The Complexity of Finding Tarski Fixed Points

Master Thesis

July 21, 2024

Nils Jensen

Advised by

Prof. Dr. Bernd Gärtner

Sebastian Haslebacher

ETH zürich

Abstract

Insert the abstract here.

Contents

List of Figures

List of Tables

1 Introduction

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.

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

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.

1.2 The TFNP landscape

The class of TFNP, is the pendant of NP, in the sence 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.

TODO: Write some inuition about the subclasses.

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[1]. 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 monoticity of this function. By Tarski's theorem, this problem is guaranteed to have a solution, and hence is a total search problem.

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

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 PPAD$\cap$PLS $=$ EOPL [2]. 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.

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

1.6 Thesis Outline

TODO: Write this section.

Preliminaries 2

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 and the complexity class **TFNP**, along with its subclasses **PLS**, **PPAD**, and **EOPL**. In addition, this chapter will describe how functions and sets are represented in this framework, and how their complexity is measured. Finally, a formal introduction to the Tarski problem is given.

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. However, real-world questions often require explicit answers, and not simply existence results. For example, while deciding whether a function has a global minimum is a decision problem, the practical interest lies in actually identifying that minimum, which goes beyond mere existence.

This is where 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.

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

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

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

Here we have simply rephrased the valid language to be the pair of a problem instance and a valid solutions.

This leads us to ask classical complexity questions about search problems: Are these problems in **P**? in **NP**? Are they **NP**-hard? It is readily apparent that search problems are inherently 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 the decision component is removed from the equation? This scenario can be achieved by ensuring that “no” is never a valid solution. Such problems, where every instance is garanteed 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 [3] is simply the class of all total search problems that lie in NP. Examples of TFNP problems are:

[3]: 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.

- Factoring, the problem of finding the prime factors of a number. Every number admits a factorisation into prime numbers, and this factorisation can be checked in polynomial time.
- Nash, the problem of finding a nash equilibrium in a bimatrix game;
- Minimize, the problem of finding the global minimum of a convex function.

2.1.2 Reductions

Similarly to decision problem we can also define reduction inside TFNP.

Definition 2.3 — Many-to-one Reduction.
For two problem $R, S \in$ TFNP, we say that R *reduces* (many to one) to S if there exist polynomial time computable functions $f : \{0,1\}^* \to \{0,1\}^*$ and $g : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ such that for $I, s \in \{0,1\}^*$: if $(f(I), s) \in S$ 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

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

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

Definition 2.4 — Turing Reduction.
For two problems $R, S \in$ TFNP, we say that R *Turing reduces* to S if polynomial-time oracle Turing machine that solves R given access to an oracle for S exists.

2.1.3 Promise Problems

TODO: Talk about promise problems.

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". This means that we should describe 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 $[2^n - 1]$. 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 $\{0, 1\}^n$. For notational convenience we will often only denote the integer $x \in [2^n - 1]$ instead of the binary string.

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

2.2.2 Representation of functions

Now that we have described how we describe 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 as described in [4], and refer an interested reader to this source for a more detailed description.

[4]: 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 $g : \{0,1\}^k \to \{0,1\}$, 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: $g(x_1, x_2) = x_1 \land x_2$,
- OR-gate: $g(x_1, x_2) = x_1 \lor x_2$,
- 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 finute directed acyclic graph, where each vertex has a *type* τ, with

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

and with the follwing 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 such a boolean circuit in Figure 2.1. Of course we now want to use a boolean circuit to represent a function. In order to do this we need give a formal definition of the function computed by a boolean circuit.

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 $f : \{0,1\}^n \to \{0,1\}^m$ 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.

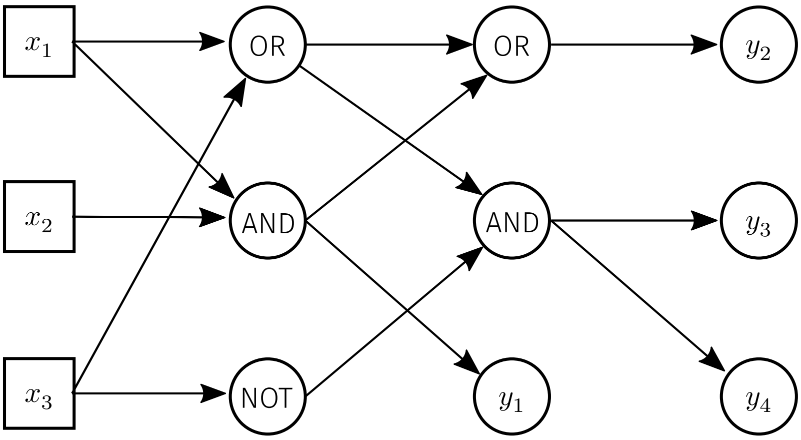

Figure 2.1: Example of a boolean circuit with three input gates and four output gates.

In Figure 2.2 we give an example of using a boolean circuit to compute a function, in particular for a function which is a Tarski instance. From now on all functions used in problems will be formally represented 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 that 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:

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

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 espacially true in a parallel setting, where all gates can be seen as setting off at the same time (exactly as in a CPU).

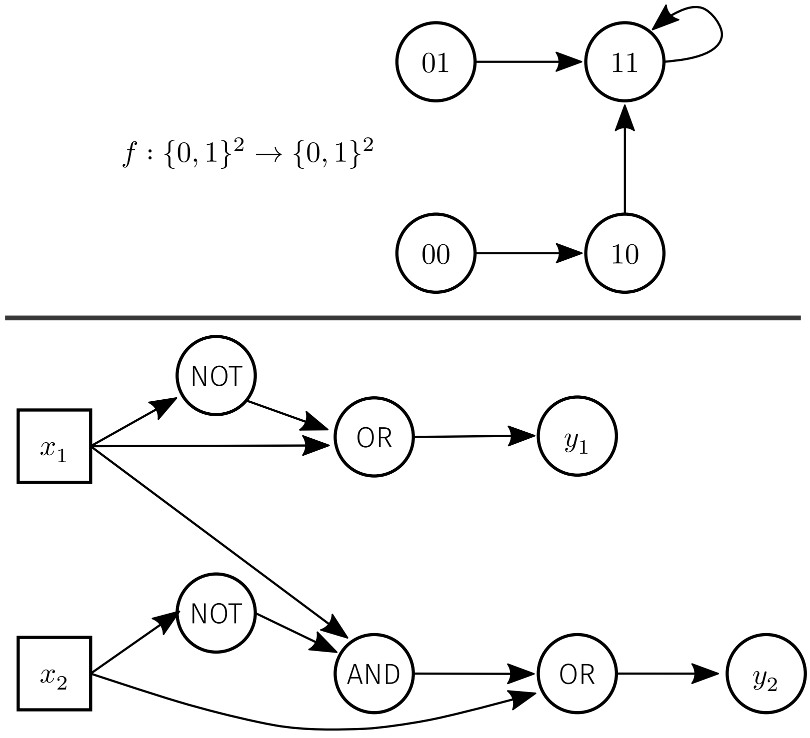

Figure 2.2: Example of how a function $f : \{0,1\}^2 \to \{0,1\}^2$ (on the top), can be computed using boolean circuits (on the bottom).

2.3 Subclasses of TFNP

The existence of complete FNP problems within TFNP would imply that NP = coNP [5], a scenario considered highly unlikely. Consequently, complete problems are not expected within TFNP, necessitating alternative approaches to investigate its structure.

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

TFNP is characterized as a *semantic* class, which means that 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 [6] for a more detailed discussion of these terms.

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

To address these challenges, we want to explore syntactic subclasses of TFNP. One approach, proposed by Papadimitriou [6], is to categorize search problems based on existence proofs that confirm 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 results which gives rise to PLS is:

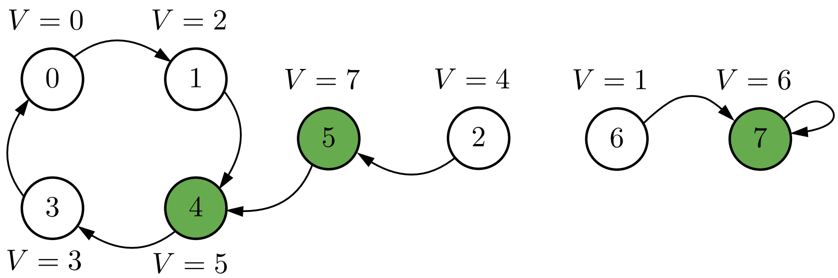

Figure 2.3: Example of a LocalOpt Problem with $n = 3$ (8 vertices). The circuit S is represented by solid lines. The valid solutions are colored green.

"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 [7]:

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

> LocalOpt
> **Input:** Two boolean circuits $S, V : [2^n] \to [2^n]$.
> **Output:** A vertex $v \in [2^n]$ such that $P(S(v)) \geq P(v)$.

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.

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

Studying "easy" problems such as PLS is particularly insightful because we strongly believe that these problems cannot be solved by any method more efficient than simply traversing the graph.. Hence given a graph of exponentially large size, it appears highly improbable to find an efficient solution. Therefore, all problems in **PLS** inherently embody the fundamental challenge of not being able to surpass the basic strategy of navigating through the a 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.

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 syntatic subclasses of **TFNP** in [3]. The existence result giving rise to this class is:

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

"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].

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

End-of-Line
Input: Boolean circuit $S, P : \{0,1\}^n \to \{0,1\}^n$ 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)

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

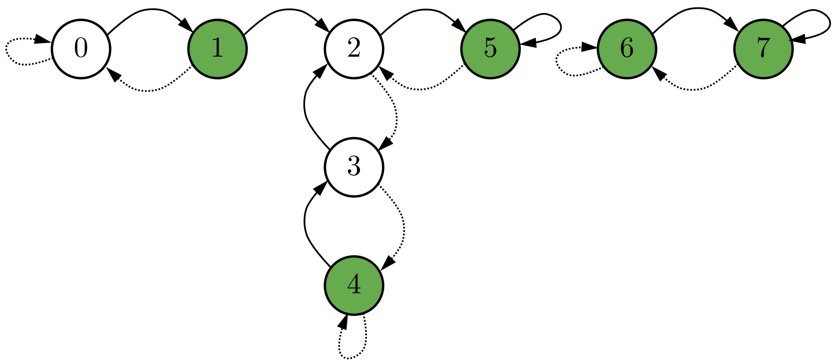

Figure 2.4: Example of an End-of-Line Problem with $n = 3$ (8 vertices). The circuit S is represented by solid lines and the circuit P by dashed lines. The solutions are the sinks $x = 5$, $x = 7$ and $x = 1$, aswell as the sources $x = 4$ and $x = 6$.

These boolean circuits represent a directed graph with maximal in and out degree 1, 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 of 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 [9]. Now we can define the complexity class **PPAD** as follows:

Notice that End-of-Line allows cycles, and that these do not induce solutions.

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

Definition 2.11 — PPAD.
The class **PPAD** is the set of all **TFNP** problems that reduce to End-of-Line.

2.3.3 End of Potential Line (EOPL)

Next we want to discuss the complexity class **EOPL** as introduced in [10]. The existence results giving rise to **EOPL** is:

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

"In a directed acyclic graph, there must be at least two unbalanced vertices."

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

End of Potential Line
Input: Two boolean circuits $S, P : \{0,1\}^n \to \{0,1\}^n$, and a boolean circuit $V : \{0,1\}^n \to [2^n - 1]$, 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 monoticity of the potential)

Here S can be thought of giving the successor of a vertex, and P as giving the predecessor of a vertex. V can be though of as a potential which is suppose to be monotonously increasing along the line.

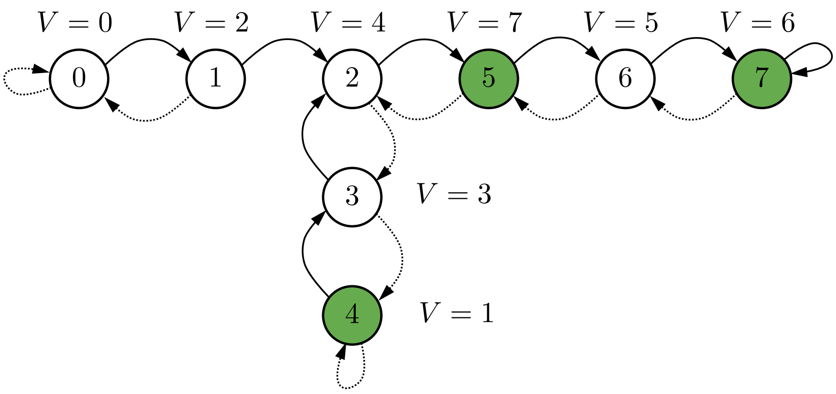

Figure 2.5: Example of an **EOPL** Problem with $n = 3$ (8 vertices). The circuit S is represented by solid lines and the circuit P by dashed lines. The solutions are the sink $x = 7$, the violation of potential at $x = 5$ and the non-standard source $x = 4$.

S and P can be though 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 a garantee 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 latice $[N]^d$, given by $x \leq y$ iff $x_i \leq y_i$ for all $i \in \{1, \ldots, d\}$. The name originates from Tarski's fixed point Theorem as introduced in [1] which we remind the reader of below:

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

TODO: Write about order preserving/monotone functions.

Theorem 2.13 — Tarski's fixed point Theorem.
Let $f : [N]^d \to [N]^d$ a function on the d-dimentional lattice. If f is monotonous (with respect to the previously discussed partial order), then f has a fixed point, i.e. there is an $x \in [N]^d$ such that $f(x) = x$.

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

A proof of this theorem can be found in the previously mentionned work [1]. Without surprise the Tarski problem as defined in [11], is now to find such a fixed-point. Formally we define the problem as follows:

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

Tarski
Input: A boolean circuit $f : [N]^d \to [N]^d$.
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 monoticity).

This is of course a total search problem, as there will always either be a fixed point, or a point violating monoticity. We give an example of a 2 dimensional Tarski instance in Figure 2.6. Before we discuss the location of Tarski in the **TFNP** lanscape 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 instaSnces 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.

Lemma 2.14 — Simplyfying Tarski.
Let $f : L \to L$ be a Tarski instance on a complete lattice L.

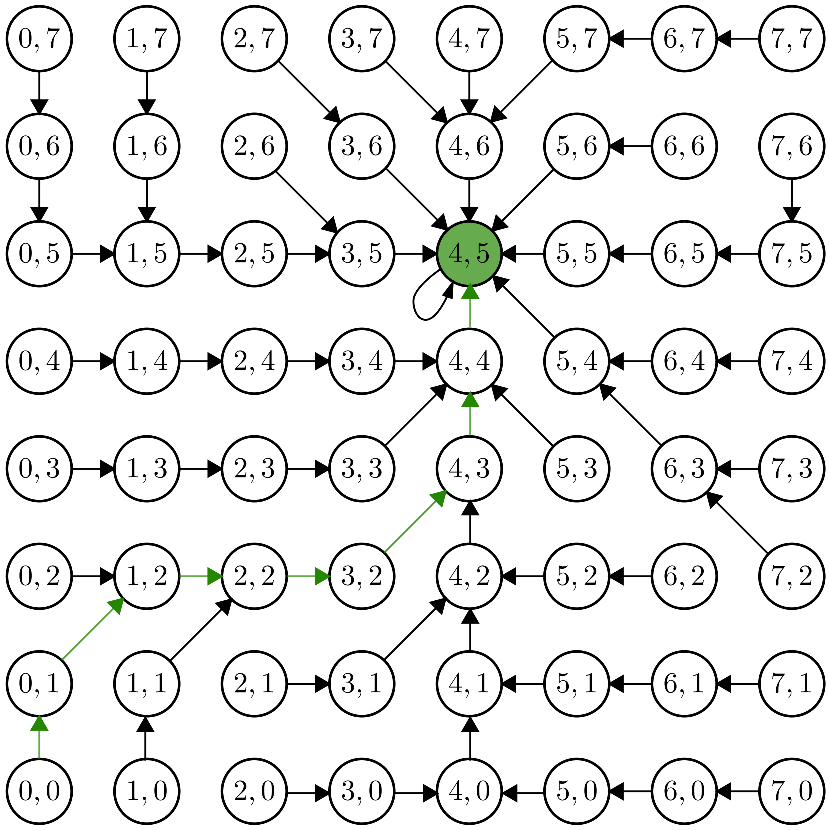

Figure 2.6: Example of a 2 dimensional TARSKI instance.

Consider $\tilde{f} : L \to L$ 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 $\tilde{f}(x)[i] \leq \tilde{f}(y)[i]$. □

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

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.

2.4.2 Two algorithms for solving Tarski

We briefly want to discuss the most commun algorithms used for solving Tarski instances. We begin with a very simple algorithm, which is based on the following observation:

Remark 2.15
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 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 $f : L \to L$
Result: A fixed point of f
$x \leftarrow 0^d$;
while $f(x) \neq x$ **do**
  $x \leftarrow f(x)$;
return x ;

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 [11] 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 the states of the algorithm. The circuit S associates to state of the algorithm the next state it will be in.

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

Next we want to describe a more advanced algorithm for solving Tarski instances. The algorithm we will present is due to [12]. We will give an alternative presentation and simplified proof the correctness of the algorithm here. Before we do this we want to introduce some notation in order to make the argument as clear as possible. For a given complete lattice $L = [N_1] \times \cdots \times [N_d]$ and some dimension $x \in L$ we define the following sublattices:

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

$$
\begin{aligned}
L_{\leq x} &= [x[1]] \times \cdots \times [x[d]], \\
L_{\geq x} &= [\![x[1], N_1]\!] \times \cdot \times [\![x[d], N_d]\!]
\end{aligned}
$$

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

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

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

Remark 2.16
Let $L = [N_1] \times \cdots \times [N_d]$ be a complete lattice and $f : L \to L$ 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. □

Hence points with these propeties seem to be particularly interesting for finding fixed points of f. Hence we want to give them a name:

Definition 2.17 — Progress point.
Let $f : L \to L$ a TARSKI function. We call a point $x \in L$ a *progress point* if $f(x) \leq x$ or $f(x) \geq x$.

The smallest and largest point of a lattice are always progress points.

This means that if we a progress point we can reduce the area in which 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 2.18
Let $f : L \to L$ 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 $f_{k=K} : L_{k=K} \to L_{K=k}$ 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 progess point of f.

This means that 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 monoticity of f.
The fact that x^* is a progess 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 $f(x^*) \geq x^*[k]$. □

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 that we can reduce the search space by a factor of at least 2, 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 size of the search space by a factor of at least 2 in each step. This means that we can solve a d dimensional TARSKI instance with a combination of 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 → L, d):
    /* Binary search in the d-th dimension                */
    l ← 0, r ← N_d ;              /* 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
            L' ← L_{d=m} ;
            /* Solve the d − 1 dimensional instance            */
            x* ← RecursiveTarskiSolver(f_{d=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 $\mathcal{O}\left(\log^d N\right)$ for $L = [N]^d$. It was conjectured by Etessami et. al. that this is an optimal algorithm for TARSKI [11]. This turned out not to be true, as a better algorithms was developped, which mostly relies on a

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

smarter way of finding progress points, or better said, a smarter way of dividing the problem in subinstances [13]. We will not discuss this algorithm in detail, as it is not relevant for the rest of the thesis, but want to mentions that it achieves a runtime of $\mathcal{O}\left(\log^{2\lceil \frac{d}{3} \rceil} N\right)$ for $L = [N]^d$. This is to date the best upper bound for solving Tarski instances.

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

2.4.3 Lower bounds for Tarski

We now want to discuss the lower bounds for solving Tarski instances. The best known lower bounds for Tarski are given by [11]. 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 requieres 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 requieres 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.

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

This gives us reason to study Tarski under the lense 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 [11] that Tarski lies in PLS as we discussed when presenting Algorithm 1. The same paper showed that Tarski lies $\mathsf{P}^{\mathsf{PPAD}}$. We will give an alternative proof of this second fact in Chapter 3. Previous work [14], showed that many-to-one reductions and Turing-reduction onto PPAD are equivalent. In particular this means that $\mathsf{P}^{\mathsf{PPAD}} = \mathsf{PPAD}$, and that Tarski lies in PPAD.

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

Now that we have established that Tarski lies inside PLS $\bigcap$ PPAD, we want to discuss the structure of PLS $\bigcap$ PPAD and describe recent advances in the study of this class. There have been two suprising advances in the study of PLS $\bigcap$ PPAD in the last years. The first is that CLS $=$ PLS $\bigcap$ PPAD [15]. CLS (Continuous Local Search) was first introduced by Daskalakis and Papadimitriou in [16], and can be informally thought of the class of all problems which can be solved by finding the local optimum of a potential in a discrete space with an adjacency relations. This result shows

[15]: Fearnley et al. (2023), *The Complexity of Gradient Descent: CLS = PPAD ∩ PLS*

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

that the problems in PLS $\bigcap$ PPAD are exactly those that can be solved by gradient descent algotithms.

A further notable collapse is the result PLS $\bigcap$ PPAD $=$ EOPL, which was only recently shown in [2]. This of course means that in particular Tarski lies in EOPL. A question which then arise, and which this thesis will try to answer, is can we construct an explicit reduction of Tarski to EndOfPotentialLine.

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

Reducing TARSKI to PPAD | 3

In this chapter, we explore the membership of TARSKI to the complexity class **PPAD**. We begin by presenting an established proof of the reduction of this problem to BROUWER [11], focusing on a high-level overview. Subsequently, we introduce a novel problem, TARSKI*, which facilitates a divide and conquer approach to solving TARSKIby 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 not only simplifies the proof but also sets the stage for further reduction of TARSKI* to **EOPL** in the subsequent chapter.

[11]: 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 [11], 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.

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

Theorem 3.1 — Brouwer's fixed point theorem.
Let $K \subset \mathbb{R}^d$ be a compact, convex set. Then every continuous function $f : K \to K$ has a fixed point $x^* \in K$, i.e. $f(x^*) = x^*$.

The original proof can be found in [17], a simpler proof relying on SPERNER'S LEMMA can be found in [18]. This theorem gives rise to a total search problem which we call BROUWER:

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

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

BROUWER
Input: A continuous function $f : K \to K$.
Output: A fixed point $x^* \in K$ such that $f(x^*) = x^*$.

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 not relevant for this high level presentation.

The problem BROUWER was first introduced and shown to be **PPAD**-complete in [19]. This means that it suffices to reduce TARSKI to BROUWER in order to show that TARSKI is in **PPAD**. We will actually reduce TARSKI to at most polynomially many instances of BROUWER, which will allow us to show that TARSKI is in $\mathsf{P}^{\mathsf{PPAD}}$. This means that we will show a Turing reduction of TARSKI to BROUWER, which suffice as **PPAD** is closed under Turing reductions [14].

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

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

The idea of the 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. This can be achieved 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 key insight is that we can use this fixed point to reduce the search area for a integral fixed point by at least half, or find a violation of monotonicity. In particular, either there is a fixed point in both $\{x \in [2^n - 1]^d : x \geq x^*\}$ and $\{x \in [2^n - 1]^d : x \leq x^*\}$, or there is a violation of monotonicity in the cell containing x^*. We can repeat this procedure always halfing 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 would like to study a single such reduction, in order 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, as for instance used in [11]. We want to construct a problem, which allows us to divide the Tarski problem into two smaller problems, where solving the smaller of the two leads to a solution.

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

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

Tarski*
Input: A boolean circuit $f : L \rightarrow L$.
Output: Either:
(T*1) Two points $x, y \in L$ such that $\|x - y\|_\infty \leq 1$, $x \leq f(x)$ and $y \geq f(y)$, or
(T*2) A violation of monotonicity: Two points $x, y \in L$ such that $x \leq y$ and $f(x) \nleq f(y)$.

We now want to show that Tarski* can be seen as 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 outputed by a Turing machine solving TARSKI*on a function f. We proceed by case distinction:

Case 1: If either $f(x) = x$ or $f(y) = y$, then we are done, 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 $\mathcal{O}\left(\|x-y\|_1\right)$ 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 $|x[i] - y[i]| \leq \|x[i] - y[i]\|_\infty \leq 1$, 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 $\mathcal{O}\left(\|x-y\|_1\right)$ 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 $x \nleq y$, 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 $f(z) \nleq f(x)$, 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 recursively solve the smaller of the two instances. 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 $\mathcal{O}\left(\log 2^{dn}\right) = \mathcal{O}\left(d \cdot n\right)$ 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. In this section, we will substantiate this claim, establishing TARSKI*'s classification within TFNP. Rather than employing *Brouwer's fixed point Theorem* — a cornerstone of continuous topology — we pivot to its discrete analogue, *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, and hence it seems more natural to use a discrete tool.
- Papadimitriou proved that BROUWER is PPAD-complete by reducing BROUWER to SPERNER [19]. Hence by reducing to BROUWER, we introduce continuity into the problem, which is not necessary, as it get removed again behind the scenes.

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

Our goal is to apply *Sperner's Lemma* on the integer lattice. This 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[1] with vertices $v_0, v_1, \ldots, v_d$. We now consider a *simplicial subdivision* of this simplex. This means 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.

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

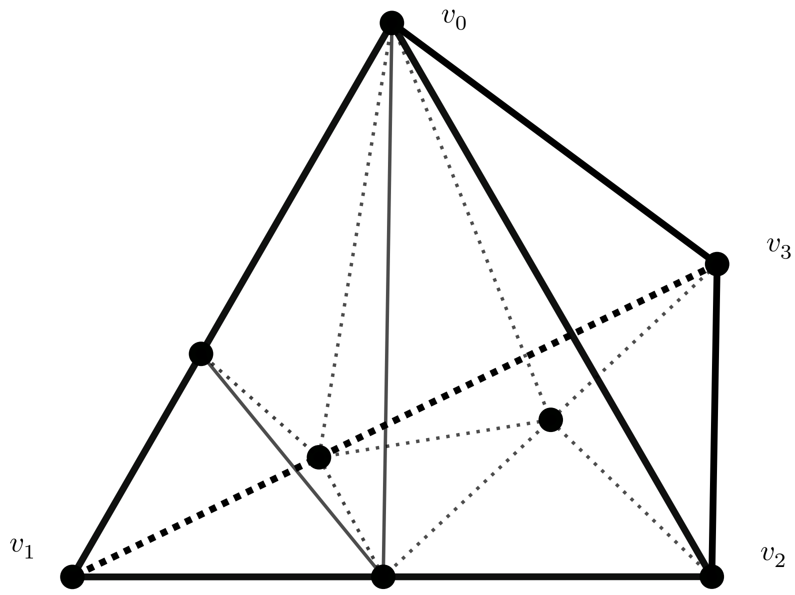

Figure 3.1: Setup for SPERNER'S LEMMA in the 3-dimensional case. The large simplex spanned by v_0, v_1, v_2, v_3 is subdivided into smaller simplices.

Now we introduce a coloring c of the vertices of this subdivision with colors $\{0, 1, \ldots, d\}$. 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 $\{v_{i_0}, \ldots, v_{i_k}\}$ are colored with colors $i_0, \ldots, i_k$. 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 [20], and for which a more modern proof can be found in [18].

[20]: Sperner (1928), *Neuer beweis für die invarianz der dimensionszahl und des gebietes*

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

Theorem 3.3 — Sperner's Lemma.
Suppose that a d-dimensional simplex with vertices $v_0, \ldots, v_d$ is subdivided into smaller simplices. Now color every vertex with a color $\{0, \ldots, d\}$ such that v_i is colored i, and the vertices on a subsimplex $\left\{v_{i_0}, \ldots, v_{i_k}\right\}$ are colored with colors $i_0, \ldots, i_k$. Then there is a subsimplex, with vertices of every color.

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

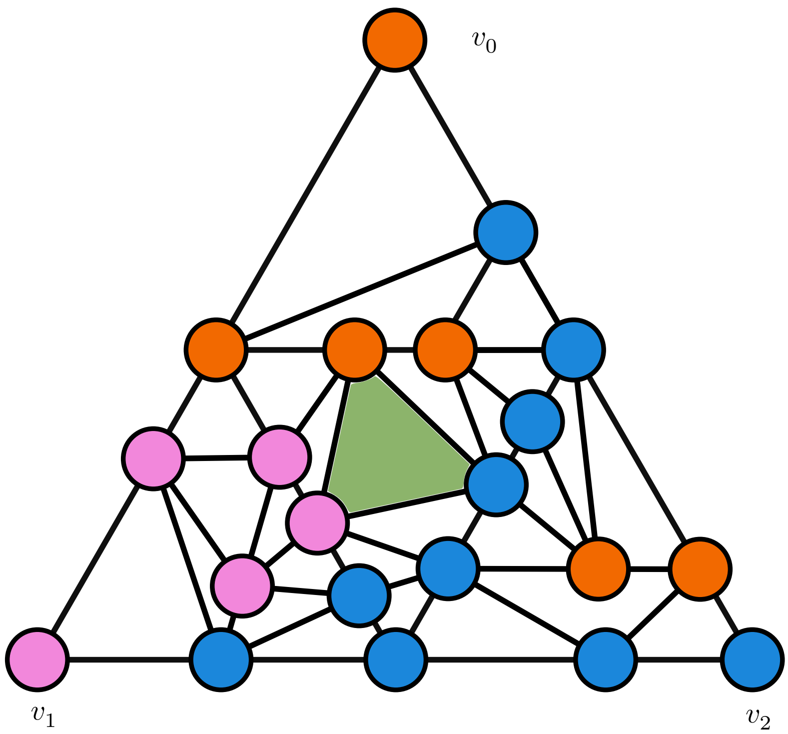

Figure 3.2: Example of SPERNERS LEMMA in the two dimensional case, with 3 colors: orange (0), purple (1) and blue (2). The subsimplex spanned by v_0 and v_1 only contains blue and purple vertices, the subsimplex spanned by v_1 and v_2 contains only purple and blue vertices and the subsimplex spanned 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), which contains all colors.

3.3.2 Sperner's Lemma for an integer lattice

Now that we have introduced *Sperner's Lemma* for a integer lattice. The motivation is to be able to find a region of a colored lattice which contains all colors under certain conditions. Instead of looking for a subsimplex, we will look for a *cell*[2] of the lattice, which contains all colors.

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

In order to do this we proceed as follows. We take the d-dimensional lattice $L = [N_1] \times \dots \times [N_d]$, we subdivide each cell into simplices[3]. 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 an we obtain an equivalent simplex, and a simplicial decomposition of this simplex.

3: How this is done is not relevant in this chapter but will be discussed in the next chapter.

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 simplical decomposition of the lattice, and we will find a simplex which contains all colors. Of course because every subsimplex is included in exactly one cell by construction, there must be a cell which contains all colors. This motivates the definition of the total problem SPERNERwhich was introduced and shown to be **PPAD**-complete in [19]. We introduce the problem for

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

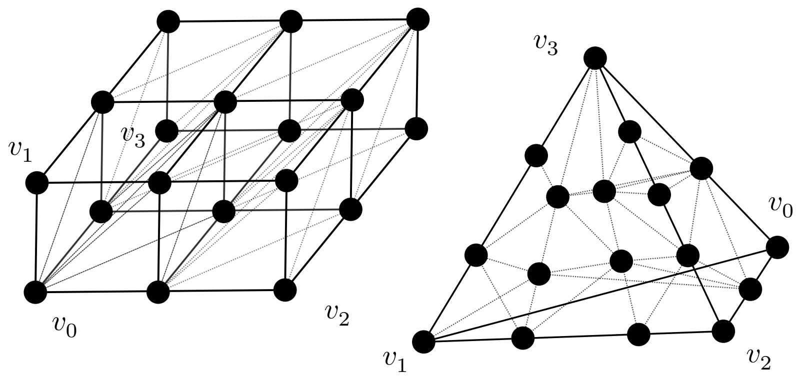

Figure 3.3: Example of the simplicial decomposition of a lattice in the 3 dimensional case on the left, and the equivalent simplicial decomposition on the right of a simplex v_0, v_1, v_2, v_3.

a general lattice $L = N_1 \times \cdots \times N_d$, such that $N_i \leq 2^n$.

SPERNER
Input: A coloring $c : L \to \{0, \dots, d\}$ of the vertices of L, such that for every $i \in \{0, \dots, d\}$ the 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 $c : L \to \{0, \dots, d\}$:

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.

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

We now need two results. First we need to show that a cell with all colors always exists, which will allow 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 $\{(0, \dots, 0), (2^n - 1, 0, \dots, 0), \dots, (0, \dots, 2^n - 1)\}$. □

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 **0** be the smallest vertex of this hypercube and **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:

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

$$f(x_0)[i] - f(x_i)[i] \geq 2$$

This implies that $f(x_0) \nleq f(x_i)$, 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: $f(\mathbf{0})[i] \nleq f(x_i)[i]$, 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 that we can solve a TARSKI* instance in $\mathcal{O}(d)$ additional steps. □

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

Reducing TARSKI to EOPL 4

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 simplical 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 Simplical 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 simplical 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 which 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 litterature as *Freudenthal's simplical decomposition* [21]. We will introduce it

[21]: Freudenthal (1942), *Simplizialzerlegungen von Beschrankter 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_π 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 $x[\pi(1)] \le x[\pi(2)] \le \dots \le x[\pi(d)]$. Then $x \in S_\pi$.

Proof. We want to show that x is a convex combination of the vertices of S_π. 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 $\lambda_i \ge 0$ 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_π 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_π, and thus $x \in S_\pi$. ☐

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 appart from on the boundary of the simplices.

Lemma 4.3
Let $S_\pi \in \mathcal{S}$ be a simplex. Then the *interior* of S_π 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. ☐

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_π 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 $v_i[\pi(i+1)] = v_{i+1}[\pi(i+1)] - 1$. ☐

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 $x \nleq y$ 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 succintly. Assume that a permutation π of the dimensions, induces a simplex S_π, woth vertices $v_0, \dots, v_d$, as defined in Definition 4.1. Then we will denote the d-dimensional simplex S_π as:

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

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_π obtained by removing the vertex v_i as:

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

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:

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

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

$$S_1: \quad v_0 \xrightarrow{\pi(1)} \cdots v_{i-1} \xrightarrow{\pi(i+1)} w_i \xrightarrow{\pi(i)} v_i \xrightarrow{\pi(i+2)} \cdots \xrightarrow{\pi(d)} v_d$$
$$S_2: \quad v_0 \xrightarrow{\pi(1)} \cdots v_{i-1} \xrightarrow{\pi(i)} w_i' \xrightarrow{\pi(i+1)} v_i \xrightarrow{\pi(i+2)} \cdots \xrightarrow{\pi(d)} v_d$$

(2) If F is of the form:

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

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:

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

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

Lemma 4.7 — Neighbooring 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 neighboors of S_π:

- $v_0 \xrightarrow{\pi(2)} v_1 \xrightarrow{\pi(1)} \cdots v_{i-1} \xrightarrow{\pi(i+1)} w_i \xrightarrow{\pi(i+1)} v_i \xrightarrow{\pi(i+2)} \cdots \xrightarrow{\pi(d)} v_d$, 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_π and using remark 4.6. □

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 π and σ of a simplex S are *equivalent* if they differ by an even permutation. That is if $\sigma = \pi \circ \tau$ for some permutation τ 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:

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

Lemma 4.10
Two orientations σ, τ of a simplex S are equivalent if and only if $\mathbf{sgn}(\sigma) = \mathbf{sgn}(\tau)$, with respect to the total order $\preceq$.

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 appearent latter 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. □

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

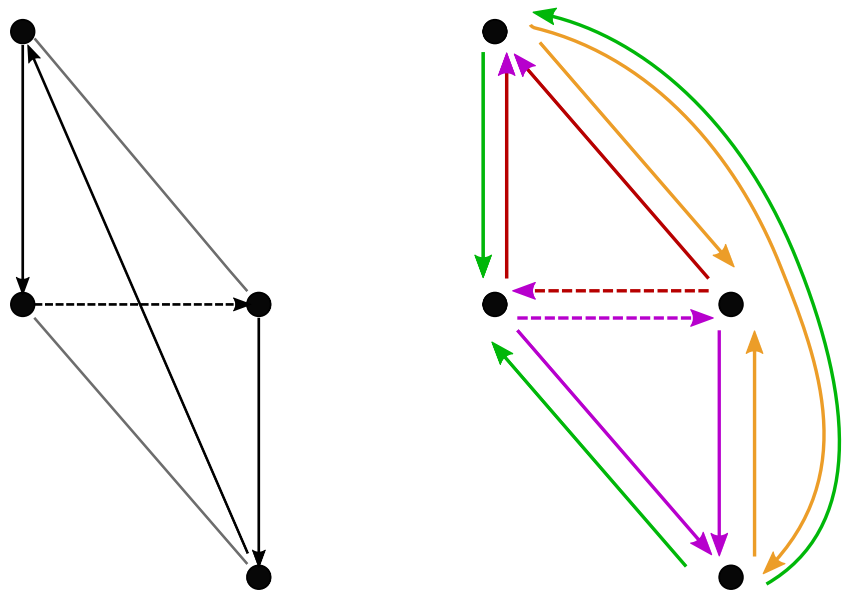

Figure 4.1: Example of the orientation of a 3-simplex on the left, and the induced orientation of the faces on the right.

to extend this notion of orientation to these simplicial complexes. Formally we define a simplicial complex as follows [22]:

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

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 sence "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 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.

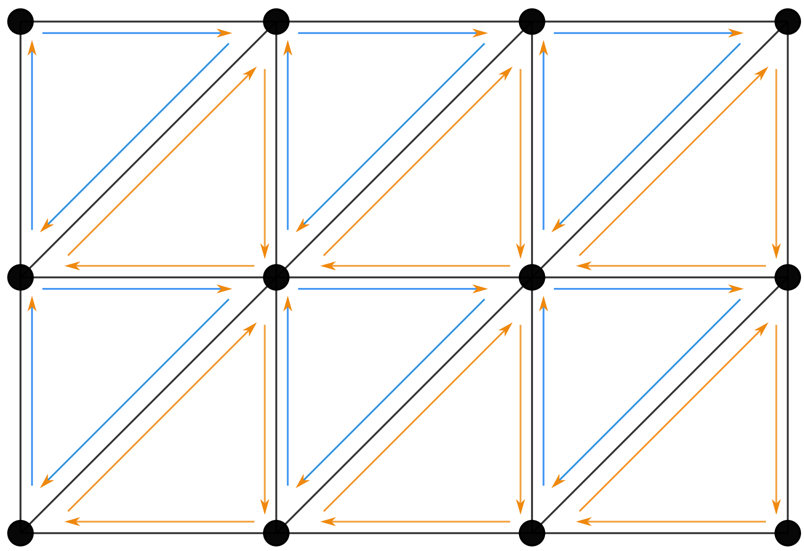

Figure 4.2: Example of the orientation of a Freundenthals simplicial complex in 2 dimensions.

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:

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

We now orient S_π using the permutation:

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

First we notice that for all $d-2$ simplices, two neighbooring $d-1$-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:

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

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

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

We immediately notice that $\mathbf{sgn}(S_1) = -\mathbf{sgn}(S_2)$. 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 $\mathbf{sgn}(S_1)$ the sign of the permutation inducing S_1.

Case 2: Next assume that S_1 and S_2 are in neighbooring cells, then as dicussed in Remark 4.6 F is of the form:

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

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

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

We once again must proceed by case distinction.

Case 2.1: If d is even, then: $\mathbf{sgn}(S_1) = -\mathbf{sgn}(S_2)$, 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 $\mathbf{sgn}(S_1) = \mathbf{sgn}(S_2)$, 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. ☐

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) $S_{i+1} \not\subset \{S_1, \ldots, S_i\}$ for all $i \in \{1, \ldots, k-1\}$.

(2) S_i and S_{i+1} share a $d-1$-dimensional face F_i for all $i \in \{1, \ldots, k-1\}$.

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 $i \in \{1, \ldots, k\}$.

(2) L_i and L_{i+1} share a $d-2$-dimensional face G_i for all $i \in \{1, \ldots, k-1\}$.

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. □

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 $f : L \to L$, the points have been colored $c : L \to \{0, \dots, d\}$ 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:

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

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 $x_i \ngeq x_j$

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 monoticity 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 $x_i \ngeq x_j$. This shows that the lemma holds. □

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 $x_i \ngeq x_j$ is equivalent to $x_i \le x_j$.

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 $i, j \in \{1, \ldots, d\}$ 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. □

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 $C \subset \{0, \ldots, d\}$ 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.

Inuitively 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 $x_{i+1}^j = y$. □

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}. □

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. □

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:

$$(S): \quad v_0 \xrightarrow{l} v_1 \to \cdots \to v_d$$

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

$$(F): \quad c(v_1) \to \cdots \xrightarrow{d} \cdots 0 \to \cdots \to c(v_d)$$

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:

$$(F): \quad c(v_1) \to \cdots 0 \to \cdots \xrightarrow{c} \cdots \to c$$

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

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

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:

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

This shows the Lemma. □

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 appart 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:

$$(F_i): \quad c(v_1) \rightarrow \cdots \xrightarrow{d} \cdots 0 \rightarrow \cdots \rightarrow c(v_d)$$
$$(F_j): \quad c(w_1) \rightarrow \cdots \xrightarrow{d} \cdots 0 \rightarrow \cdots \rightarrow c(w_d)$$

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] 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, 12).

[2] Mika Goos et al. 'Further Collapses in TFNP'. In: (2022) (cited on pages 2, 18).

[3] 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. (Visited on 03/05/2024) (cited on pages 4, 10).

[4] 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 5).

[5] 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 8).

[6] Christos H. Papadimitriou. *Computational complexity*. Reading (Mass): Addison-Wesley, 1994 (cited on page 8).

[7] 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 page 9).

[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 page 10).

[9] 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. (Visited on 03/10/2024) (cited on page 10).

[10] 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 page 11).

[11] 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 12, 14, 16, 17, 19, 20).

[12] 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 14).

[13] 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. (Visited on 07/21/2024) (cited on page 17).

[14] 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. (Visited on 02/24/2024) (cited on pages 17, 19).

[15] John Fearnley et al. 'The Complexity of Gradient Descent: CLS = PPAD ∩ PLS'. In: *Journal of the ACM* 70.1 (Feb. 28, 2023), pp. 1–74. DOI: 10.1145/3568163. (Visited on 07/21/2024) (cited on page 17).

[16] 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 page 17).

[17] L. E. J. Brouwer. 'Über Abbildung von Mannigfaltigkeiten'. In: *Mathematische Annalen* 71.1 (Mar. 1911), pp. 97–115. DOI: 10.1007/BF01456931. (Visited on 05/04/2024) (cited on page 19).

[18] 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 19, 23).

[19] 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. (Visited on 03/22/2024) (cited on pages 19, 22, 24).

[20] 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. (Visited on 04/18/2024) (cited on page 23).

[21] Hans Freudenthal. 'Simplizialzerlegungen von Beschrankter Flachheit'. In: *The Annals of Mathematics* 43.3 (July 1942), p. 580. DOI: 10.2307/1968813. (Visited on 03/21/2024) (cited on page 28).

[22] James Raymond Munkres. *Elements of algebraic topology*. The advanced book program. Boca Raton London New York: CRC Press, 2018. 454 pp. (cited on page 35).

Alphabetical Index