0[i] - x_i[i]}_{\geq 0} + \underbrace{x_i[i] - f(x_i)[i]}_{\geq 1} \\ f(x_0)[i] - f(x_i)[i] &\geq 2 \end{aligned}$$

This implies that  $f(x_0) \not\leq f(x_i)$ , and hence  $x_0, x_i$  are two points witnessing a violation of monotonicity of  $f$ , which form a solution to TARSKI\*.

**Case 3:** If  $x_0 \neq f(x_0)$  and  $x_0 \neq \mathbf{0}$ . We claim that either  $f(\mathbf{0}) \leq \mathbf{0}$ , or we have a violation of monotonicity. Assume for the sake of contradiction that there is an  $i$  such that  $f(\mathbf{0})[i] > \mathbf{0}[i]$ . Then we must have  $f(x_i)[i] < x_i[i]$  hence we get:  $f(\mathbf{0})[i] \not\leq f(x_i)[i]$ , which is a violation of monotonicity. This means that either we can return  $y = x_0$  and  $x = \mathbf{0}$  as a solution to TARSKI\*, or  $x_i$  and  $\mathbf{0}$  as a violation of monotonicity.

This shows we can solve a TARSKI\* instance in  $\mathcal{O}(d)$  additional steps.  $\square$

This shows that TARSKI\* is a total search problem and can be reduced to SPERNER. Hence, TARSKI\* lies in **PPAD**, and by using that  $\mathbf{P}^{\mathbf{PPAD}} = \mathbf{PPAD}$ , we have shown that TARSKI lies in **PPAD**, without relying on BROUWER.

In the previous chapter, we exhibited how one can demonstrate the membership of Tarski in **PPAD** through a reduction to Sperner. We now demonstrate that the same approach yields a reduction to **ENDOFPOTENTIALLINE**, which lies within **EOPL**. This will necessitate a more meticulous examination of the structure of a Tarski instance and the induced colouring of the lattice points. In order to achieve our objective, we must first construct a specific simplicial decomposition of the lattice. This will be done with the intention of obtaining certain useful properties. Ultimately, our goal is to demonstrate that for a monotone Tarski instance, the associated **EndOfLine** instance does not contain any cycles. This will prove sufficient to establish a reduction to **ENDOFPOTENTIALLINE**.

## 4.1 Choosing a simplicial decomposition of the lattice — Freudenthal’s Simplicial Decomposition

In the previous chapter, we left the choice of a specific simplicial decomposition of the lattice open, as it did not contribute to our reduction. In this chapter, we aim to be more precise in our approach by selecting a specific simplicial decomposition that will enable us to derive structural results. We begin by outlining the desired properties of our simplicial decomposition. The most fundamental property is that every simplex of the decomposition must be contained within a single cell of the lattice. This implies that we can limit our inquiry to the identification of a simplicial decomposition of a single  $d$ -dimensional hypercube of side-length 1. Additionally, it is important to note that our objective does not entail the introduction of any new vertices; instead, we seek a decomposition of the hypercube that can be expressed as a set of subsets of the hypercube’s vertices. Finally, we wish for the vertices of a given simplex be totally ordered with respect to the partial order defined in Section 2.4. This will allow us to argue that two vertices, inside a given simplex, are always comparable, and thus their images through  $f$  must also be comparable, which will be useful.

Such a decomposition exists, and is known in the literature as *Freudenthal’s simplicial decomposition* [23]. We will introduce it

4.1	Freudenthal’s Simplicial Decomposition . . . . .	30
4.2	Orientating the simplices . . . . .	34
4.2.1	Orienting a simplex . . . . .	35
4.2.2	Orienting a simplicial complex . . . . .	36
4.3	Sequences of simplices . . . . .	40
4.4	Properties of the coloring . . . . .	41
4.4.1	General properties of the coloring . . . . .	41
4.4.2	Properties of sequences of simplices . . . . .	42
4.5	No cycles in the <b>ENDOFLINE</b> instance . . . . .	44

[23]: Freudenthal (1942), *Simplizialzerlegungen von Beschränkter Flachheit*

in a combinatorial way here, and refer the reader to the original paper for a geometric construction of the same decomposition.

**Definition 4.1 — Freudenthal's Simplicial Decomposition.**

Consider a unit hypercube  $[0, 1]^d$  in  $\mathbb{R}^d$  and consider  $S_d$  the group of all permutations of the dimensions of the hypercube  $\{1, \dots, d\}$ . For every permutation  $\pi \in S_d$ , define the simplex  $S_\pi$  as the convex hull of the vertices:

$$\begin{aligned} v_0 &= (0, 0, \dots, 0) \\ v_1 &= v_0 + e_{\pi(1)} \\ v_2 &= v_1 + e_{\pi(2)} \\ &\vdots \\ v_d &= v_{d-1} + e_{\pi(d)} = (1, 1, \dots, 1) \end{aligned}$$

Here we will use the notation  $e_i$  to denote the  $i$ -th unit vector in  $\mathbb{R}^d$ .

The set of such simplexes  $\mathcal{S} = \{S_\pi : \pi \in S_d\}$  is Freudenthal's simplicial decomposition of the hypercube  $[0, 1]^d$ .

We want to begin by arguing why this decomposition is well-defined. We begin by showing that every point of the hypercube is contained in at least one simplex of  $\mathcal{S}$ .

**Lemma 4.2**

Let  $x = (x[1], \dots, x[d]) \in [0, 1]^d$ , let  $\pi \in S^d$  be the permutation such that  $x[\pi(1)] \leq x[\pi(2)] \leq \dots \leq x[\pi(d)]$ . Then  $x \in S_\pi$ .

*Proof.* We want to show that  $x$  is a convex combination of the vertices of  $S_\pi$ . We define the following sequence of real numbers:

$$\begin{aligned} \lambda_0 &= x[\pi(1)] \\ \lambda_1 &= x[\pi(2)] - x[\pi(1)] \\ \lambda_2 &= x[\pi(3)] - x[\pi(2)] \\ &\vdots \\ \lambda_{d-1} &= x[\pi(d)] - x[\pi(d-1)] \\ \lambda_d &= 1 - x[\pi(d)] \end{aligned}$$

Notice that we have  $\lambda_i \geq 0$  for all  $i$  and  $\sum_{i=0}^d \lambda_i = 1$ , by telescoping the sum. We can now write  $x$  as a convex combination

of the vertices of  $S_\pi$  as follows by noticing that  $v_i = \sum_{j=0}^i e_{\pi(j)}$ :

$$\begin{aligned} \sum_{i=0}^d \lambda_i v_i &= \sum_{i=0}^d \lambda_i \left( \sum_{j=0}^i e_{\pi(j)} \right) = \sum_{i=0}^d \sum_{j=1}^i \lambda_i e_{\pi(j)} \\ &= \sum_{j=1}^d \sum_{i=0}^j \lambda_i e_{\pi(j)} = \sum_{j=1}^d e_{\pi(j)} \sum_{i=0}^j \lambda_i = \sum_{j=1}^d e_{\pi(j)} x[\pi(j)] = x \end{aligned}$$

This shows that  $x$  is a convex combination of the vertices of  $S_\pi$ , and thus  $x \in S_\pi$ .  $\square$

Next we want to discuss why this really forms a partition of the hypercube. Of course a given point  $x$  can be contained in multiple simplexes, but we want to show that this does not happen apart from on the boundary of the simplices.

#### Lemma 4.3

Let  $S_\pi \in \mathcal{S}$  be a simplex. Then the *interior* of  $S_\pi$  is:

$$\text{int}(S_\pi) = \{x \in [0, 1]^d : 0 < x[\pi(1)] < x[\pi(2)] < \dots < x[\pi(d)] < 1\}$$

*Proof.* The same proof as for lemma 4.2, holds with the added constraint that all  $\lambda_i > 0$ , this then shows that these points are in the interior of the simplex.  $\square$

These two lemma's together show that we have a well-defined simplicial decomposition of the hypercube. We can now use this decomposition to prove some structural results about the lattice points of a TARSKI instance. We start by showing that this simplicial decomposition has the desired properties.

#### Lemma 4.4

Let  $S_\pi \in \mathcal{S}$  be a simplex. Then the vertices of  $S_\pi$  are totally ordered with respect to the partial order defined in Section 2.4. In particular we claim that:

$$v_0 < v_1 < v_2 < \dots < v_d$$

*Proof.* Because this relation is transitive it suffice to show that  $v_i < v_{i+1}$  for all  $i \in \{0, \dots, d-1\}$ . This follows immediately from the construction of the  $v_i$  as we have  $v_i[j] = v_{i+1}[j]$  for all  $j \neq \pi(i+1)$  and  $v_i[\pi(i+1)] = v_{i+1}[\pi(i+1)] - 1$ .  $\square$

This directly implies the following corollary.

#### Corollary 4.5

For two vertices  $x, y$  of any simplex  $S \in \mathcal{S}$ , if for any  $i \in$

$\{1, \dots, d\}$  we have  $x[i] < y[i]$ , then  $x < y$ . In particular  $x \not\leq y$  is equivalent to  $x > y$ .

Notice that this is not the case for any two points in the hypercube, as the partial order is not a total order. This is why choosing a simplicial decomposition with this property will be crucial in the following sections. Next we want to introduce a new notation which will allow us describe these simplices more succinctly. Assume that a permutation  $\pi$  of the dimensions, induces a simplex  $S_\pi$ , with vertices  $v_0, \dots, v_d$ , as defined in Definition 4.1. Then we will denote the  $d$ -dimensional simplex  $S_\pi$  as:

$$v_0 \xrightarrow{\pi(1)} v_1 \xrightarrow{\pi(2)} v_2 \xrightarrow{\pi(3)} \dots \xrightarrow{\pi(d)} v_d$$

This notation means that we obtain  $v_i$  by moving by one unit-length in the direction  $\pi(i)$  from  $v_{i-1}$ . We already briefly discussed how the faces of a given simplex are given. We will also describe how to describe these faces in our notation. We will denote the face of  $S_\pi$  obtained by removing the vertex  $v_i$  as:

$$v_0 \xrightarrow{\pi(1)} v_1 \xrightarrow{\pi(2)} \dots v_{i-1} \xrightarrow{\pi(i), \pi(i+1)} v_{i+1} \xrightarrow{\pi(i+2)} \dots \xrightarrow{\pi(d)} v_d$$

We can remark the following about the faces of a simplex.

**Remark 4.6**

For a given  $d - 1$  dimensional simplex  $F$  in  $\mathcal{S}$  we have that:

(1) If  $F$  is of the form:

$$F : v_0 \xrightarrow{\pi(1)} \dots v_{i-1} \xrightarrow{\pi(i), \pi(i+1)} v_{i+1} \xrightarrow{\pi(i+2)} \dots \xrightarrow{\pi(d)} v_d$$

Then  $F$  is a face of exactly two simplices  $S_1$  and  $S_2$ :

$$S_1 : v_0 \xrightarrow{\pi(1)} \dots v_{i-1} \xrightarrow{\pi(i+1)} w_i \xrightarrow{\pi(i)} v_i \xrightarrow{\pi(i+2)} \dots \xrightarrow{\pi(d)} v_d$$

$$S_2 : v_0 \xrightarrow{\pi(1)} \dots v_{i-1} \xrightarrow{\pi(i)} w'_i \xrightarrow{\pi(i+1)} v_i \xrightarrow{\pi(i+2)} \dots \xrightarrow{\pi(d)} v_d$$

(2) If  $F$  is of the form:

$$F : v_0 \xrightarrow{\pi(1)} v_1 \xrightarrow{\pi(2)} \dots \xrightarrow{\pi(d-1)} v_{d-1}$$

Notice that the case (1) is the case where the face is inside the cell, and the case (2) is the case where the face is on the border of the cell.



Then  $F$  is a face of exactly two simplices  $S_1$  and  $S_2$ :

$$\begin{aligned} S_1 : & \quad v_0 \xrightarrow{\pi(1)} v_1 \xrightarrow{\pi(2)} \cdots \xrightarrow{\pi(d-1)} v_{d-1} \xrightarrow{\pi(d)} w_d \\ S_2 : & \quad w_0 \xrightarrow{\pi(d)} v_0 \xrightarrow{\pi(1)} v_1 \xrightarrow{\pi(2)} \cdots \xrightarrow{\pi(d-1)} v_{d-1} \end{aligned}$$

We want to discuss what simplices of the decomposition neighbor each other. We claim that a given simplex has  $d - 1$  neighboring simplices inside a given cell, and two neighboring simplices in neighboring cells. More precisely we have the following lemma.

**Lemma 4.7 — Neighboring Simplices.**

Let  $S_\pi \in \mathcal{S}$  be a simplex:

$$v_0 \xrightarrow{\pi(1)} v_1 \xrightarrow{\pi(2)} v_2 \xrightarrow{\pi(3)} \cdots \xrightarrow{\pi(d)} v_d$$

Then the following simplices are neighbors of  $S_\pi$ :

- $v_0 \xrightarrow{\pi(2)} v_1 \xrightarrow{\pi(1)} \cdots v_{i-1} \xrightarrow{\pi(i+1)} w_i \xrightarrow{\pi(i+1)} v_i \xrightarrow{\pi(i+2)} \cdots \xrightarrow{\pi(d)} v_d$ , for all  $i \in \{1, \dots, d-1\}$ , where  $w_i$  is the vertex obtained by moving one unit in the direction  $\pi(i+1)$  from  $v_{i-1}$ .
- $w_d \xrightarrow{\pi(d)} v_0 \xrightarrow{\pi(1)} v_1 \xrightarrow{\pi(2)} \cdots \xrightarrow{\pi(d-1)} v_{d-1}$ , where  $w_d$  is the vertex obtained by moving one unit in the direction  $-\pi(d)$  from  $v_0$ .
- $v_2 \xrightarrow{\pi(2)} \cdots \xrightarrow{\pi(d-1)} v_{d-1} \xrightarrow{\pi(d)} v_d \xrightarrow{\pi(1)} w_1$ , where  $w_1$  is the vertex obtained by moving one unit in the direction  $\pi(1)$  from  $v_d$ .

*Proof.* The proof follows directly by enumerating the faces of  $S_\pi$  and using remark 4.6.  $\square$

## 4.2 Orientation of a the simplicial decomposition

In this section we want to discuss how to orient the simplicial decomposition of the lattice, we defined in the previous section. This will be important as we will argue in the next section, that the existence of a cycle would contradict the orientation of the simplicial decomposition. We will start by defining what we mean

by an orientation of a simplex and then discuss how to extend this to a general simplicial complex.

### 4.2.1 Orientation of a simplex

#### Definition 4.8 — Orientation of a simplex.

An *orientation* of a simplex  $S$  spanned by the vertices  $v_0, \dots, v_d$  is a choice of a permutation of the vertices  $[v_{\pi(0)}, \dots, v_{\pi(d)}]$ .

Notice that this leaves us with  $d!$  possible orientations of a simplex. Our notion of orientability should only lead to two possible classes of orientations, as an orientation of a 1-simplex is simply a choice of direction, and an orientation of a 2-simplex is a choice of a cyclic order of the vertices. Hence we want to define when two orientations are equivalent.

#### Definition 4.9 — Equivalent orientations.

Two orientations  $\pi$  and  $\sigma$  of a simplex  $S$  are *equivalent* if they differ by an even permutation. That is if  $\sigma = \pi \circ \tau$  for some permutation  $\tau$  with an even number of inversions.

In particular we give a more explicit definition of the equivalence of orientations of a 2-simplex, by relying on a total order  $\preceq$  of the vertices. We then get the following useful lemma:

#### Lemma 4.10

Two orientations  $\sigma, \tau$  of a simplex  $S$  are equivalent if and only if  $\text{sgn}(\sigma) = \text{sgn}(\tau)$ , with respect to the total order  $\preceq$ .

For a lattice this can be achieved by defining  $\preceq$  to be the lexicographic order of the vertices.

We would like to define the *opposite orientation* of a simplex, which should be an orientation which has the opposite sign with respect to the total order  $\preceq$ . This can be achieved by setting:

$$-[v_0, v_1, v_2, \dots, v_d] = [v_1, v_0, v_2, \dots, v_d]$$

We then have that the opposite orientation is not equivalent to the original orientation. This way we have a representative of both equivalence classes.

This means that we now have two equivalence classes of orientations for any simplex. We want to discuss how an orientation of a simplex extends to the faces of this simplex next. Notice that the faces of a simplex are themselves simplices, and thus have an orientation. Let  $[v_0, \dots, v_d]$  be an orientation of a simplex  $S$ . Now notice that every face can be obtained by removing one of the vertices  $v_j$  of  $S$ . Hence for every face  $F$ , the permutation  $[v_0, \dots, \hat{v}_j, \dots, v_d]$  is an orientation of  $F$ . But the

We use the notation  $\hat{v}_j$  to denote that  $v_j$  is missing.

orientation  $-[v_0, \dots, \hat{v}_j, \dots, v_d]$  is also a valid orientation of  $F$ . For reasons which will become apparent later we define the induced orientation of a face as follows:

**Definition 4.11 — Induced orientation of a face.**

Let  $\sigma = [v_0, \dots, v_d]$  be an orientation of a simplex  $S$ . The *induced orientation* of a face  $F$  of  $S$ , which is obtained by removing the vertex  $v_j$  from the vertex, is the orientation:

$$\sigma_j = (-1)^j \cdot [v_0, \dots, \hat{v}_j, \dots, v_d]$$

We claim that the induced orientations of faces, yields a consistent orientation of the simplex, that is that for every  $d-2$ -simplex  $E$  which is a face of two  $d-1$ -simplices  $S_1$  and  $S_2$ , the induced orientations of  $E$  in  $S_1$  and  $S_2$  are opposite.

**Claim 4.12**

Let  $F_1$  and  $F_2$  be two  $d-1$ -simplices in  $S$  which share a common face  $E$ . Then the induced orientations of  $E$  in  $S_1$  and  $S_2$  are opposite.

*Proof.* Let  $[v_0, \dots, v_d]$  be an orientation of  $S$ . The face  $E$  is obtained by removing two vertices  $v_i, v_j$  from  $S$ . Without loss of generality assume that  $F_1$  is obtained by removing  $v_i$  from  $S$  and  $F_2$  is obtained by removing  $v_j$  from  $S$ . Then the induced orientations  $S_1$  and  $S_2$  are:

$$\begin{aligned} S_1 : & \quad (-1)^i \cdot [v_0, \dots, \hat{v}_i, \dots, v_d] \\ S_2 : & \quad (-1)^j \cdot [v_0, \dots, \hat{v}_j, \dots, v_d] \end{aligned}$$

Now without loss of generality assume that  $i < j$ , then we have that the induced orientations of  $E$  in  $S_1$  and  $S_2$  are:

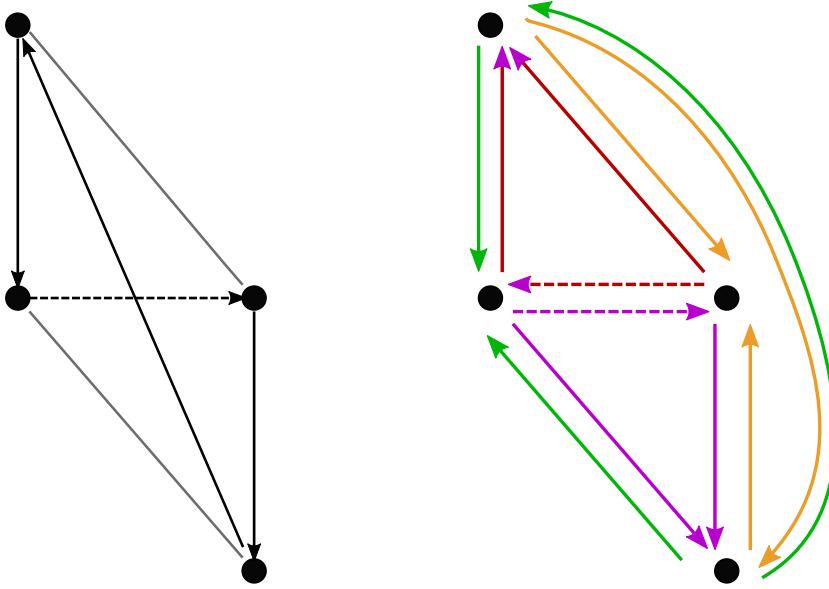
$$\begin{aligned} E \text{ in } S_1 : & \quad (-1)^i \cdot (-1)^{j-1} \cdot [v_0, \dots, \hat{v}_i, \dots, \hat{v}_j, \dots, v_{d-1}] \\ E \text{ in } S_2 : & \quad (-1)^j \cdot (-1)^i \cdot [v_0, \dots, \hat{v}_i, \dots, \hat{v}_j, \dots, v_{d-1}] \end{aligned}$$

This shows that the induced orientations of  $E$  in  $S_1$  and  $S_2$  are opposite.  $\square$

We give an example of the orientation of a 3-simplex and its faces in Figure 4.1. We can now discuss how we can extend this notion to a general simplicial complex.

## 4.2.2 Orientation of a simplicial complex

A simplicial complex can be thought of as a collection of simplices which are be glued together on their face. Our goal is now



**Figure 4.1:** Example of the orientation of a 3-simplex on the left, and the induced orientation of the faces on the right.

to extend this notion of orientation to these simplicial complexes. Formally we define a simplicial complex as follows [24]:

**Definition 4.13 — Simplicial complex.**

A simplicial complex  $\mathcal{K}$  in  $\mathbb{R}^d$  is a collection of simplices such that:

- (1) Every face of a simplex in  $\mathcal{K}$  is also in  $\mathcal{K}$ .
- (2) The intersection of any two simplices in  $\mathcal{K}$  is a face of both simplices.

Notice that the lattice points which we are interested in, together with Freudenthal's simplicial decomposition of each cell, form a simplicial complex. We now want to define an orientation of a simplicial complex. Of course such an orientation relies on an orientation of each simplex, and we want to make sure that these orientations are in some sense "compatible" on the faces of the simplicial complex. We will define this notion in the following definition.

**Definition 4.14 — Orientation of a simplicial complex.**

An *orientation* of a simplicial complex  $\mathcal{K}$  is a choice of an orientation of every  $d$ -simplex in  $\mathcal{K}$ , such that for every intersection of two simplices  $S_1, S_2 \in \mathcal{K}$ , the induced orientation of the face  $F = S_1 \cap S_2$  in  $S_1$  and  $S_2$  are opposite.

If such an orientation exists, we say that the simplicial complex is *orientable*.

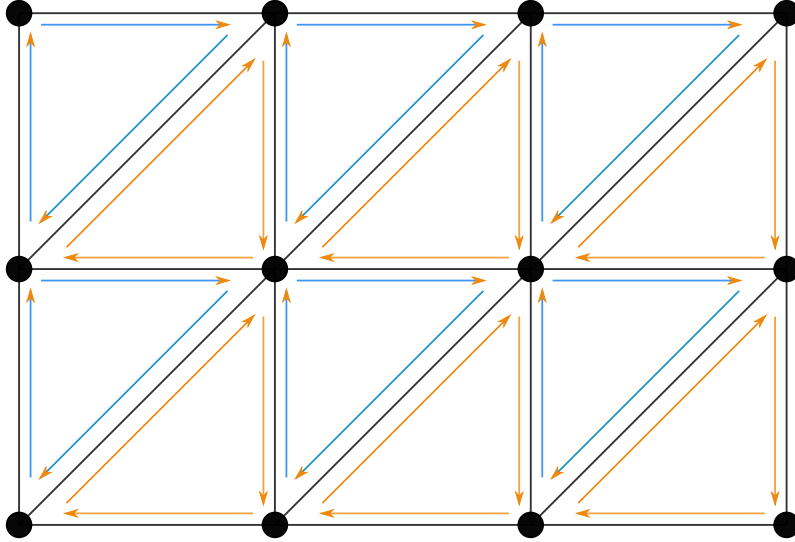
We now claim that the simplicial complex formed by the lattice points and Freudenthal's simplicial decomposition is orientable. This will be crucial in the next section, where we will argue that

[24]: Munkres (2018), *Elements of algebraic topology*

the existence of a cycle in the ENDOFLINE instance would contradict the orientation of the simplicial complex. In particular this shows that a Mobius Strip or the higher dimensional equivalents do not exist in our simplicial complex.

**Claim 4.15**

The simplicial complex formed by the lattice points and Freudenthal's simplicial decomposition is orientable.



**Figure 4.2:** Example of the orientation of a Freundenthal's simplicial complex in 2 dimensions.

*Proof.* We will give an orientation of every  $d$  simplex, and then show that the induced orientation of the faces of the simplicial complex are opposite. Let  $\pi \in S^d$  be a permutation of the dimensions, and  $v_0 \in L$  a vertex of the lattice. We then obtain a simplex  $S_\pi \in \mathcal{S}$  as previously described:

$$v_0 \xrightarrow{\pi(1)} v_1 \xrightarrow{\pi(2)} v_2 \xrightarrow{\pi(3)} \dots \xrightarrow{\pi(d)} v_d$$

We now orient  $S_\pi$  using the permutation:

$$\sigma = \text{sgn}(\pi) \cdot [v_0, \dots, v_d]$$

First we notice that for all  $d - 2$  simplices, two neighboring  $d - 1$ -simplices are contained in exactly one  $d$  simplex of the decomposition, and hence the orientation is consistent, as discussed in claim 4.12.

Now let us look at a common face  $F$  of two  $d$ -simplices  $S_1$  and  $S_2$ . We proceed by case distinction:

**Case 1:** Assume that  $S_1$  and  $S_2$  are in the same cell, then  $F$  is of the form:

$$F : v_0 \xrightarrow{\pi(1)} \cdots v_{i-1} \xrightarrow{\pi(i), \pi(i+1)} v_{i+1} \xrightarrow{\pi(i+1)} v_i \xrightarrow{\pi(i+2)} \cdots \xrightarrow{\pi(d)} v_d$$

And we have that  $S_1$  and  $S_2$  are of the form:

$$\begin{aligned} S_1 : v_0 &\xrightarrow{\pi(1)} \cdots v_{i-1} \xrightarrow{\pi(i+1)} w_i \xrightarrow{\pi(i)} v_i \xrightarrow{\pi(i+2)} \cdots \xrightarrow{\pi(d)} v_d \\ S_2 : v_0 &\xrightarrow{\pi(1)} \cdots v_{i-1} \xrightarrow{\pi(i)} w'_i \xrightarrow{\pi(i+1)} v_i \xrightarrow{\pi(i+2)} \cdots \xrightarrow{\pi(d)} v_d \end{aligned}$$

We immediately notice that  $\text{sgn}(S_1) = -\text{sgn}(S_2)$ . We remove a vertex  $w_i, w'_i$  of the same rank in  $S_1$  and  $S_2$  in order to obtain  $F$ . Hence the induced orientation of  $F$  in  $S_1$  and  $S_2$  are opposite.

By abuse of notation we will denote by  $\text{sgn}(S_1)$  the sign of the permutation inducing  $S_1$ .

**Case 2:** Next assume that  $S_1$  and  $S_2$  are in neighboring cells, then as discussed in Remark 4.6  $F$  is of the form:

$$F : v_0 \xrightarrow{\pi(1)} v_1 \xrightarrow{\pi(2)} \cdots \xrightarrow{\pi(d-1)} v_{d-1}$$

And we have that  $S_1$  and  $S_2$  are of the form:

$$\begin{aligned} S_1 : v_0 &\xrightarrow{\pi(1)} v_1 \xrightarrow{\pi(2)} \cdots \xrightarrow{\pi(d-1)} v_{d-1} \xrightarrow{\pi(d)} w_d \\ S_2 : w_0 &\xrightarrow{\pi(d)} v_0 \xrightarrow{\pi(1)} v_1 \xrightarrow{\pi(2)} \cdots \xrightarrow{\pi(d-1)} v_{d-1} \end{aligned}$$

We once again must proceed by case distinction.

**Case 2.1:** If  $d$  is even, then:  $\text{sgn}(S_1) = -\text{sgn}(S_2)$ , and we remove a vertex of rank  $d$  in  $S_1$  and of rank 0 in  $S_2$  to obtain  $F$ . We have  $(-1)^d = (-1)^0 = 1$  and hence the induced orientation of  $F$  in  $S_1$  and  $S_2$  are opposite.

**Case 2.2:** If  $d$  is odd, then  $\text{sgn}(S_1) = \text{sgn}(S_2)$ , and we remove a vertex of rank  $d$  in  $S_1$  and of rank 0 in  $S_2$  to obtain  $F$ . We have  $(-1)^d = -1$  and  $(-1)^0 = 1$  and hence the induced orientation of  $F$  in  $S_1$  and  $S_2$  are opposite.

This shows that the simplicial complex formed by the lattice points and Freudenthal's simplicial decomposition is orientable.  $\square$

We give an example of such an orientation in Figure 4.2. We have now introduced the necessary tools to argue that a certain type of cycle cannot exist as we will argue in the following.

### 4.3 Sequences of simplices

In this section we want to introduce and study *sequences of simplices*. We will show that they have some nice properties in regard to the orientation of the simplicial decomposition which we previously discussed. This will be useful as we will argue that paths in the ENDOFLINE instance are sequences of simplices, and that the orientation of the simplicial decomposition will prevent these paths from forming cycles. We will start by defining what we mean by a sequence of simplices.

**Definition 4.16 — Sequence of simplices.**

A *sequence of simplices*, or *simplicial sequence* is a sequence  $(S_i)_{i=1}^k$  of  $d$ -dimensional simplices  $S_i \in \mathcal{S}$  such that:

- (1)  $S_{i+1} \not\subset \{S_1, \dots, S_i\}$  for all  $i \in \{1, \dots, k-1\}$ .
- (2)  $S_i$  and  $S_{i+1}$  share a  $d-1$ -dimensional face  $F_i$  for all  $i \in \{1, \dots, k-1\}$ .

Observe that because of the orientation of the simplicial complex, the orientation of the faces of the simplices in a sequence are consistent. We want to show that this consistent orientation also extends to faces of the simplices in the sequence. In order to do this we need to introduce what we mean by *sequence of faces*. We will then show that the orientation of the simplicial complex implies that the orientation of the faces of the simplices in a sequence are consistent.

**Definition 4.17 — Sequence of faces.**

A *sequence of faces* of a simplicial sequence  $(S_i)_{i=1}^k$  is a sequence  $(L_i)_{i=1}^k$  of simplices such that:

- (1)  $L_i$  is a subsimplex of the simplices  $S_i$  for all  $i \in \{1, \dots, k\}$ .
- (2)  $L_i$  and  $L_{i+1}$  share a  $d-2$ -dimensional face  $G_i$  for all  $i \in \{1, \dots, k-1\}$ .

Notice that the face  $L_i$  can be of dimension  $d-1$  or  $d-2$ , in this definition.

We now want to show that such a sequence of face is consistent with the orientation of the simplicial complex.

**Proposition 4.18 — Orientation of sequences of faces.**

Let  $(S_i)_{i=1}^k$  be a simplicial sequence, and  $(L_i)_{i=1}^k$  a sequence of faces of the simplices in the sequence. Then the orientation of the faces  $L_i$  is consistent.

*Proof.* It suffices to show that for any two faces  $L_i$  and  $L_j$ , the induced orientation of  $Q = L_i \cap L_j$  in  $L_i$  and  $L_j$  are opposite. We proceed by case distinction:

**Case 1:** Assume that  $L_i$  and  $L_j$  are faces of the same simplex  $S$ , then  $Q$  is a face of  $S$  and the induced orientation of  $Q$  in  $L_i$  and  $L_j$  are opposite by Claim 4.12.

**Case 2:** Assume that  $L_i$  and  $L_j$  are faces of two simplices  $S_1$  and  $S_2$  which share a common face  $F$ . Then  $Q$  is a face of  $F$  and the induced orientation of  $Q$  in  $L_i$  and  $L_j$  are opposite by Claim 4.12.

This shows that the orientation of the faces of the simplices in a sequence is consistent.  $\square$

## 4.4 Properties of the coloring of TARSKI instances

In this section we want to discuss different properties with the coloring of TARSKI instances have. This will be helpful in arguing that the resulting ENDOFLINE instance does not contain any cycles. We will start with general properties and then move on to properties of sequences of simplices.

### 4.4.1 General properties of the coloring

In this section we will assume that we are working on a integer lattice  $L$ , and that for a function  $f : L \rightarrow L$ , the points have been colored  $c : L \rightarrow \{0, \dots, d\}$  as in Section 3.4. Now we are ready to present a first observation, which will be a helpful stepping stone for more advanced results.

Recall that the coloring was given by:

$$c(x) = \begin{cases} 0 & \text{if } x \leq f(x) \\ 1 & \text{else if } x[1] > f(x)[1] \\ \vdots & \\ d & \text{else if } x[d] > f(x)[d] \end{cases}$$

#### Lemma 4.19

Assume that  $f$  is monotone and that we have  $x_i, x_j \in L$ ,  $c(x_i) = i$  and  $c(x_j) = j$  for  $i, j \in \{1, \dots, d\}$  and  $x_i[i] = x_j[i]$ , then either:

- (1)  $i \geq j$  or
- (2)  $i < j$  and  $x_i \not\leq x_j$

*Proof.* Assume that  $i < j$  and  $x_i \geq x_j$ . We must then have  $f(x_j)[i] \geq x_j[i] = x_i[i] > f(x_i)[i]$ . Now by monotonicity of  $f$  we must have  $f(x_i) \geq f(x_j)$ , which is not possible if  $f(x_j)[i] > f(x_i)[i]$ . Hence we must have  $x_i \not\leq x_j$ . This shows that the lemma holds.  $\square$

For vertices of a given simplex we get the following corollary.

Notice that if we assume that  $x_i$  and  $x_j$  are in the same simplex of the simplicial decomposition, then the condition  $x_i \not\leq x_j$  is equivalent to  $x_i \leq x_j$ .



**Corollary 4.20**

Assume that  $f$  is monotone and that we have  $x_i, x_j \in S$ , for some simplex  $S \in \mathcal{S}$ . Further assume that  $c(x_i) = i$  and  $c(x_j) = j$  for  $i, j \in \{1, \dots, d\}$  with  $i < j$  and that  $x_i[i] = x_j[i]$ , then  $x_i < x_j$ .

*Proof.*  $x_i \leq x_j$ , follows immediately. Because  $x_i$  and  $x_j$  are colored differently, they can not be equal which shows the strict inequality.  $\square$

**4.4.2 Properties of sequences of simplices**

Now we want to work with sequences of simplices, and show that the coloring of the vertices of these simplices have some nice properties. We start by defining what we mean by a sequence of simplices. Let  $C \subset \{0, \dots, d\}$  be a subset of colors.

**Definition 4.21 — Valid sequence of simplices.**

A *valid sequence of simplices for colors  $C$*  is a sequence  $(S_i)_{i=1}^k$  of simplices as defined previously in Definition 4.16 such that the  $F_i$  are colored exactly colors of  $C$ .

Notice that this means that the first and last simplex of the sequence could be colored with any color. All other simplices must be colored with colors in  $C$ .

These sequences are the objects that latter get reduced to paths in the ENDOFLINE instance, which is why we want to study them in detail. We define the some more terminology to help us with this.

**Definition 4.22 — Cycle.**

A *cycle of simplices for colors  $C$*  is a valid sequence  $(S_i)_{i=1}^k$  of simplices  $S_i \in \mathcal{S}$  for colors  $C$  such that  $S_{k+1} = S_1$ .

Note that the empty sequence, and all sequences consisting of a single simplex are cycles.

**Definition 4.23 — Maximal sequence.**

A *maximal sequence of simplices for colors  $C$*  is a valid sequence  $(S_i)_{i=1}^k$  of simplices  $S_i \in \mathcal{S}$  for colors  $C$  such that:

- (1) There is no simplex  $S_{k+1} \in \mathcal{S}$  such that  $(S_i)_{i=1}^{k+1}$  is a valid sequence.
- (2) There is no simplex  $S_0 \in \mathcal{S}$  such that  $(S_i)_{i=0}^k$  is a valid sequence.

Intuitively we say that a sequence is maximal if we cannot make it longer by adding simplices at the beginning or end.

Finally we want to define the sequence of all transitions between simplices in a sequence.

**Definition 4.24 — Transition sequence.**

The *transition sequence* of a valid sequence  $(S_i)_{i=1}^k$  of sim-

plices  $S_i \in \mathcal{S}$  is the sequence  $(F_i)_{i=1}^{k-1}$  of  $(d-1)$ -dimensional faces  $F_i = S_i \cap S_{i+1}$ .

We now are ready to study the properties of these sequence in more detail. We now restrict ourselves to the case where  $C \subset \{0, \dots, d\}$  contains exactly  $d$  colors (i.e. only one color is left out). Notice that for a valid sequence  $(S_i)_{i=1}^k$  we then have that the transition sequence  $(F_i)_{i=1}^{k-1}$  is a sequence of  $(d-1)$ -dimensional simplices  $S_i$  which are colored with all  $d$  colors of  $C$ . This means that for every  $j \in C$  we get a sequence of vertices  $(x_i^j)_{i=1}^k$  such that  $x_i^j \in F_i$  and  $c(x_i^j) = j$ . We will now study this special case in more detail.

#### Lemma 4.25

Let  $S_i, F_i$  and  $x_j$  be as above. For any  $i \in \{1, \dots, k-1\}$  there is exactly one  $j \in C$  such that we have  $x_i^j \neq x_{i+1}^j$ .

*Proof.*  $F_i$  and  $F_{i+1}$  are two faces of the same  $d$  dimensional simplex, and thus they share exactly  $d-1$  vertices. This means that there is exactly one vertex  $x$  which is in  $F_i$  but not in  $F_{i+1}$ , and exactly one vertex  $y$  which is in  $F_{i+1}$  but not in  $F_i$ . This means that there is exactly one  $j$  such that  $x_i^j = x$  and  $x_{i+1}^j = y$ .  $\square$

Now for a valid sequence  $(S_i)_{i=1}^k$  of simplices, inside the color set  $C = \{0, \dots, d-1\}$  we can consider all the vertices that are not colored with the color 0. These vertices have a nice structure as the following proposition shows.

#### Proposition 4.26

Let  $(S_i)_{i=1}^k$  be a valid sequence of simplices for colors  $C = \{0, \dots, d-1\}$ , then defining  $L_i$  to be the face of  $S_i$  which is spanned by the vertices colored with colors in  $C \setminus \{0\}$ . Then the sequence  $(L_i)_{i=1}^k$  is a sequence of faces as defined in Definition 4.17.

*Proof.* We need to show that the two conditions set by Definition 4.17 are satisfied. The first condition is immediate as  $L_i$  is a face of  $S_i$ . For the second condition we need to show that  $L_i$  and  $L_{i+1}$  share a  $d-2$ -dimensional face for every  $i \in \{1, \dots, k-1\}$ . In order to see this notice that  $S_i$  and  $S_{i+1}$ , share a common face  $F_i$ , which contains exactly one vertex colored with color 0. This means that  $L_i$  and  $L_{i+1}$  share a  $d-2$ -dimensional face of  $F_i$ , and hence a  $d-2$  dimensional face of  $S_i$  and  $S_{i+1}$ .  $\square$

As a direct consequence of this proposition we get the following corollary, which will be a key tool in the discussion on cycles in the ENDOFLINE instance.

**Corollary 4.27**

The  $(L_i)_{i=1}^k$  defined above can be oriented consistently.

*Proof.* This is a direct consequence of Proposition 4.18, which showed that the orientation of the faces of a sequence of simplices is consistent, when using the induced orientation of the faces by the individual simplices.  $\square$

## 4.5 No cycles in the ENDOFLINE instance

We are now ready to show that the ENDOFLINE instance does not contain any cycles. We will do this by showing that the existence of a cycle would contradict the orientation of the simplicial complex. A cycle is a valid sequence such that  $S_0$  and  $S_k$  are a  $(d-1)$ -simplex as a face. We will show that certain situations cannot occur due to the orientation of the simplicial complex. And we will then argue that these situations must occur in a cycle. This will be enough to show that the ENDOFLINE instance does not contain any cycles.

Before we start we want to make an observation on how the dimension  $d$  plays together with the orientation of the simplicial complex.

**Lemma 4.28**

Let  $l \in \{1, \dots, d-1\}$  be a dimension. Let  $S$  be a  $d$ -simplex with colors  $C = \{0, \dots, d-1\}$  in the colored simplicial complex, such that  $S$  is of the form:

$$(S) : \quad v_0 \xrightarrow{l} v_1 \rightarrow \dots \rightarrow v_d$$

and assume that the face  $F$  spanned by  $v_1, \dots, v_d$  is a rainbow face. Then we must have for the colors:

$$(F) : \quad c(v_1) \rightarrow \dots \xrightarrow{d} \dots 0 \rightarrow \dots \rightarrow c(v_d)$$

*Proof.* Every color  $c \in \{0, \dots, d-1\}$  appears exactly once in the face  $F$ . If the color  $c \neq 0$  appear after 0, then by Corollary 4.20 we must have that we move in dimension  $c$  between 0 and  $c$ :

$$(F) : \quad c(v_1) \rightarrow \dots 0 \rightarrow \dots \xrightarrow{c} \dots \rightarrow c$$

Because we have this for every color  $c_i \neq 0$ , which appears after 0 in  $F$  we must have:

$$(F) : \quad c(v_1) \rightarrow \dots 0 \xrightarrow{c_1} c_1 \xrightarrow{c_2} c_2 \dots \xrightarrow{c_k} c_k$$

Now it is clear that because no vertex is colored with  $d$ , we must have that the change in dimension  $d$  occurs before the vertex colored 0 appears. This shows that we must have:

$$(F) : \quad c(v_1) \rightarrow \dots \xrightarrow{d} \dots 0 \rightarrow \dots \rightarrow c(v_d)$$

This shows the Lemma.  $\square$

We now want to show that a sequence of colored simplices as defined previously, can only cross a given hyperplane, given by fixing a dimension  $l$ , at most once. This will be a key tool in the discussion on cycles in the ENDOFLINE instance. Formally we have the following proposition.

**Proposition 4.29**

Let  $l \in \{1, \dots, d-1\}$  be a dimension. Consider the hyperplane  $H$  given by fixing  $l = L$  for some  $L \in [N]$ . Then consider the sequence of simplices  $(S_i)_{i=1}^k$  such as defined previously. Then there can not be two  $i \neq j$  such that  $F_i \subset H$  and  $F_j \subset H$ .

In other words  $H$  is given by  $H = \{x \in \mathbb{R}^d \mid x[l] = L\}$

Before we prove this we want to detail why this is a very powerful result. This means that in all dimensions apart from  $d$  the sequences of simplices which induce the paths in the ENDOFLINE instance are monotone. Now let us prove this result.

*Proof.* For the sake of contradiction assume that there are two such  $i \neq j$ , such that  $F_i \subset H$  and  $F_j \subset H$ . Without loss of generality we can further assume that  $i < j$  and that for all  $k \in \{i+1, \dots, j-1\}$  we have  $F_k \not\subset H$ , if so then replace  $j$  with the smallest such  $k$ .

Now notice that  $S_i$  and  $S_j$  are on opposite sides of  $H$ , and that  $S_{i+1}$  and  $S_j$  are on the same side of  $H$ . Now let us consider the sequence of colored simplices  $S_{i+1}, \dots, S_j$ . Notice that Lemma 4.28, which we proved earlier, implies that both  $F_i$ , and  $F_j$  must be of the following form:

$$(F_i) : \quad c(v_1) \rightarrow \dots \xrightarrow{d} \dots 0 \rightarrow \dots \rightarrow c(v_d)$$

$$(F_j) : \quad c(w_1) \rightarrow \dots \xrightarrow{d} \dots 0 \rightarrow \dots \rightarrow c(w_d)$$

We will now proceed by case distinction:

**Case 1:** Assume that  $F_i$  and  $F_j$  are comparable. We want to show that this is not possible. Assume without loss of generality that  $F_i$  is smaller than  $F_j$ . Then the vertex colored 0 in  $F_i$  must be smaller than the vertex colored  $l$  in  $F_j$ . This leads to a violation of monotonicity according to Corollary 4.20.

By *comparable* we mean that the cell containing  $F_i$  is either larger or smaller than the cell containing  $F_j$ .

**Case 2:** Assume that  $F_i$  and  $F_j$  are not comparable. Our goal is to show that in this case we do not have a consistent orientation of the faces  $L_i$  and  $L_{j-1}$ .

□

# APPENDIX

# Bibliography

- [1] Christos H. Papadimitriou. 'On the complexity of the parity argument and other inefficient proofs of existence'. In: *Journal of Computer and System Sciences* 48.3 (June 1994), pp. 498–532. doi: [10.1016/S0022-0000\(95\)80063-7](#). (Visited on 03/05/2024) (cited on pages 2, 5, 12).
- [2] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. 'How easy is local search?' In: *Journal of Computer and System Sciences* 37.1 (Aug. 1988), pp. 79–100. doi: [10.1016/0022-0000\(88\)90046-3](#). (Visited on 03/06/2024) (cited on pages 2, 11).
- [3] John Fearnley et al. *End of Potential Line*. Apr. 18, 2018. URL: <http://arxiv.org/abs/1804.03450> (visited on 03/02/2024) (cited on pages 2, 13).
- [4] Alfred Tarski. 'A lattice-theoretical fixpoint theorem and its applications.' In: *Pacific Journal of Mathematics* 5.2 (Jan. 1, 1955), pp. 285–309 (cited on pages 2, 14).
- [5] Kousha Etessami et al. 'Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria'. In: (2020). In collab. with Thomas Vidick. doi: [10.4230/LIPICS.ITCS.2020.18](#). (Visited on 02/24/2024) (cited on pages 2, 14, 16, 19, 21, 22).
- [6] Mika Goos et al. 'Further Collapses in TFNP'. In: (2022) (cited on pages 3, 20).
- [7] Sanjeev Arora and Boaz Barak. *Computational complexity : a modern approach*. Cambridge: Cambridge University Press, 2009 (cited on page 4).
- [8] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. 'The Complexity of Computing a Nash Equilibrium'. In: *SIAM Journal on Computing* 39.1 (Jan. 2009), pp. 195–259. doi: [10.1137/070699652](#). (Visited on 03/11/2024) (cited on pages 5, 12).
- [9] Constantinos Daskalakis and Christos Papadimitriou. 'Continuous Local Search'. In: *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*. Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, Jan. 23, 2011, pp. 790–804. doi: [10.1137/1.9781611973082.62](#). (Visited on 03/12/2024) (cited on pages 5, 20).
- [10] Alexandros Hollender. 'Structural Results for Total Search Complexity Classes with Applications to Game Theory and Optimisation'. PhD thesis. Oxford University, 2021 (cited on page 6).
- [11] Raymond Greenlaw and H. James Hoover. 'Chapter 9 - Circuit Complexity'. In: *Fundamentals of the Theory of Computation: Principles and Practice*. Ed. by Raymond Greenlaw and H. James Hoover. Oxford: Morgan Kaufmann, Jan. 1, 1998, pp. 241–257. doi: [10.1016/B978-1-55860-547-3.50013-3](#) (cited on page 7).
- [12] Nimrod Megiddo and Christos H. Papadimitriou. 'On total functions, existence theorems and computational complexity'. In: *Theoretical Computer Science* 81.2 (Apr. 1991), pp. 317–324. doi: [10.1016/0304-3975\(91\)90200-L](#). (Visited on 03/05/2024) (cited on page 10).
- [13] Christos H. Papadimitriou. *Computational complexity*. Reading (Mass): Addison-Wesley, 1994 (cited on page 10).

- [14] Paul W. Goldberg and Alexandros Hollender. ‘The Hairy Ball problem is PPAD-complete’. In: *Journal of Computer and System Sciences* 122 (Dec. 2021), pp. 34–62. DOI: [10.1016/j.jcss.2021.05.004](https://doi.org/10.1016/j.jcss.2021.05.004). (Visited on 03/10/2024) (cited on page 12).
- [15] Chuangyin Dang, Qi Qi, and Yinyu Ye. *Computations and Complexities of Tarski’s Fixed Points and Supermodular Games*. May 19, 2020. URL: <http://arxiv.org/abs/2005.09836> (visited on 07/21/2024) (cited on page 16).
- [16] John Fearnley, Dömötör Pálvölgyi, and Rahul Savani. ‘A Faster Algorithm for Finding Tarski Fixed Points’. In: *ACM Transactions on Algorithms* 18.3 (July 31, 2022), pp. 1–23. DOI: [10.1145/3524044](https://doi.org/10.1145/3524044). (Visited on 07/21/2024) (cited on page 19).
- [17] Samuel R. Buss and Alan S. Johnson. ‘Propositional proofs and reductions between NP search problems’. In: *Annals of Pure and Applied Logic* 163.9 (Sept. 2012), pp. 1163–1182. DOI: [10.1016/j.apal.2012.01.015](https://doi.org/10.1016/j.apal.2012.01.015). (Visited on 02/24/2024) (cited on pages 19, 21).
- [18] John Fearnley et al. ‘The Complexity of Gradient Descent:  $\text{CLS} = \text{PPAD} \cap \text{PLS}$ ’. In: *Journal of the ACM* 70.1 (Feb. 28, 2023), pp. 1–74. DOI: [10.1145/3568163](https://doi.org/10.1145/3568163). (Visited on 07/21/2024) (cited on page 19).
- [19] L. E. J. Brouwer. ‘Über Abbildung von Mannigfaltigkeiten’. In: *Mathematische Annalen* 71.1 (Mar. 1911), pp. 97–115. DOI: [10.1007/BF01456931](https://doi.org/10.1007/BF01456931). (Visited on 05/04/2024) (cited on page 21).
- [20] Martin Aigner and Günter M. Ziegler. *Proofs from THE BOOK*. 6th ed. 2018. Berlin, Heidelberg: Springer Berlin Heidelberg : Imprint: Springer, 2018. 1 p. (cited on pages 21, 25).
- [21] Christos H. Papadimitriou. ‘On the complexity of the parity argument and other inefficient proofs of existence’. In: *Journal of Computer and System Sciences* 48.3 (June 1994), pp. 498–532. DOI: [10.1016/S0022-0000\(05\)80063-7](https://doi.org/10.1016/S0022-0000(05)80063-7). (Visited on 03/22/2024) (cited on pages 21, 24, 26).
- [22] E. Sperner. ‘Neuer beweis für die invarianz der dimensionszahl und des gebietes’. In: *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* 6.1 (Dec. 1928), pp. 265–272. DOI: [10.1007/BF02940617](https://doi.org/10.1007/BF02940617). (Visited on 04/18/2024) (cited on page 25).
- [23] Hans Freudenthal. ‘Simplizialzerlegungen von Beschränkter Flachheit’. In: *The Annals of Mathematics* 43.3 (July 1942), p. 580. DOI: [10.2307/1968813](https://doi.org/10.2307/1968813). (Visited on 03/21/2024) (cited on page 30).
- [24] James Raymond Munkres. *Elements of algebraic topology*. The advanced book program. Boca Raton London New York: CRC Press, 2018. 454 pp. (cited on page 37).



# Alphabetical Index

- EOPL, 13
- PPAD, 12
- SPERNER, 27
- TARSKI\*, 22
  
- AND-gates, 8
  
- Boolean circuit, 8
- Brouwer, 21
- Brouwer's fixed point theorem, 21
  
- cell, 26
- Computed function of a boolean circuit, 8
  
- Cycle, 42
  
- decision problems, 4
- Depth of a boolean circuit, 9
  
- End of Potential Line, 13
- End-of-Line, 12
- equivalent, 35
- Equivalent orientations, 35
  
- Freudenthal's Simplicial Decomposition, 31
  
- Gate, 8
- gates, 7
  
- Induced orientation of a face, 36
  
- inputs gates, 8
- instance, 4
- integral, 22
- interior, 32
  
- language, 4
- Localopt, 11
  
- Many-to-one Reduction, 5
- Maximal sequence, 42
- monotone, 14
- Monotone function, 14
  
- Neighboring Simplices, 34
  
- non-standard source, 13
- NOT-gates, 8
  
- opposite orientation, 35
- OR-gates, 8
- order preserving, 14
- orientable, 37
- orientation, 35, 37
- Orientation of a simplex, 35
- Orientation of a simplicial complex, 37
- Orientation of sequences of faces, 40
- output gates, 8
  
- Polynomial Local Search (PLS), 11
- Progress point, 17
- promise problems, 6
  
- Search Problem, 4
- search problem, 4
- search problems, 4
- semantic, 10
- Sequence of faces, 40
- Sequence of simplices, 40
  
- Simplicial complex, 37
- simplicial complex, 37
- simplicial sequence, 40
- simplicial subdivision, 25
  
- Simplyfying TARSKI, 15
- Size of a boolean circuit, 9
  
- solution, 4
- Sperner's Lemma, 25
- syntactic, 10
  
- Tarski, 14
- Tarski's fixed point Theorem, 14
- total search problem, 5
- Total search problems, 5
- Transition sequence, 42
- Turing Reduction, 5
- type, 8
  
- unit vector, 31
  
- Valid sequence of simplices, 42
- violations, 6
  
- wires, 7