The Complexity of Finding Tarski Fixed Points

Master Thesis

August 27, 2024

Nils Jensen

ADVISED BY

PROF. DR. BERND GÄRTNER

SEBASTIAN HASLEBACHER

ETH zürich

Abstract

Insert the abstract here.

Acknowledgements

Insert the acknowledgements here.

Contents

List of Figures

List of Tables

List of Algorithms

Introduction | 1

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 size of the input. This endeavor has proven particularly challenging for many problems, with a significant gap between the best-known upper bounds, determined by existing algorithms, and the best-known lower bound.

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 reduce onto the same fundamental problem.

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 problems 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 existence of a k-clique in a graph, it is likely one would wish to explicitly identify this clique or verify its absence. Similarly, in addition to recognizing a number as prime, one might want to determine its prime factors. Instead of simply deciding whether a function has a global minimum, the objective would be to compute it efficiently.

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 proven by mathematical theorems. A notable example within this subclass is the problem of identifying a sink in a directed acyclic graph. This is a *total* problem because every such graph has a sink.

1.2 The TFNP landscape

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

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

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

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

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

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

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

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

1.3 The Tarski problem

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

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

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

The TARSKI-problem has numerous applications in various fields. For example, it can be shown that supermodular games, which model certain economic situation, have a equilibrium by *Tarski's Theorem* [6, 7]. These equilibria can be found by solving TARSKI-instances [5]. The existence of equilibria in some stochastic games can be found using *Tarski's Theorem*, and finding this equilibria can be reduced to solving a TARSKI instance [8].

[6]: Topkis (1979), *Equilibrium Points in Nonzero-Sum n -Person Submodular Games*
[7]: Milgrom and Roberts (1990), *Rationalizability, Learning, and Equilibrium in Games with Strategic Complementarities*
[8]: Condon (1992), *The complexity of stochastic games*
[9]: Gärtner et al. (2021), *A subexponential algorithm for ARRIVAL*

Another application of this problem can be found when studying the celebrated ARRIVAL problem. It can be shown that ARRIVAL reduces to TARSKI [9] hence studying the TARSKI problem can help understand the complexity of ARRIVAL.

1.4 Current algorithms for solving TARSKI

We want to give an overview of the different known strategies for solving TARSKI-instances. This has a theoretical interest, as the state of these algorithms often describe graphs, which can be seen as instances of **TFNP**-complete problems, and hence can help construct reductions.

The most fundamental approach to solving the Tarski problem is a simple iterative algorithm that leverages the monotonicity of the function to converge to a fixed point iteratively. Starting from the smallest point within the lattice, the Algorithm applies the function repeatedly until a fixed point is reached [5]. This method is straightforward but can be computationally expensive, as it may require a large number of iterations to converge, particularly for functions defined over large lattices in the worst case for a lattice $L = [N]^d$, this Algorithm requires time $\mathcal{O}(N \cdot d)$.

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

A more sophisticated approach involves a binary search technique, where the lattice is systematically divided, and the function's monotonicity is used to eliminate regions that cannot contain a fixed point. This is done by recursively solving lower-dimensional subproblems until the fixed point is found [10]. This method can significantly reduce the search space and converges faster than the iterative Algorithm, with a runtime of $\mathcal{O}\left(\log^d(N)\right)$.

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

The latest developments in solving the TARSKI problem involve advanced decomposition methods that reduce the search space. These methods decompose the problem into smaller instances that can be more easily managed and solved. Using these techniques a runtime of $\mathcal{O}\left(\log^{\lceil \frac{d-1}{2} \rceil} N\right)$ can be achieved [11].

[11]: Chen and Li (2022), *Improved Upper Bounds for Finding Tarski Fixed Points*

1.5 Location of Tarski in TFNP

It is known that the Tarski problem lies in **PPAD** and in **PLS**. A recent breakthrough has shown that the class $\mathsf{PPAD} \cap \mathsf{PLS} = \mathsf{EOPL}$ [12]. This result immediately implies that the Tarski problem is in **EOPL**, which in turn means that there must be a reduction from Tarski to **EOPL**-complete problems, in particular to the EndOfPotentialLine problem. The main goal of this thesis is to understand why Tarski lies in **EOPL** and to construct a reduction from Tarski to the EndOfPotentialLine problem, which is **EOPL**-complete.

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

1.6 Thesis Outline

TODO: Write this section.

Preliminaries 2

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

2.1 Total search problems

The study of complexity classes has traditionally focused on *decision problems*, which involve determining whether an object belongs to a set, also called a *language*. Notable examples include determining whether a Boolean formula is satisfiable or whether a k-clique exists in a given graph [13]. However, real-world questions often require explicit answers rather than existence results. For example, while deciding whether a function has a global minimum is a decision problem, the practical interest lies in identifying that minimum, which goes beyond mere existence. Here, so-called *search problems* come into play.

[13]: Arora and Barak (2009), *Computational complexity : a modern approach*

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 is, 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 rephrased the valid language as the pair of a problem instance and a valid solution.

The above shows that every search problem can be seen as a decision problem of a broader language. This perspective allows us to ask classical complexity questions about search problems: Are these problems in **P** or **NP**? Are they **NP**-hard? It is evident that search problems are at least as complex as their decision counterparts since solving a search problem inherently solves the associated decision question.

Similarly to decision problems, we want to study which problems can be solved efficiently and which cannot. This question leads us to the definition of the complexity class **FNP**, which is pendant for search problems, to **NP** for decision problems. We introduce **FNP** formally as in [14].

[14]: Megiddo and Papadimitriou (1991), *On total functions, existence theorems and computational complexity*

Definition 2.2 — Function NP (FNP).
We say that a relation $R \subset \{0,1\}^* \times \{0,1\}^*$ is in **FNP** if it is *polynomially balanced*, i.e. there exists a polynomial p such that for every $(I, s) \in R$ implies $|y| \leq p(|x|)$. The class **FNP** is the set of all relations R that are polynomially balanced.

This means that checking a solution to a search problem can be done in polynomial time. What cannot be done in polynomial time is checking whether no solution exists. This leads to the question of what happens if we remove this decision problem from the search problem. This question is what leads to the definition of a total search problem [14]:

Definition 2.3 — Total search problems.
A *total search problem* is a search problem given by a relation $R \subset \{0,1\}^* \times \{0,1\}^*$, such that for every given instance $I \in \{0,1\}^*$ there is a solution $s \in \{0,1\}^*$ that satisfies: $(I, s) \in R$.

This means that a solution always exists for any input, i.e. "No" is never a valid answer.

The complexity class **TFNP** is simply the class of all *total* problems in **FNP**.

Definition 2.4 — Total Function NP (TFNP).
The class **TFNP** is the set of all problems given by total relations polynomially bounded relations R.

Similarly to **P**, **FP** is the class of all search problems that can be solved in polynomial time. It is not known whether **FP** is equal to **TFNP**, but it is widely believed — similarly to **P** and **NP** — that they are different. Examples of **TFNP** problems are:

- Factoring, the problem of finding the prime factors of a number. Every number admits a factorization into prime numbers, which can be checked in polynomial time;
- Nash, the problem of finding a Nash equilibrium in a bimatrix game [15];
- Minimize, the problem of finding the global minimum of a convex function [16].

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

[16]: Daskalakis and Papadimitriou (2011), *Continuous Local Search*

2.1.2 Reductions

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

Definition 2.5 — 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 $I, s \in \{0,1\}^*$:

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

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

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

Notice that many-to-one reductions map *instances* to *instances*, if we instead assume that we can compute a solution, we use a *Turing reduction*, which we introduce analogously to the classical Turing reduction.

Definition 2.6 — Turing Reduction.
For two problems $R, S \in$ **TFNP**, we say that R *Turing reduces* to S if a polynomial-time oracle Turing machine that solves R given access to an oracle for S exists.

An *oracle* is a black-box which solves S.

2.1.3 Promise Problems

We have defined total search problems as problems where a solution always exists for *any* input in $\{0,1\}^*$. However, in practice, we often study problems where a solution is guaranteed to exist only for a subset of the inputs. For instance, every convex function has a global minimum, but this existence result relies on the fact that we are given a convex function. This leads us to the notion of *promise problems* as introduced in [17]. Formally, we restrict the instance space to some subset $\mathcal{X} \subset \{0,1\}^*$. We only require our algorithm to solve the problem for instances in $\mathcal{X}$, and it can behave arbitrarily on instances outside of $\mathcal{X}$.

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

We highlight that formally **TFNP** does not contain promise problems where $\mathcal{X} \neq \{0,1\}^*$. We still want to study these problems inside the **TFNP**-Framework. There is a trick for restricting the input space to a subset $\mathcal{X} \subset \{0,1\}^*$, where the language $\mathcal{X}$ can be decided in polynomial time. For a search problem R on $X \subset \{0,1\}^*$, we can then define the promise problem R' on $\{0,1\}^*$ by adding a solution $(I, \star)$ to R for all $I \in \{0,1\}^* \setminus R$, where $\star$ is some special binary string. Because it can be decided in polynomial time whether an instance is in $\mathcal{X}$, we can solve R' by checking whether the instance is in $\mathcal{X}$ and then solving R, hence obtaining a problem in **TFNP**.

For example, in this thesis, we use syntactic validation when the instances are a function or a boolean circuit to validate that the given input is indeed an encoding of a function or circuit. This verification can be done in polynomial time [18], and a special binary string can be outputed if this verification fails. For example, this is the case for the Tarski problem, where the instances are boolean circuits, and the validity of the instances can be checked in polynomial time. For the sake of simplicity, we will assume that this step implicitly when defining **TFNP**-Problems, and allow instances which have an input space $\mathcal{X} \subset \{0,1\}^*$ if it can be validated in polynomial time.

[18]: Greenlaw and Hoover (1998), *Chapter 9 - Circuit Complexity*

Additionally to this syntactic validation, we can construct **TFNP**-instances by adding *violations* to the solution space. For example, if we are interested in finding the global minima of convex functions, we can construct a total search problem by:

(1) Checking syntactically that the input defines a function;
(2) Adding a violation of convexity to the solution space. Formally, this is done by changing the relation R to ensure that a solution exists for every instance I; this can be thought of as allowing more solutions.

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

This means we can often construct a **TFNP**-problem starting out with a promise-problem by checking the validity of the input syntactically and adding violations to the solution space. However, it is essential to note that this is only sometimes the case and that constructing a **TFNP** problem from a promise problem can be a non-trivial task. Also, there is no unique way of constructing **TFNP**-Problems from promise problems, and care has to be taken to introduce the studied problem rigorously.

2.1.4 Representation of sets

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

Similarly, when considering the d-dimensional case, we can represent the set $L = [2^n]^d$, which corresponds to a d-dimensional lattice with side length 2^n, as the set of binary strings of length $n \cdot d$, i.e. $\{0,1\}^{nd}$. Again, for simplicity, while the underlying functions rely on the binary strings, these naturally correspond to a unique point $(x_1, \dots, x_d) \in [2^n]^d$. We will use both notations interchangeably.

2.1.5 Representation of functions

Now that we have described the sets, we can describe how we represent the functions. We will represent the functions by using so-called boolean circuits. In this section, we will rely on the presentation of boolean circuits described in [18] and refer an interested reader to this source for a more detailed description.

[18]: Greenlaw and Hoover (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 2.7 — 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**: $g(x_1, x_2) = x_1 \wedge x_2$,
- **OR-gate**: $g(x_1, x_2) = x_1 \vee x_2$,
- **NOT-gate**: $g(x) = \neg x$.

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

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

Definition 2.8 — Boolean circuit.
A boolean circuit C is a labeled finite 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 following properties:

- If $\tau(v) = \text{INPUT}$, then v has no incoming edges. We call these vertices the *inputs gates*.
- If $\tau(v) = \text{OUTPUT}$, then v has one incoming edge. We call these vertices the *output gates*.
- If $\tau(v) = \text{AND}$, then v has two incoming edges. We call these vertices the *AND-gates*.
- If $\tau(v) = \text{OR}$, then v has two incoming edges. We call these vertices the *OR-gates*.

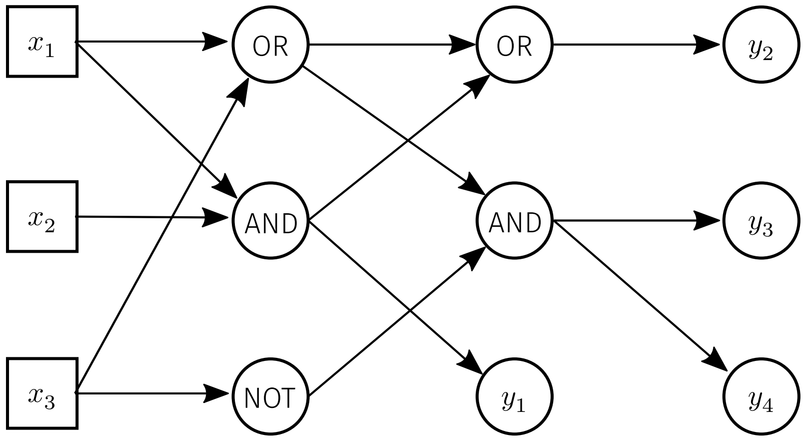

Figure 2.1: example of a Boolean circuit with three input and four output gates.

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

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

Definition 2.9 — Computed function of a boolean circuit. A boolean circuit C with inputs $x_1, \ldots, x_n$ and outputs $y_1, \ldots, y_m$ 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.

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

2.1.6 Complexity of boolean circuits

Of course, formally, the complexity of a problem is defined in terms of the *size* of the input. This means we also need to define

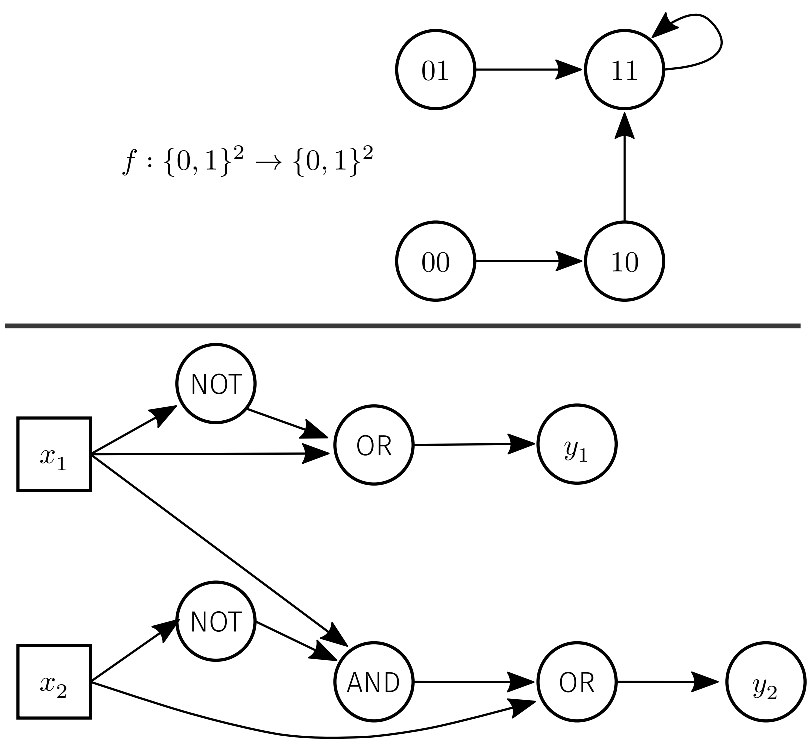

Figure 2.2: example of how a function $f : \{0,1\}^2 \to \{0,1\}^2$ (on the top), can be computed using boolean circuits (on the bottom).

what we mean by the size of a boolean circuit. We will use the following definition:

Definition 2.10 — Size of a boolean circuit.
The size of a boolean circuit C is the number of gates in the circuit.

The size of the boolean circuits is a measure of the input complexity, 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 2.11 — Depth of a boolean circuit.
The depth of a boolean circuit C is the length of the longest path from an input gate to an output gate.

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

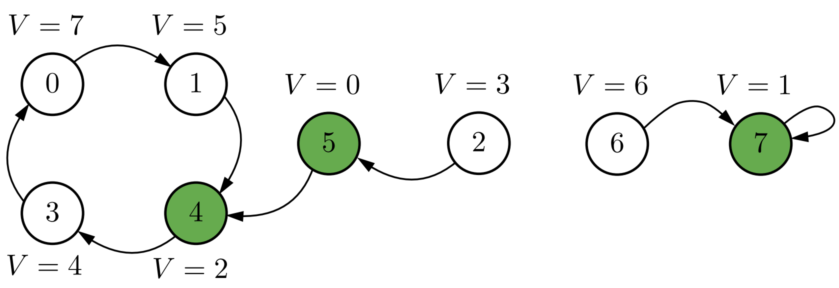

Figure 2.3: Example of a LocalOpt Problem with $n = 3$ (8 vertices). Solid lines represent the circuit S. The valid solutions are colored green.

2.2 Subclasses of TFNP

Complete **FNP** problems within **TFNP** would imply that **NP** $=$ **coNP** [14], a highly unlikely scenario. Consequently, complete problems are not expected within **TFNP**, necessitating alternative approaches to investigate its structure.

[14]: Megiddo and Papadimitriou (1991), *On total functions, existence theorems and computational complexity*

TFNP is known as a *semantic* class [19]. This is known to mean that it is unlikely that we can find complete **TFNP** problems [20]. We refer the reader to Papadimitriou's work for a more detailed discussion of these terms and their implications. We want to explore *syntactic* subclasses of **TFNP** to address this challenge. One approach, proposed by Papadimitriou [19], categorizes search problems based on existence proofs confirming their totalness. This basic strategy leads to the detailed study of specific complexity classes discussed in the following sections.

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

[20]: Sipser (1982), *On relativization and the existence of complete sets*

2.2.1 Polynomial Local Search (PLS)

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

"Every directed acyclic graph has a sink."

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

[2]: 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.

Let us discuss why solving a LocalOpt instance 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 2.12 — Polynomial Local Search (PLS).
The class **PLS** is the set of all **TFNP** problems that reduce to LocalOpt.

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

2.2.2 Polynomial Parity Argument on Directed Graphs (PPAD)

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

[1]: 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 [15].

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

End-of-Line (EoL)
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 as giving the successor of a vertex, and P as giving the predecessor of a vertex.

These boolean circuits represent a directed graph with maximal in and out-degree of one by having an edge from x to y if and only if $S(x) = y$ and $P(y) = x$. The goal is to find a sink in the graph or another source. It can be shown that the general case of finding a second imbalanced vertex in a directed graph (a problem called Imbalance) can be reduced to End-of-Line [21]. Now we can define the complexity class **PPAD** as follows:

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

[21]: Goldberg and Hollender (2021), *The Hairy Ball problem is PPAD-complete*

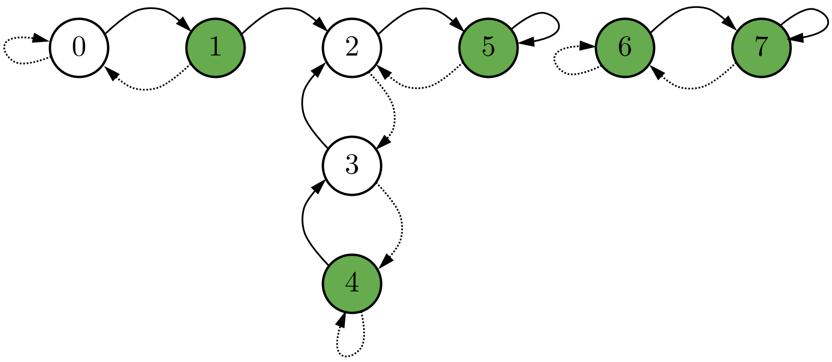

Figure 2.4: Example of an END-OF-LINE Problem with $n = 3$ (8 vertices). Solid lines represent the circuit S, and dashed lines represent the circuit P. The solutions are the sinks $x = 5$, $x = 7$ and $x = 1$, aswell as the sources $x = 4$ and $x = 6$.

Definition 2.13 — PPAD.
The class **PPAD** is the set of all **TFNP** problems that reduce to END-OF-LINE.

2.2.3 End of Potential Line (EOPL)

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

[3]: 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 as giving the successor of a vertex, and P as giving the predecessor of a vertex. V is a potential that is supposed to increase monotonously along the line.

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

Definition 2.14 — EOPL.
The class **EOPL** is the set of all **TFNP** problems that reduce to END OF POTENTIAL LINE.

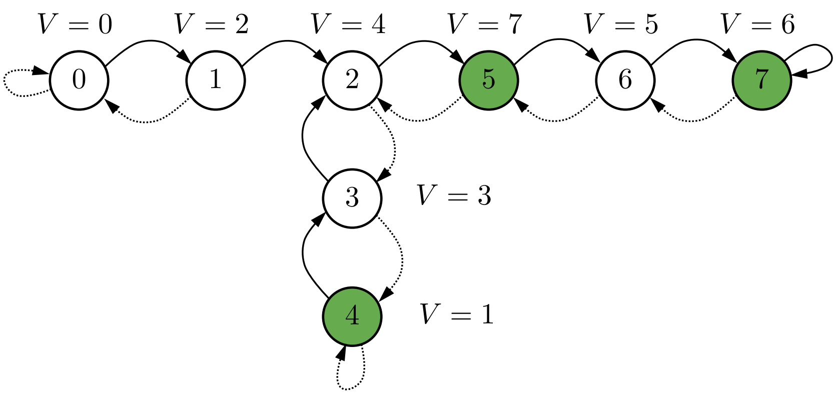

Figure 2.5: Example of an **EOPL** Problem with $n = 3$ (8 vertices). Solid lines represent the circuit S, and dashed lines represent the circuit P. The solutions are the sink $x = 7$, the violation of potential at $x = 5$, and the non-standard source $x = 4$.

2.3 The Tarski Problem

2.3.1 Definition of the Tarski Problem

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

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

Definition 2.15 — Monotone function.
A function $f : [N]^d \to [N]^d$ is *monotone* if for all $x, y \in [N]^d$ we have $x \leq y$ implies $f(x) \leq f(y)$.

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

The Tarski-problem originates from Tarski's fixed point Theorem, introduced in [4]. In our setting the Theorem states the following:

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

Theorem 2.16 — Tarski's fixed point Theorem.
Let $f : [N]^d \to [N]^d$ a function on the d-dimentional lattice. If f is monotone (for the previously discussed partial order), then f has a fixed point, i.e. there is an $x \in [N]^d$ such that $f(x) = x$.

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

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

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

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

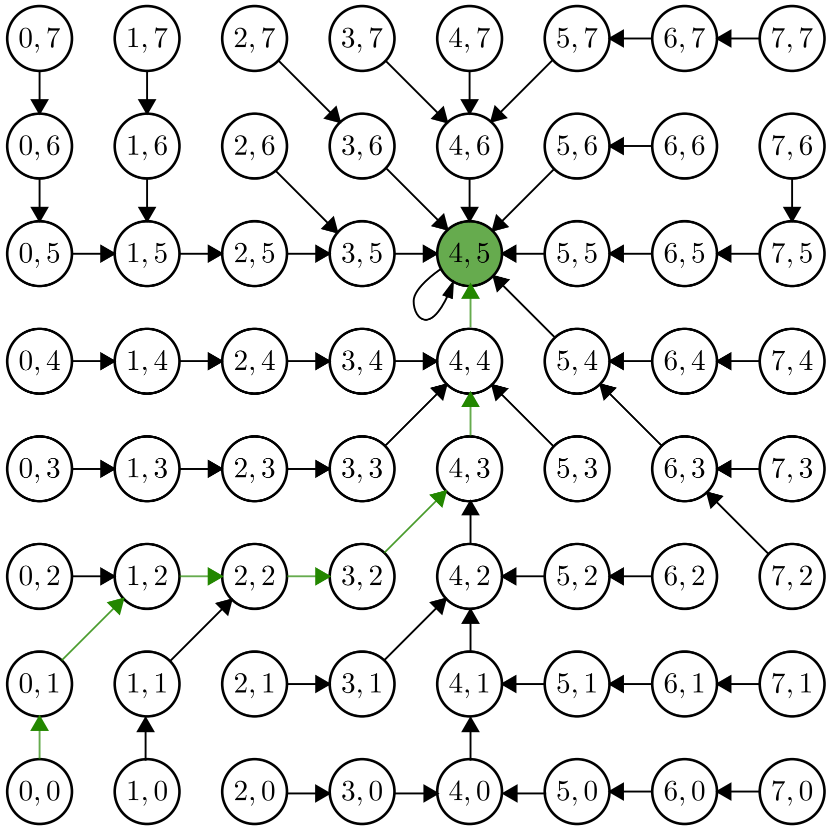

Figure 2.6: Example of a 2 dimensional TARSKI instance. A fixed point is located at $x = (4, 5)$. The path to the fixed point which Algorithm 1 finds is colored in green.

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

Lemma 2.17 — Simplyfying TARSKI.
Let $f : [2^n]^d \to [2^n]^d$ be a TARSKI instance on a complete lattice $[2^n]^d$. Consider $\tilde{f} : [2^n]^d \to [2^n]^d$ 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 $x, y \in [2^n]^d$, $f(x) \leq f(y)$ if and only if $\tilde{f}(x) \leq \tilde{f}(y)$.

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

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

This means that in this thesis, we can consider the simplified version of the TARSKI problem, where for every $x \in [2^n]^d$, we have $\|x - f(x)\|_\infty \leq 1$, which we will implicitly assume from now on.

Also note that given a circuit that computes f, we can construct a circuit with at most $\mathcal{O}(\text{poly}\, d)$ additional gates which computes $\tilde{f}$.

2.3.2 Two algorithms for solving TARSKI

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

Remark 2.18
Let f be a TARSKI instance on a complete lattice L. If f is monotone, and for some $x \in L$ we have $f(x) \geq x$, then $f(f(x)) \geq f(x)$.

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

Algorithm 1: Iterative Algorithm for TARSKI

```
Data: A boolean circuit f : L → L
Result: A fixed point of f
x ← 0 ;
while f(x) ≠ x do
    if f(x) ≱ x then
        return "x_old, x are a violation of monotonicity." ;
    else
        x_old ← x x ← f(x) ;
return x ;
```

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

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

Next, we describe a more advanced algorithm, due to [10], for solving TARSKI instances and also give an alternative presentation and simplified proof of its correctness. The following notation aims 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:

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

$$L_{\leq x} = [x[1]+1] \times \cdots \times [x[d]+1],$$
$$L_{\geq x} = [\![x[1], N_1]\!] \times \cdots \times [\![x[d], N_d]\!]$$

We denote by $[\![a, b]\!]$ the set of whole numbers $\{a, a+1, \ldots, b\}$.

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

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

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

Remark 2.19
Let $L = [N_1] \times \cdots \times [N_d]$ be a complete lattice and $f : L \to L$ a monotone 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 properties seem particularly interesting when searching for fixed points of f. Hence, we want to give them a name:

Definition 2.20 — 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 lattice's smallest vertex and largest vertex are always progress points.

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

Remark 2.21
Let $f : L \to L$ on a complete lattice $L = [N_1] \times \cdots \times [N_d]$, for a monotone 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.

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

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

The fact that x^* is a 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 we can reduce the search space by a factor of at least two by solving a $d-1$ dimensional TARSKI instance. We can solve a d dimensional TARSKI instance by repeatedly solving $d-1$ dimensional TARSKI instances and reducing the search space size by a factor of at least 2 in each step. This means we can solve a d dimensional TARSKI instance by combining a $d-1$ dimensional TARSKI solver and a binary search. The $d-1$ dimensional instances can be solved recursively. We give the recursive algorithm for solving TARSKI instances in Algorithm 2.

A simple analysis shows that this algorithm runs in $\mathcal{O}\left(\log^d N\right)$ for $L = [N]^d$. It was conjectured that this is an optimal algorithm for TARSKI [5]. This turned out not to be true, as a better algorithm was developed [22], which mostly relies on a faster way of finding a progress point in the three-dimensional case, which they call the *inner algorithm*. An *outer algorithm* then repeatedly applies the inner algorithm on 3-dimensional instances. Overall this approach achieves a query complexity of $\mathcal{O}\left(\log^{2\lceil \frac{d}{3} \rceil} N\right)$ for $L = [N]^d$.

A further advance using more advanced methods were made in [11]. They achieve a query complexity of $\mathcal{O}\left(\log^{\lceil \frac{d-1}{2} \rceil} N\right)$ for

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

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

[11]: Chen and Li (2022), *Improved Upper Bounds for Finding Tarski Fixed Points*

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 then
            x* ← m
        else
            /* 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*
```

$L = [N]^d$. This is to date the best known bound for solving TARSKI instances.

2.3.3 Lower bounds for TARSKI

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

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

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

2.3.4 Location of TARSKI in TFNP

Next we summarize where TARSKI lies inside of **TFNP**. It has been shown in [5] that TARSKI lies in **PLS** as we discussed when presenting Algorithm 1. The same paper showed that TARSKI lies $\mathbf{P}^{\mathbf{PPAD}}$. We will provide an alternative proof of this second fact in Chapter 3. Previous work [23] showed that many-to-one reductions and Turing-reduction onto **PPAD** are equivalent. In

[23]: Buss and Johnson (2012), *Propositional proofs and reductions between NP search problems*

particular this means that $\mathsf{P}^{\mathsf{PPAD}} = \mathsf{PPAD}$, and that TARSKI lies in PPAD.

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 surprising advances in the study of PLS $\bigcap$ PPAD in the last few years. The first is that CLS = PLS $\bigcap$ PPAD [24]. CLS (Continuous Local Search) was first introduced by Daskalakis and Papadimitriou in [16] and can be informally thought of as the class of all problems that can be solved by finding the local optimum of a potential in a discrete space equipped with an adjacency relation. This result shows that the problems in PLS $\bigcap$ PPAD are exactly those that gradient descent algorithms can solve.

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

[16]: Daskalakis and Papadimitriou (2011), *Continuous Local Search*

A further notable collapse is the result PLS $\bigcap$ PPAD = EOPL, which was only recently shown in [12]. This means that TARSKI lies in EOPL. A question that then arises, and which this thesis will attempt to answer, is whether we can construct an explicit reduction of TARSKI to ENDOFPOTENTIALLINE.

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

2.3.5 Variants of TARSKI

Before we conclude this chapter, we want to discuss some variants of the TARSKI problem. The first variant we introduce is the *promise* variant of TARSKI, which is defined as follows:

PROMISETARSKI
Input: A boolean circuit $f : [N]^d \to [N]^d$ such that $x \leq y \implies f(x) \leq f(y)$
Output: A fixpoint x of f.

Notably in this variant of TARSKI we are *promised* that we have a monotone function, and hence we do not allow violations of monotonicity as solutions (as these should not be possible).

Next we introduce two variants of TARSKI which are also promise variants of TARSKI. Instead of promising that the function is monotone, we are promised some properties related to uniqueness of solutions. The first variant is the *unique fixpoint* variant of TARSKI, which is defined as follows:

UNIQUETARSKI
Input: A boolean circuit $f : [N]^d \to [N]^d$ such that f is monotone and there is a unique fixpoint x
Output: The unique fixpoint x of f.

A very notable recent result, is that the query complexity of UniqueTarski is equal to the query complexity of Tarski [25]. We also introduce an even stronger uniqueness variant of Tarskinext. This time the promise is that when fixing a subset of coordinates, the induced function has a unique fixpoint, i.e. $\tilde{f} = f_{d_1=K_1,\dots,d_k=K_k}$ has a unique fixpoint for any choice of $d_1, \dots, d_k \in \{1, \dots, d\}$ and $K_1, \dots, K_k \in [N]$.

[25]: Chen et al. (2023), *Reducing Tarski to Unique Tarski (In the Black-Box Model)*

SuperUniqueTarski
Input: A boolean circuit $f : [N]^d \to [N]^d$ such that f is monotone and when fixing a subset of coordinates the induce function $\tilde{f}$ has a unique fixpoint.
Output: The unique fixpoint x of f.

The problem SuperUniqueTarski is a very strong promise variant of Tarski, it implies that Algorithm 2 find a unique fixpoint in every step.

Reducing TARSKI to PPAD | 3

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

3.1 Presentation of the known reduction of TARSKI to PPAD

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

[5]: 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 [26], and a more straightforward proof relying on SPERNER'S LEMMA can be found in [27]. This theorem gives rise to a total search problem, which we call BROUWER:

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

[27]: Aigner and Ziegler (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 irrelevant for this high-level presentation.

The problem BROUWER was first introduced and shown to be **PPAD**-complete in [28], meaning that it suffices to reduce TARSKI to BROUWER in order to show that TARSKI is in **PPAD**. We will reduce TARSKI to, at most polynomially, many instances of BROUWER, which will allow us to show that TARSKI is in $\mathbf{P}^{\mathbf{PPAD}}$. Overall, we will construct a Turing reduction of TARSKI to BROUWER, which suffice as **PPAD** is closed under Turing reductions [23].

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

[23]: Buss and Johnson (2012), *Propositional proofs and reductions between NP search problems*

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

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

3.2 Introducing Tarski*

In the previous section, we have seen that Tarski can be reduced to a polynomial number of Brouwer instances. We want to study a single such reduction to give an alternative proof that Tarski is in **PPAD**. In order to do this, we introduce a new problem, Tarski*. This problem can be thought of as a subproblem towards solving Tarski. A standard strategy to solve Tarski is to use a *divide-and-conquer* strategy, for instance, used in [5], and presented previously in Algorithm 2, of Subsection 2.3.2. We want to construct a problem that allows us to divide the Tarski problem into two smaller problems, where solving the smaller of the two leads to a solution.

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

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

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

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

Claim 3.2
An instance of Tarski can be solved using $\mathcal{O}(d \cdot n)$ calls to Tarski* and up to $\mathcal{O}(d)$ additional function evaluations.

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

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

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

Case 2.2: If $x < y$ and $f(x) \leq f(y)$, we claim that we can solve the Tarski instance in $\mathcal{O}\left(\|x-y\|_1\right)$ 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 $x \not\leq y$, then we can partition the set of lattice points into two sets S_x and S_y, as follows:

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

These two sets are disjoint: if there was a $z \in S_x \cap S_y$, then $x \leq z \leq y$, which would imply $x \leq y$, which is a contradiction. We will show that S_x must contain a solution to the TARSKI instance. If for some $z \in S_x$, we have $f(z) \notin S_x$, then we have $f(z) \not\leq f(x)$, which means that z and x form a violation of monotonicity. This means that S_x forms a new valid instance of TARSKI. By the same argumentation, S_y also forms a valid instance of TARSKI and hence, it suffices to solve the smaller of the two instances recursively. In particular, because they are disjoint, one of the instances S_x or S_y contains less than half of the lattice points of L, and hence we can solve the instance in $\mathcal{O}\left(\log 2^{dn}\right) = \mathcal{O}\left(d \cdot n\right)$ 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. This section will substantiate this claim, establishing Tarskistar's classification within **TFNP**. Rather than employing *Brouwer's fixed point theorem* — a cornerstone of continuous topology — we pivot to its discrete analog, *Sperner's Lemma*, a foundational result in combinatorial topology. This approach is particularly apt for two main reasons:

- We are working on a discrete lattice, so it seems more natural to use a discrete tool.
- Papadimitriou proved that BROUWER is **PPAD**-complete by reducing BROUWER to SPERNER [28]. Hence, by reducing to BROUWER, we introduce continuity into the problem, which is unnecessary, as it gets removed again behind the scenes.

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

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

3.3.1 Sperner's Lemma for Simplices

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

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

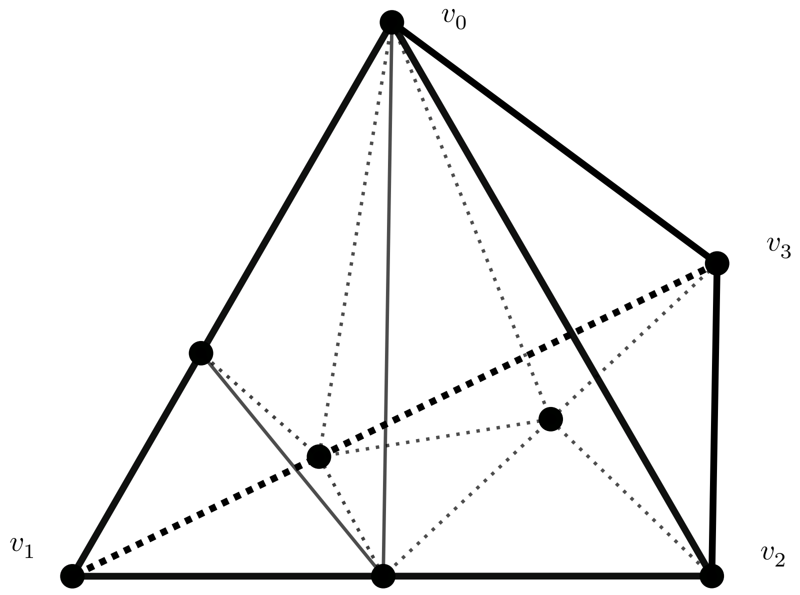

Figure 3.1: Setup for SPERNER'S LEMMA in the 3-dimensional case. The large simplex spanned by v_0, v_1, v_2, v_3 is subdivided into smaller simplices.

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

We now introduce Sperner's Lemma, which was first proven in [29], and for which a more modern proof can be found in [27].

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

[27]: Aigner and Ziegler (2018), *Proofs from THE BOOK*

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

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

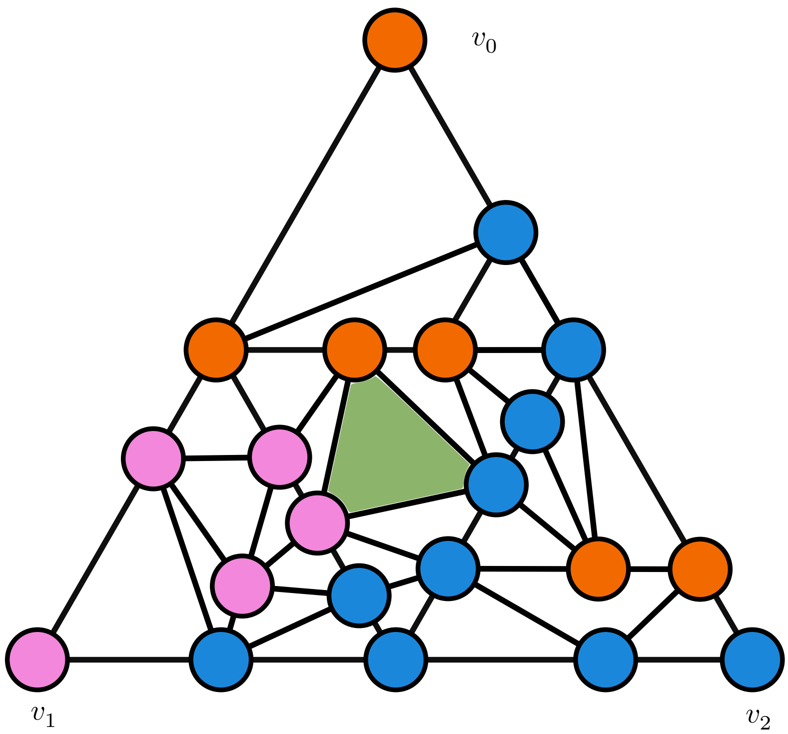

Figure 3.2: Example of SPERNERS LEMMA in the two-dimensional case, with three colors: orange (0), purple (1), and blue (2). The subsimplex spanned by v_0 and v_1 only contains blue and purple vertices, the subsimplex spanned by v_1 and v_2 contains only purple and blue vertices, and the subsimplex spanned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

3.3.2 Sperner's Lemma for an integer lattice

In the previous section, we introduced *Sperner's Lemma* for a large outer simplex. We want to construct a **TFNP** problem, which captures the finding of the subsimplex containing all colors. The challenge here is that encoding the structure of this simplex is not straightforward, and hence, defining circuits that encode the setting of the problem will be difficult. Furthermore, our final goal is to reduce TARSKI* to the problem of finding a subsimplex containing all colors. In this case, our setting is not a large outer simplex but an integer lattice.

We want to apply *Sperner's Lemma* to an integer lattice for these reasons. We proceed as follows: We take the d-dimensional lattice $L = [N_1] \times \cdots \times [N_d]$, we subdivide each cell into simplices[1]. We need a technical lemma, which states that the simplicial complex we obtain can be deformed into a large outer simplex.

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

Lemma 3.4 — Simplicial deformation of integer lattice.
Let $L = [N_1] \times \cdots \times [N_d]$ be a d-dimensional lattice. We subdivide each cell into simplices. Then, there is a deformation of the lattice into a large outer simplex.

Proof. We start by choosing the vertices of the lattice which will form the outer simplex. Consider the following $d + 1$ vertices:

$$\begin{aligned} v_0 &= (0, \dots, 0) \\ v_1 &= (N_1, 0, \dots, 0) \\ v_2 &= (0, N_2, 0, \dots, 0) \\ &\vdots \\ v_d &= (0, \dots, 0, N_d) \end{aligned}$$

Now, we need to discuss how we can deform the lattice into this simplex. The idea is the following: Every point on the boundary of the lattices should be moved to the boundary of the simplex. The points in the interior of the lattice should be moved to the interior of the simplex. We can do this by a linear interpolation between the two points. We now give the construction of the deformation function $D : [0, N_1] \times \dots \times [0, N_d] \to [0, N_1] \times \dots \times [0, N_d]$. Notice that we define the deformation function for the hypercube $H = [0, N_1] \times \dots \times [0, N_d]$ for simplicity. The deformation of the lattice then immediately follows by restricting the deformation function to the lattice.

Let $x = (x_1, \dots, x_d)$ be a point on the boundary of the hypercube H, i.e. there is a $i \in \{1, \dots, d\}$ such that $x_i = 0$ or $x_i = N_i$. Notice that we can write:

$$x = \sum_{i=1}^{d} \lambda_i \cdot v_i \quad \text{with} \quad \lambda_i > 0$$

Now, we can define the deformation function for these points on the boundary as follows:

$$D(x) = \frac{1}{\|\lambda\|_1} \cdot \sum_{i=1}^{d} \lambda_i \cdot v_i$$

We immediately notice that if x is on the boundary of the hypercube, then $D(x)$ is on the boundary of the simplex. We can now define the deformation function for the interior of the hypercube by linear interpolation. Let y be a point in the interior of the hypercube; then we can write: $y = \gamma \cdot x$, for some x on the boundary of the hypercube and $\gamma \in [0, 1]$. We can now define the deformation function for the interior of the hypercube as follows:

$$D(y) = \gamma \cdot D(x)$$

This construction has some fundamental properties. Most importantly, segments (edges of the simplicial decomposition) are mapped to segments, and simplices are mapped to simplices.

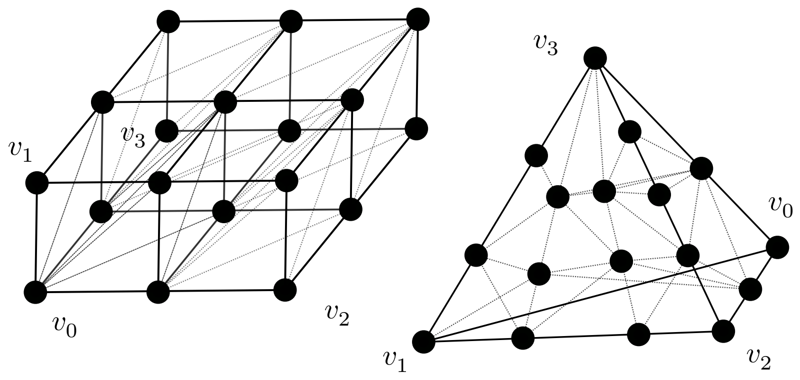

Figure 3.3: Example of the simplicial decomposition of a lattice in the three-dimensional case on the left, and the equivalent simplicial decomposition on the right of a simplex v_0, v_1, v_2, v_3.

This means that the simplicial decomposition of the lattice is mapped to a simplicial decomposition of the simplex. It follows that we can apply *Sperner's Lemma* to the simplicial decomposition of the lattice. This concludes the proof. □

We give an example of such a subdivision in the 3-dimensional case in Figure 3.3. Notice that we can deform the lattice and obtain an equivalent simplex and a simplicial decomposition of this simplex.

Assuming that we color all vertices of the lattice with colors $\{0, \ldots, d\}$, such that v_i is colored i, and every vertex x with $x[i] = 0$, is *not* colored i for $i \in \{1, \ldots, d\}$. Then, we can apply *Sperner's Lemma* to this simplicial decomposition of the lattice, and we will find a simplex that contains all colors. Of course, because every subsimplex is included in exactly one cell by construction, there must be a cell that contains all colors. This motivates the definition of the total problem SPERNER, which was introduced in [28]. We introduce the problem for a general lattice $L = N_1 \times \cdots \times N_d$, such that $N_i \leq 2^n$.

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

SPERNER
Input: A coloring $c : L \rightarrow \{0, \ldots, d\}$ of the vertices of L, such that for every $i \in \{0, \ldots, d\}$ the the vertices $\{x \in L : x[i] = 0\}$ are not colored i.
Output: A cell C such that for all $i \in \{0, \ldots, d\}$ there is a vertex $x \in C$ such that $c(x) = i$.

3.3.3 Reducing SPERNER to ENDOFLINE

This section will discuss the reduction of SPERNER to ENDOFLINE. This reduction was first constructed in the three-dimensional case in [28]. The same paper also gives the idea of the generalization to the d-dimensional case. In the following section, we will

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

give the complete construction of the reduction. This is important as we will use this reduction to argue as to why TARSKI* is in EOPL.

Theorem 3.5 — SPERNER is in PPAD.
SPERNER reduces to ENDOFLINE.

We start by giving the idea of the construction. We want to find a cell that contains all colors: assume that we start at a cell with all but one color d. Then, we move to the neighboring cell through a face containing all colors but d. Now, either this cell contains the color d, in which case we are finished, or we have a second face containing all colors but d. We can repeat this process until we find a cell containing all colors.

Now, there are two problems we need to discuss. First, once again, using a cell leads to some difficulty as there could be more than two faces for a given cell, which could contain all colors but one. We will solve this problem again by considering the lattice's simplicial decomposition. Second, we need to define a designated source. In order to do this, we will expand the simplicial complex slightly. We are now ready for the formal proof; we recommend the reader to follow along with the construction in Figure 3.4.

Proof. Formally, we will proceed by induction over the dimensions of the lattice. First, we discuss the base case. We have a lattice $L = [N]$ in the one dimensional case. We color the lattice with colors $\{0, 1\}$. Now for every segment $s = [x, x+1]$, of which there are $N-1$ which we number from left to right of the lattice, we define the circuits $S, P : [N-1] \to [N-1]$ as follows:

$$S(x) = \begin{cases} x+1 & \text{if } c(x+1) = 0 \\ x & \text{else} \end{cases} \quad \text{and}$$

$$P(x) = \begin{cases} x-1 & \text{if } c(x-1) = 0 \\ x & \text{else} \end{cases}$$

Furthemore set $P(0) = \tilde{x}$ and $S(N-1) = N-1$. By adding a designated source $\tilde{x}$ outside of the lattice and setting $P(\tilde{x}) = \tilde{x}$ and $S(\tilde{x}) = 0$, we obtain an instance of ENDOFLINE.

Now, we are ready to proceed with the induction. Assume the theorem holds for all dimensions $1, \dots, d-1$. We will prove it for dimension d.

Consider the simplicial subdivision of the colored lattice. A given cell is subdivided into $d!$ simplices. It follows that we have

A more detailed discussion of this is discussed in Section 4.2.

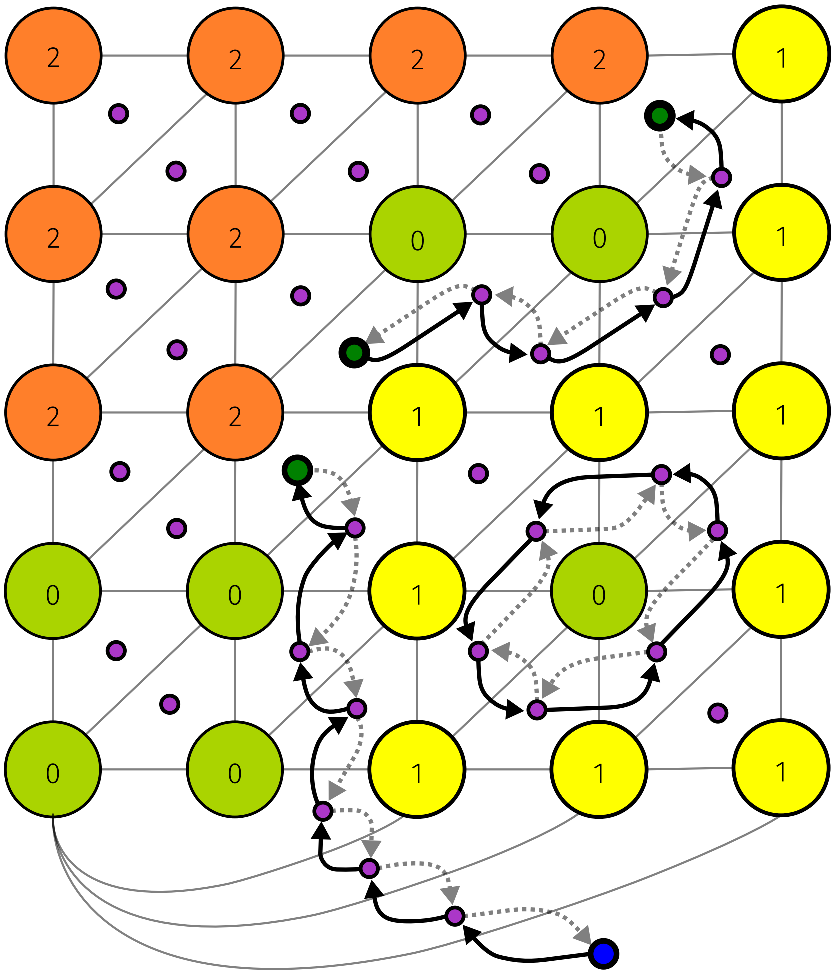

Figure 3.4: Reduction of two dimensional Sperner to EndOfLine. The circuit S is given by the full arrows and S by the dashed arrows. The solutions are colored green. The designated source is colored blue.

$N = N_1 \cdot \dots \cdot N_d \cdot d!$ simplices, which we can number from 0 to $N-1$.

We now can define the circuit $S, P : [N] \to [N]$. For a given d-simplex $x \in [N]$, we consider the color of the vertices. We proceed by case distinction:

Case 1: If x has a vertex with every color in $\{0, \dots, d-1\}$, but no vertex colored d, then x has two faces colored with all colors but d. One of these faces is oriented positively, and one is oriented negatively. Now, define $S(x)$ as the simplex obtained by moving from x through the positively oriented face and define $P(x)$ as the simplex obtained by moving to the negatively oriented face.

See Subsection 4.5.1 for a detailed discussion of how we define these orientations.

Case 2: If x is a simplex with all colors, look at the face spanned by the vertices colored $\{0, \ldots, d-1\}$. If this face is positively oriented, then define $S(x)$ as the simplex obtained by moving through this face and $P(x) = x$. If this face is oriented negatively, then define $P(x)$ as the simplex obtained by moving through this face and $S(x) = x$. Notice that these are the sources/sinks of the circuit and the solutions to the sperner instances.

Case 3: We define $S(x) = P(x) = x$ in all other cases.

Notice that we can compute S and P in polynomial time with respect to d. We still need to define a distinguished source. Finding a distinguished source can be seen as finding a face colored with all colors $\{0, \ldots, d-1\}$ on the face spanned by the vertices $v_0, \ldots, v_{d-1}$. We can do this by solving a $d-1$ dimensional Sperner instance on this face. By induction hypothesis, this can be reduced to an EndOfLine instance. We can now define the distinguished source as the source of this EndOfLine instance and slightly modify the circuit S and P to combine this EndOfLine instance with the circuits S and P we have constructed above. This concludes the proof. □

Next, we will use this problem to show that Tarski* reduces to Sperner 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 $c : L \to \{0, \ldots, d\}$:

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 give an example of the coloring of a Tarski instance in Figure 3.5. We now need two results. First, we need to show that a cell with all colors always exists, allowing us to show that Tarski* is a total search problem. Second, we need to show that finding a cell with all colors yields a solution to Tarski*in polynomial time.

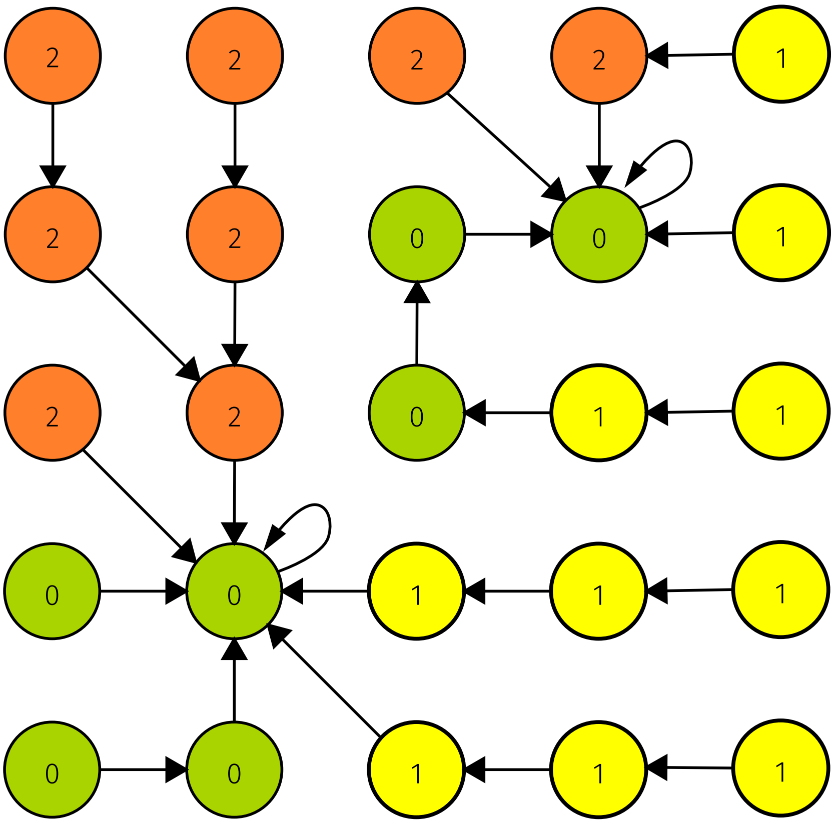

Figure 3.5: Coloring of a TARSKI* instance on a 2-dimensional lattice. The vertices colored 0 indicate that the function points weakly forward in all dimensions, the vertices colored 1 indicate that the function points backward in the first dimension, and the vertices colored 2 indicate that the function points backward in the second dimension and not in the first.

Claim 3.6
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 3.7
Finding a cell with all colors yields a solution to TARSKI*, in $\mathcal{O}(d)$ additional steps.

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

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

We now proceed by case distinction:

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

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

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

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

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

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

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

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 demonstrated how one can prove 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. We do this 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 Warmup: No cycles in two-dimensional TARSKI*-instances

The aim of this chapter is to prove that there are no cycles in the ENDOFLINE instances that result from the reduction of monotone TARSKI*-instances onto ENDOFLINEwe presented in Subsection 3.3.3. We will do this by studying the SPERNER instances, and showing that the sequence of simplices that form the lines of the ENDOFLINE-instances, cannot form a cycle if the underlying function f is monotone.

Proving this will be a somewhat involed in the general d-dimensional case. To motivate the definitions, and the ideas we will use in the following sections, we want to start by presenting a proof for the much more simple two-dimensional case.

Proposition 4.1
The ENDOFLINE-instance that results from reducing the two-dimensional SPERNER-instance generated by a two-dimensional monotone function $f : L \to L$, has no cycles.

Proof. The idea of the proof will be to show that any sequence of simplices that are generated by walking through edges colored $0-1$, can only cross an axis parallel to the x-axis, once.

Let $L_{d_1=K}$ for $K \in [N]$ be such an axis given by:

$$L_{d_1=K} = \{x \in L : x[1] = K\}$$

Now let us assume that we have a path of simplices that crosses $L_{d_1=K}$ twice. Then there must be two edges colored $0-1$ on $L_{d_1=K}$. Now let us call these two edges (x_1, x_2) and (y_1, y_2), such that $x_1 < x_2$ and $y_1 < y_2$. We want to discuss how these edges can be colored. If $c(x_1) = 0$ and $c(x_2) = 1$, we get a contradiction because this leads to $f(x_1)[1] > f(x_2)[1]$. Hence this cannot occur. A similar argument for y_1, y_2 yields that $c(x_1) = c(y_1) = 1$ and $c(x_2) = c(y_2) = 0$. For convenience a sketch of the situation is given in Figure 4.1.

Now when walking along edges colored $0-1$, we notice that the edge we traverse should always be oriented in the same way. For instance the vertex on our left should be colored 0 and the vertex on our right colored 1. Without loss of generality assume that we crossed the edge $x_1 - x_2$ first, then the vertices $y_1 - y_2$ should be colored opposite, to allow us to traverse $L_{d_1=K}$ in the reverse direction. But this is not the case which means that we traverse $L_{d_1=K}$ at most once. This is true for any $K \in [N]$ which garantees that a cycle cannot exist. $\square$

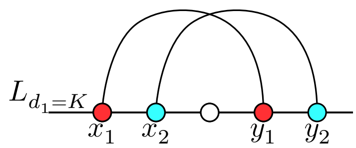

Figure 4.1: Sketch of the setting for the two dimensional proof

Let us briefly discuss what the key ingredients for this proof were and what the steps are that we will take to generalize them to the d-dimensional case.

On a high level we want to show that one cannot cross a given hyperplane of the lattice more than once when going through rainbow-faces colored $\{0, \dots, d-1\}$. This turns out not to be true but we will be able to show a weaker result, which still prohibits cycles. In two dimensions we did this by showing that when we traverse such a hyperplane we had restrictions on how the edges could be oriented. We will generalize this notion of *orientation* for the colored faces we are working with. We will then achieve a contradiction by showing that these faces must be oriented the same way if they are part of a path of simplices that form a line in the ENDOFLINE reduction and obtain a contradiction from this.

In order to this we will need to discuss in detail how we subdivide the cells of the lattice into simplices, we will do this by introducing *Freudenthal's simplicial decomposition*. Once we have the simplices, we will discuss *sequences of simplices* which are the objects that will turn into the lines of the ENDOFLINE instances. Then we will need to define the notion of *orientation* for colored simplices. We will then need to show that the monotonicity of f prohibits certain simplices from existing, and use this to obtain a contradiction to the existence of cycles.

4.2 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 2.3. 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* [30]. We will introduce it in a combinatorial way here, and refer the reader to the original paper for a geometric construction of the same decomposition.

[30]: Freudenthal (1942), *Simplizialzerlegungen von Beschrankter Flachheit*

Definition 4.2 — 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:

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

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

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

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 4.3
Let $x = (x[1], \dots, x[d]) \in [0,1]^d$, let $\pi \in S^d$ be the permutation such that $x[\pi(1)] \leq x[\pi(2)] \leq \dots \leq x[\pi(d)]$. Then $x \in S_\pi$.

Proof. We want to show that x is a convex combination of the vertices of S_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 $\lambda_i \geq 0$ for all i and $\sum_{i=0}^{d} \lambda_i = 1$, by telescoping the sum. We can now write x as a convex combination of the vertices of S_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 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 4.4
Let $S_\pi \in \mathcal{S}$ be a simplex. Then the *interior* of S_π is:

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

Proof. The same proof as for lemma 4.3, 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 Lemmata 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 4.5
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 2.3. 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 4.6
For two vertices x, y of any simplex $S \in \mathcal{S}$, if for any $i \in \{1, \dots, d\}$ we have $x[i] < y[i]$, then $x < y$. In particular $x \not\leq y$ is equivalent to $x > y$.

Notice that this is not the case for any two points in the hypercube, as the partial order is not a total order. This is why choosing a simplicial decomposition with this property will be crucial in the following sections. Next we want to introduce a new notation which will allow us describe these simplices more succinctly. Assume that a permutation π of the dimensions, induces a simplex S_π, with vertices $v_0, \dots, v_d$, as defined in Definition 4.2. 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 4.7
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:

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

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

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

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.

We discuss what simplices of the decomposition neighbour 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 4.8 — 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 4.7. □

4.3 Sequences of simplices

In this section we introduce and study *sequences of simplices*. They will be useful as we will argue that paths in the ENDOFLINE instance are sequences of simplices, and latter that the colored orientation of the simplicial decomposition will prevent these paths from forming cycles. We start by defining what we mean by a sequence of simplices.

Definition 4.9 — Sequence of simplices.
A *sequence of simplices*, or *simplicial sequence* is a sequence $(S_i)_{i=1}^k$ of d-dimensional simplices $S_i \in \mathcal{S}$ such that:
(1) $S_{i+1} \not\subset \{S_1, \dots, S_i\}$ for all $i \in \{1, \dots, k-1\}$.
(2) S_i and S_{i+1} share a $d-1$-dimensional face F_i for all $i \in \{1, \dots, k-1\}$.

We want to also introduce formally the notion of *transition faces*, which will represent the face we are *walking through*.

Definition 4.10 — Transition faces.
Let $(S_i)_{i=1}^k$ be a simplicial sequence. We call the sequence $(F_i)_{i=1}^{k-1}$, of $d-1$-dimensional faces given by $F_i = S_i \cap S_{i+1}$, the *transition sequence*. We call the individual F_i *transition faces*.

Now of course we need to discuss how we can add colors. In the following $C \subset \{0, \dots, d\}$, will be a subset of colors. We will use this subset of colors to define restrictions on how sequences of simplices we study should be colored.

Definition 4.11 — Rainbow face.
Let $C \subset \{0, \dots, d\}$ be a subset of colors, and F a face of a simplicial complex $\mathcal{S}$. We say that F is a *rainbow face* for colors C, if all vertices of F are colored with colors from C and all colors in C appear in F.

Let us now restrict these general definitions to the context that we will be studying we will fix $C = \{0, \dots, d-1\}$. We will only be studying sequences where the F_i's are rainbow faces i.e. faces colored with exactly the colors $C = \{0, \dots, d-1\}$. We will call a sequence of simplices with this property a *rainbow simplicial sequence*. We will only be studying valid sequences of simplices that have this property, as these are the sequences of simplices that lines in the ENDOFLINE instances. One property is still missing: the sequences of simplices that reduce to the lines in the ENDOFLINE instances are *maximal* in the sence that they can not be prolongated. This motivates the following definition.

Definition 4.12 — 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.

A special type of line that we can obtain in the ENDOFLINE instances are cycles. These are the result of reducing a special class of simplicial sequences, we will also call *cycles*.

Definition 4.13 — Cycle.
A *cycle of simplices for colors C* is a valid maximal sequence $(S_i)_{i=1}^k$ of simplices $S_i \in \mathcal{S}$ for colors C such that $S_1 \cap S_k$ is a $d-1$ dimensional rainbow face for colors C.

Notice that maximal rainbow simplicial sequences are exactly the sequences of simplices which yield the lines in the ENDOFLINE instance which we reduce SPERNER instances to. We can now discuss how we orient these simplices.

4.4 A side note on super-unique TARSKI instances

TODO.

4.5 Orientation of a the simplicial decomposition

In this section we 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 start by defining what we mean by an orientation of a simplex and then discuss how to extend this to a general simplicial complex.

4.5.1 Orientation of a simplex

Definition 4.14 — Orientation of a simplex.
An *orientation* of a simplex S spanned by the vertices $v_0, \ldots, v_d$ is a choice of a permutation of the vertices $[v_{\pi(0)}, \ldots, v_{\pi(d)}]$.

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 4.15 — 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 4.16
Two orientations σ, τ of a simplex S are equivalent if and only if $\operatorname{sgn}(\sigma) = \operatorname{sgn}(\tau)$, 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:

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

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 $[v_0, \ldots, v_d]$ 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 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:

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

Definition 4.17 — 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 4.18
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 4.2. We can now discuss how we can extend this notion to a general simplicial complex.

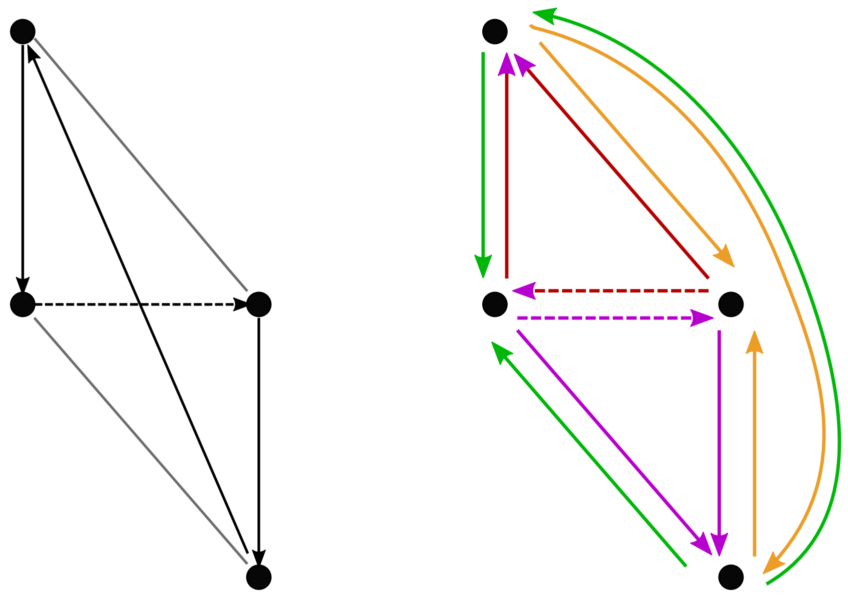

Figure 4.2: Example of the orientation of a 3-simplex on the left, and the induced orientation of the faces on the right.

4.5.2 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 to extend this notion of orientation to these simplicial complexes. Formally we define a simplicial complex as follows [31]:

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

Definition 4.19 — 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.

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 4.20 — 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 4.21
The simplicial complex formed by the lattice points and Freudenthal's simplicial decomposition is orientable.

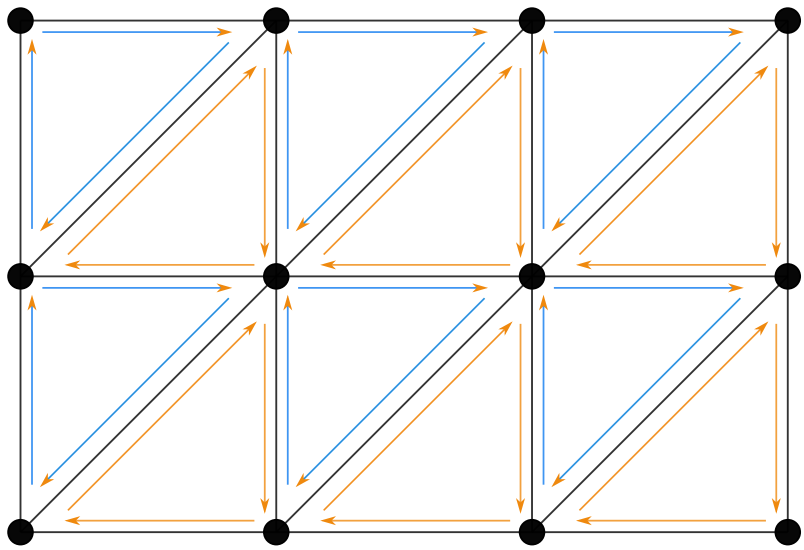

Figure 4.3: Example of the orientation of a Freundenthals simplicial complex in 2 dimensions.

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 4.18.

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:

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

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 4.7 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:

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

We 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 4.3. Now that we have shown that the Freudenthal decomposition is orientable, we want to talk about how it can be extended to colored simplices in the next section.

4.5.3 Orienting colored simplices

In the previous sections we dicussed how one could orient simplices. We now want to discuss the orientation of colored faces. We are only interested in rainbow-faces, i.e. $d-1$-dimensional faces, colored with all colors $\{0,\dots,d-1\}$, as these are the faces that we traverse when reducing SPERNER to ENDOFLINE. We should obtain two orientations for every face, depending on what direction we traverse it in.

Definition 4.22 — Orientation of colored faces.
Let F be a face of a simplex S. Let the vertices of F be labelled $v_1,\dots,v_d$, such that $v_1<\dots<v_d$. Then there is a permutation $\gamma\in S_d$ such that:

$$c(v_{\gamma(1)})<c(v_{\gamma(2)})<\dots<c(v_{\gamma(d)})$$

Then with σ_F the induced orientation of F in S, the orientation of the colored face F is:

$$\mathbf{orient}_S(F)=\mathbf{sgn}\left(\gamma\right)\cdot\mathbf{sgn}\left(\sigma_F\right)$$

Now we should check that our definition is sound. We want that the orientation of a face F is opposite when traversing the face from opposite sides. This is the content of the following lemma.

Lemma 4.23 — Well-definedness of colored orientation.
Let S_1 and S_2 be two d-dimensional simplices which share a $d-1$-dimensional rainbow face F. Then:

$$\mathbf{orient}_{S_1}(F)=-\mathbf{orient}_{S_2}(F).$$

Proof. Because the simplicial complex we work with is orientable, for $\mathbf{indorient}_{S_1}(F)$ the induced orientation of F in S_1 and $\mathbf{indorient}_{S_2}(F)$ the induced orientation of F in S_2, we have:

$$\mathbf{sgn}\left(\mathbf{indorient}_{S_1}(F)\right)=-\mathbf{sgn}\left(\mathbf{indorient}_{S_2}(F)\right)$$

This immediatetly implies the desired result. □

4.6 Properties of colored of oriented simplical sequences

4.6.1 General properties of the coloring

For this section we 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 4.24
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) $i \ge j$ or
(2) $i < j$ and $x_i \not\ge x_j$

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

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

Corollary 4.25
Assume that f is monotone and that we have $x_i, x_j \in S$, for some simplex $S \in \mathcal{S}$. Further assume that $c(x_i) = i$ and $c(x_j) = j$ for $i, j \in \{1, \dots, d\}$ with $i < j$ and that $x_i[i] = x_j[i]$, then $x_i < x_j$.

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

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

4.6.2 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 $C = \{0, \dots, d-1\}$ be the subset of colors, and let $(S_i)_{i=1}^{k}$ be a valid simplicial sequence.

Lemma 4.26
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 $x_{i+1}^j = y$. □

Lemma 4.27 — Orientation of transition faces.
Let $(S_i)_{i=1}^k$ be a valid rainbow simplicial sequence. Then $\left(\textbf{orient}_{S_i}(F_i)\right)_{i=1}^{k-1}$ is constant.

This means that our definition of orientation of colored faces, makes sence: when we walk though these faces, we always walk through faces which are oriented in the same way.

Proof. It suffices to show that for all $i \in \{1, \dots, k-2\}$:

$$\textbf{orient}_{S_i}(F_i) = \textbf{orient}_{S_{i+1}}(F_{i+1})$$

Fix i, and notice that F_i and F_{i+1} are both faces of S_{i+1}. Let us label the vertices of S_i $v_0 < v_1 < \dots < v_d$. Let us denote k and j such that F_i is obtained by removing v_k from S_{i+1} and F_{i+1} is obtained by removing v_j from S_{i+1}. By using Claim 4.18 and noticing that v_k and v_j have the same color we immediately get:

$$-\textbf{orient}_{S_{i+1}}(F_i) = \textbf{orient}_{S_{i+1}}(F_{i+1})$$

Now by using Lemma 4.23 we get:

$$\textbf{orient}_{S_i}(F_i) = -\textbf{orient}_{S_{i+1}}(F_i) = \textbf{orient}_{S_{i+1}}(F_{i+1})$$

This is the desired result and concludes the proof. □

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:

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

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 4.25 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:

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

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:

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

This shows the Lemma. □

Lemma 4.29 — Ordering of the vertices in transition faces.
Let $(S_i)_{i=1}^k$ be a valid rainbow simplicial sequence. For all i such that F_i is a face between cellsthen assume that we are moving in dimension $l \in \{1, \dots, d-1\}$, that is l is not a dimension of F_i. Then for all colors $c \in \{k+1, \dots, d-1\} \cup \{0\}$, we have that c appears after l in F_i. This means that F_i is of the form:

$$(F_i): \quad c(v_1) \to \cdots \to l \to \cdots \to c \to \cdots \to c(v_d)$$

A face between cell is a face of type (2) in Remark 4.7.

Proof. Assume for the sake of contradiction that for $c \in \{k+1, \dots, d-1\} \cup \{0\}$ we have that c appears before l in F_i. Then by Corollary 4.25 we must have that we move in dimension l between l and c. But the dimension l is not a dimension of F_i. This is a contradiction and shows the Lemma. □

4.7 No cycles in the EndOfLine instance

Remark 4.30
Assume that we have a valid colored simplicial sequence which crosses a hyperplane obtained by fixing one dimension $H = L_{k=K}$, for a dimension $k \in \{1, \dots, d\}$ and $K \in [N]$, twice. Let

F_i and F_j be the transition faces which cross the hyperplane. Then:

$$\mathbf{orient}_{S_i}(F_i) = -\,\mathbf{orient}_{S_{j+1}}(F_j)$$

This means that when looking at the hyperplane from any side, F_i and F_j have opposite orientations.

Proof. This is a reformulation of the more general Lemma 4.27. □

Now we are ready for the proof of our main result. We will start by proving that there are no cycles in the three dimensional case, and then extend this to the general case.

Theorem 4.31 — No cycles in three-dimensional TARSKI*. There are no cycles in the ENDOFLINE instance which we reduce three dimensional TARSKI* instances to.

Proof. Assume for the sake of contradiction that we have a cycle in the ENDOFLINE instance. Let $(S_i)_{i=1}^{k}$ be a cycle of simplices for colors $C = \{0, 1, 2\}$, and $(F_i)_{i=1}^{k}$ be the rainbow transition faces. We will show that this leads to a contradiction.

Consider the faces L_i^c which are the faces spanned by vertices colored not c in S_i. Formally:

$$L_i^c = \{v \in S_i \mid c(v) \neq c\} \quad \text{for } c \in C$$

Notice that these L_i^c's and are always either 2-dimensional or 1-dimensional simplices. Now for the sake of simplicity assume that we remove all 1-dimensional edges from these sequences, and only consider the 2-dimensional faces. Notice that this is in itself a valid oriented simplicial sequence, and in particular a cycle.

This means that we have three cycles of faces. Now for each of these face sequences we look at the transion edges $(Q_i^c)_i$ of $(L_i^c)_i$. We then have that by Lemma 4.27 that the orientations of the sequence $(Q_i^c)_i$ is constant.

Recall that this means that:

$$Q_i^c = L_i^c \cap L_{i+1}^c$$

We will first argue that $(S_i)_{i=1}^{k}$ cannot only move in two dimensions. Assume for the sake of contradiction that this is the case. TODO: Write this argument, maybe as a seperate Lemma.

This means that any cycle must move in all 3 dimensions and in particular cross a hyperplane H_1 obtained by fixing dimension 1, and a hyperplane H_2 obtained by fixing dimension 2 at least twice. Let F_i and F_j be the transition faces which cross the hyperplane H_1 and M_i and M_j be the transition faces which cross the hyperplane H_2. Then by the remark above we have that:

$$\mathbf{orient}_{S_i}(F_i) = -\,\mathbf{orient}_{S_{j+1}}(F_j) \quad \text{and} \quad \mathbf{orient}_{S_i}(M_i) = -\,\mathbf{orient}_{S_{j+1}}(M_j)$$

This means that when looking at these hyperplanes from any side, the transition faces have opposite orientations.

Now first notice, that because we have fixed dimension 1, in both F_i and F_j, we must have that the vertex colored 0 is larger than the vertex colored 1, by Corollary 4.25. Now of course these two eges are on the cycle $(L_i^2)_i$,

Case 1: If we move in dimension 1 then, let H be a hyperplane obtained by fixing dimension 1 which we cross twice. Then let Q_i and Q_j be the transition faces which cross the hyperplane. Then by the remark above we have that: $\mathbf{orient}_{S_i}(Q_i) = \mathbf{orient}_{S_j}(Q_j)$, which means that for either Q_i or Q_j, we have that the vertex colored 1 is larger than the vertex colored 2. This is not possible because we do not move in dimension 1 between these two vertices. This contradicts monotonicity by Corollary 4.25.

Case 2: If we move in dimensions 2 then, let H be a hyperplane obtained by fixing dimension 2 which we cross twice. Then let M_i and M_j be the transition faces which cross the hyperplane. Then by the remark above we have that: $\mathbf{orient}_{S_i}(M_i) = \mathbf{orient}_{S_j}(M_j)$, which means that for either M_i or M_j, we have that the vertex colored 2 is larger than the vertex colored 0. This is not possible because we do not move in dimension 2 between these two vertices. This contradicts monotonicity by Corollary 4.25

Because we will always move in at least dimension 1 or 2 this concludes the proof. □

4.8 Discussing the reduction of TARSKI* to ENDOFPOTENTIALLINE

We have now shown that there are no cycles in the ENDOFLINE instance which arise from monotone TARSKI* instances. This does not directly imply a direction onto ENDOFPOTENTIALLINE. Inuitively an ENDOFLINE instance without cycles should be equivalent to an ENDOFPOTENTIALLINE instance. But in ENDOFPOTENTIALLINE the non-existence of a cycle is given by a potential which grows monotonically along the lines.

The challenge in our case is to define such a potential. We briefly want to give some inuition for why this is not easy. A first naive attempt, would be to define the potential as something similar to the sum of the coordinates of a vertex of the simplex. This will not work in our case though, because it can very well happen that the path though the simplicial complex is not monotone with respect to the sum of the coordinates, in particular we can cross a hyperplane multiple times, which would not be possible if the sum of the coordinates was a valid potential.

Recall that this is exactly the same potential as we used to reduce TARSKI onto LOCALOPT.

This means that we need to be more creative in our definition of the potential. The idea is to decide on how we orient our potential locally. We start by giving every cell of the simplicial complex coordinates, these will be the coordinates of the smallest vertex of cell. Now look at the colors of the vertices of the cell.

Appendix

Bibliography

[1] Christos H. Papadimitriou. 'On the complexity of the parity argument and other inefficient proofs of existence'. In: *Journal of Computer and System Sciences* 48.3 (June 1994), pp. 498–532. DOI: 10.1016/S0022-0000(05)80063-7. (Visited on 03/05/2024) (cited on pages 2, 13).

[2] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. 'How easy is local search?' In: *Journal of Computer and System Sciences* 37.1 (Aug. 1988), pp. 79–100. DOI: 10.1016/0022-0000(88)90046-3. (Visited on 03/06/2024) (cited on pages 2, 12).

[3] John Fearnley et al. *End of Potential Line*. Apr. 18, 2018. URL: http://arxiv.org/abs/1804.03450 (visited on 03/02/2024) (cited on pages 2, 14).

[4] Alfred Tarski. 'A lattice-theoretical fixpoint theorem and its applications.' In: *Pacific Journal of Mathematics* 5.2 (Jan. 1, 1955), pp. 285–309 (cited on pages 2, 15).

[5] Kousha Etessami et al. 'Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria'. In: (2020). In collab. with Thomas Vidick. DOI: 10.4230/LIPICS.ITCS.2020.18. (Visited on 02/24/2024) (cited on pages 2, 3, 15, 17, 19, 20, 23, 24).

[6] Donald M. Topkis. 'Equilibrium Points in Nonzero-Sum *n* -Person Submodular Games'. In: *SIAM Journal on Control and Optimization* 17.6 (Nov. 1979), pp. 773–787. DOI: 10.1137/0317054. (Visited on 07/31/2024) (cited on page 3).

[7] Paul Milgrom and John Roberts. 'Rationalizability, Learning, and Equilibrium in Games with Strategic Complementarities'. In: *Econometrica* 58.6 (Nov. 1990), p. 1255. DOI: 10.2307/2938316. (Visited on 07/31/2024) (cited on page 3).

[8] Anne Condon. 'The complexity of stochastic games'. In: *Information and Computation* 96.2 (Feb. 1992), pp. 203–224. DOI: 10.1016/0890-5401(92)90048-K. (Visited on 07/31/2024) (cited on page 3).

[9] Bernd Gärtner, Sebastian Haslebacher, and Hung P. Hoang. 'A subexponential algorithm for ARRIVAL'. In: (2021). In collab. with Bansal, Nikhil, Merelli, Emanuela, and Worrell, James. Artwork Size: 14 p. Medium: application/pdf Publisher: ETH Zurich, 14 p. DOI: 10.3929/ETHZ-B-000507056. (Visited on 07/30/2024) (cited on page 3).

[10] 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 pages 3, 18).

[11] Xi Chen and Yuhao Li. 'Improved Upper Bounds for Finding Tarski Fixed Points'. In: *Proceedings of the 23rd ACM Conference on Economics and Computation*. EC '22: The 23rd ACM Conference on Economics and Computation. Boulder CO USA: ACM, July 12, 2022, pp. 1108–1118. DOI: 10.1145/3490486.3538297. (Visited on 07/30/2024) (cited on pages 3, 19).

[12] Mika Goos et al. 'Further Collapses in TFNP'. In: (2022) (cited on pages 4, 21).

[13] Sanjeev Arora and Boaz Barak. *Computational complexity : a modern approach*. Cambridge: Cambridge University Press, 2009 (cited on page 5).

[14] 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 pages 6, 12).

[15] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. 'The Complexity of Computing a Nash Equilibrium'. In: *SIAM Journal on Computing* 39.1 (Jan. 2009), pp. 195–259. DOI: 10.1137/070699652. (Visited on 03/11/2024) (cited on pages 6, 13).

[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 pages 6, 21).

[17] Alexandros Hollender. 'Structural Results for Total Search Complexity Classes with Applications to Game Theory and Optimisation'. PhD thesis. Oxford University, 2021 (cited on page 7).

[18] 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 pages 8, 9).

[19] Christos H. Papadimitriou. *Computational complexity*. Reading (Mass): Addison-Wesley, 1994 (cited on page 12).

[20] Michael Sipser. 'On relativization and the existence of complete sets'. In: *Automata, Languages and Programming*. Ed. by Mogens Nielsen and Erik Meineche Schmidt. Vol. 140. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 523–531. DOI: 10.1007/BFb0012797. (Visited on 07/30/2024) (cited on page 12).

[21] 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 13).

[22] 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 19).

[23] 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 20, 23).

[24] 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 21).

[25] Xi Chen, Yuhao Li, and Mihalis Yannakakis. 'Reducing Tarski to Unique Tarski (In the Black-Box Model)'. In: *LIPIcs, Volume 264, CCC 2023* 264 (2023). In collab. with Amnon Ta-Shma. Artwork Size: 23 pages, 1766105 bytes ISBN: 9783959772822 Medium: application/pdf Publisher: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 21:1–21:23. DOI: 10.4230/LIPICS.CCC.2023.21. (Visited on 07/30/2024) (cited on page 22).

[26] 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 23).

[27] 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 23, 27).

[28] 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 23, 26, 30).

[29] 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 27).

[30] 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 38).

[31] James Raymond Munkres. *Elements of algebraic topology*. The advanced book program. Boca Raton London New York: CRC Press, 2018. 454 pp. (cited on page 46).

Alphabetical Index