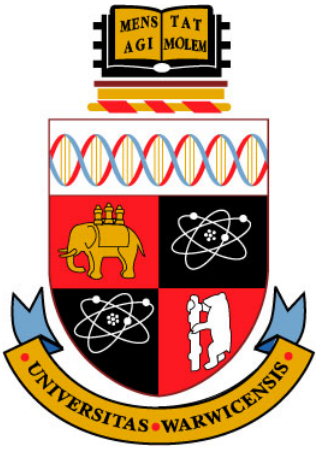

Counting Complexity of PureCircuit -1

CS907 Dissertation Project

Christos Demetriou 2018918

Supervisor: Dr. Christian Ikenmeyer

Department of Computer Science

University of Warwick

2024-2025

Abstract

When this thing gets up to 88 mph, you're gonna see some serious s· · ·.

Keywords: Flux Capacitor, 1.21 Gigawatts, Calvin Klein

Acknowledgements

Always good to acknowledge people.

Abbreviations

Pure Circuit	PC

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

$[n]_0$ Defined as $[n] \cup \{0\}$.

$[n]$ List of numbers $1, ..., n$.

$\mathbb{N}_0$ Natural numbers including 0.

$\mathbb{N}$ Set of natural numbers excluding 0.

$\mathbb{T}$ Defines the set $\{0, 1, \bot\}$ which depicts the kleene algebra..

$n^{O(1)}$ Set of all functions $f : \mathbb{N} \to \mathbb{N}$, where $f \in O(n^c)$ for some $c \in \mathbb{N}$.

$x_{-i}, x_{-i}(a)$ Given $x \in A^d$, we use x_{-i} to denote all members excluding index i. To show that we replace the index i with y, we say $x_{-i}(a)$.

1 Introduction

2 Theoretical Preliminaries

Our work lies in the intersection of the three core fields: counting complexity, total search problems and Kleene logic.

2.1 Search problems

We offer brief preliminaries to function problems and the different kinds that exist. First and foremost when

Definition 2.1: Search Problems

Search problems can be defined as relations $R \subseteq \mathbb{B}^* \times \mathbb{B}^*$, where given $x \in \mathbb{B}^*$, we want to find $y \in \mathbb{B}^*$ such that xRy.
Total Search problems are search problems such that for each input, there must exist at least one solution.

Like decision problems, we can define their search analogues using the definitions in 2.1. Moreover, we use the *Levin reductions* 2.1 to associate two problems.

Definition 2.2: FNP **and** TFNP

FNP are *search problems* such that there exists poly-time TM $M : \mathbb{B}^* \to \mathbb{B}$ and a poly function $p : \mathbb{N} \to \mathbb{N}$ such that:

$$\forall x \in \mathbb{B}^*, y \in \mathbb{B}^{p(|x|)} : xRy \iff M(x, y) = 1$$

Lastly $\mathbf{TFNP} = \{L \in \mathbf{FNP} \mid L \text{ is total}\}$

Definition 2.3: Levin Reductions

Given two search problems R, R', a pair of poly-time computable functions f, g where $f, g : \{0,1\}^* \to \{0,1\}^*$, is called a *Levin reduction*, if

$$S_R \triangleq \{x \mid \exists y : xRy\}$$
$$R(x) \triangleq \{y \in \{0,1\}^* \mid xRy\}$$
$$\forall x \in \{0,1\}^* : x \in S_R \implies f(x) \in S_{R'}$$
$$\forall x \in S_R, y' \in R'(f(x)) : (x, g(x, y)) \in R$$

Within the current project, we are mainly interested with the total search problems or at least a specific hierarchy, as shown in figure **??**. These problems were introduced by various researchers as seen in , and all such classes the guarantee of an existence of a solution by using some combinatorial property of the nature of the problem but can be very difficult to find. In the current project we are mainly interested in **PPAD**, known as "Polynomial Parity of Augmented DiGraphs", created by Papadimitriou [28]. We will give a formal definition of the class as well as several cornerstone problems that we will refer to in the upcoming chapters.

We will first introduce the *EndOfLine* problem 2.1by Papadimtirou [28], which takes advantage of the combinatorial fact of directed graphs where a pair of odd-degree vertices come in pairs.

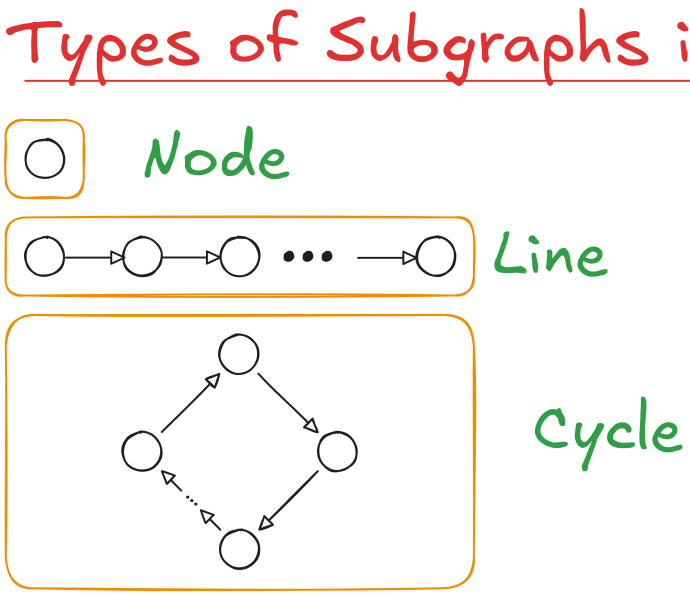

Figure 2.1: Types of subgraphs in EndOfLine

Definition 2.4: *EndOfLine* problem [28]

Given poly-sized circuits $S, P \in \mathbb{B}^n \to \mathbb{B}^n$, we define a digraph $G = (V, E)$, such that $V = \mathbb{B}^n$ and E defined as:

$$E = \{(x, y) \in V^2 : S(x) = y \wedge P(y) = x\}$$

We define source nodes as $v \in V : deg(v) = (0, 1)$ and sink nodes as $deg(v) = (1, 0)$. We also syntactically ensure that the 0^n node is always a source, meaning $S(P(0^n)) \neq 0 \wedge P(S(0^n)) = 0^n$. A node $v \in V$ is a solution *iff* $in\text{-}deg(v) \neq out\text{-}deg(v)$.

An illustrative example of an EndOfLine instance can be seen in figure 2.1. From this problem, we define the PPAD complexity class 2.5.

Definition 2.5: PPAD complexity class [28]

PPAD is defined as the set of search problems that are *levin reducible* 2.1 to the EndOfLine problem 2.1.

2.1.1 EndOfLine variants and extensions

We want introduce a small number of the several key variants of the *EndOfLine* that have been created in the past. Alexandros et al. in the paper was able to demonstrate that for several modifications of the original problem such as adding k known sources, where k can be a polynomial function with respect to the number of sources. Moreover, people have experimented with changing the degrees of the graphs, as demonstrated by Ikenmeyer et al. [21], with the problems such as SourceOrExcess$(k, 1)$ where k it the maximum in-degree for each node. All these problems remain **PPAD** complete. More interestingly, Papadimitriou et al. and Alexandros et al. were able to demonstrate that in the more general case of superpolynomial graphs, if we bound the in-degree and out-degree of each node to some polynomial of the input size, then problem remains **PPAD**. We will give the main definitions to the most relevant problems in the upcoming sections.

Definition 2.6: SourceOrExcess **problem [21]**

We define as *SourceOrExcess*$(k, 1)$ for $k \in \mathbb{N}_{\geqslant 2}$ the search problem as such: Given a poly-sized successor circuit $S : \mathbb{B}^n$ and a set of predecessor poly-sized circuits $\{P_i\}_{i\in[k]}$, we define the graph $G = (V, E)$ such that, $V = \mathbb{B}^n$ and E as:

$$\forall x, y \in V : (x, y) \in E \iff (S(x) = y) \wedge \bigvee_{i\in[k]} P_i(y) = x$$

We ensure that 0^n is as source, meaning $\deg(0^n) = (0, 1)$. A valid solution is a vertex v such that *in-deg*$(v) \neq$ *out-deg*(v).

In addition to the aforementioned adjustments, researchers were able to give a topological interpretation to the EndOfLine 2.1 problem, by creating embeddings of the graph into 2D or 3D spaces as demonstrated by several researchers deligkas, chen, Alexandros. We will mainly look at two of these problems, known as RLeafD and 2D-EOL created by Chen et al. and Alexandros respectively.

Definition 2.7: 2D-EOL **problem**

Input: Two poly-sized circuits $P, S : \{0, 1\}^{2n} \to \{0, 1\}^{2n}$ where $P(0^n, 0^n) = (0^n, 0^n) \neq S(0^n, 0^n)$. **Output**: A point $x \in \{0, 1\}^{2n}$ such that one of the following conditions are satisfied:

1. Sinks: $P(S(x)) \neq x$
2. Sources: $S(P(x)) \neq x \wedge x \neq 0^n$
3. Invalid successor: $x - S(x) \notin \{(0, 0), (0, 1), (0, -1), (1, 0), (-1, 0)\}$
4. Invalid predecessor: $x - P(x) \notin \{(0, 0), (0, 1), (0, -1), (1, 0), (-1, 0)\}$

The above description defines a superpolynomial such that an edge $(x, y) \in E$ if and only if $S(x) = y \wedge P(y) = x$ as well as the closeness constraint $\|x - S(x)\|_1 \leqslant 1$. We will now refer to a special case of 2D-EOL known as RLeafD by Chen et al. where they give a planar embedding of any graph G onto a 2D plane. We find this construction useful for our results and therefore will investigate this method further.

Planar Graph Embedding We define the graph embedding of a graph $G = (V, E)$ as $G_n = (V_n, E_n)$ where $|V| = n$ and V_n is defined as

$$\forall n \in \mathbb{N} : V_n \triangleq \left\{\mathbf{u} \in \mathbb{N}_0^2 \mid \mathbf{u}_1 \leqslant 3(n^2 - 1), \mathbf{u}_2 \leqslant 6(n - 1) + 3\right\}$$

G can have any in- and out-degree. For a vertex $i \in V$, we correspond it to the node $(0, 6i)$ in the embedded version. To define the edge set, we will first introduce the following notations.

Definition 2.8: $E(\mathbf{p}^1, \mathbf{p}^2)$

Given two points $\mathbf{p}^1, \mathbf{p}^2 \in V_n$ we define the set of edges $E(\mathbf{p}^1, \mathbf{p}^2)$ as:

$$E(\mathbf{p}^1, \mathbf{p}^2) \triangleq \begin{cases} \{(\alpha, k) \mid k \in \mathbb{N} : \min\{\mathbf{p}_2^1, \mathbf{p}_2^2\} \leqslant k \leqslant \max\{\mathbf{p}_2^1, \mathbf{p}_2^2\}\} & \text{if } \mathbf{p}_1^1 = \mathbf{p}_1^2 = \alpha \\ \{(k, \alpha) \mid k \in \mathbb{N} : \min\{\mathbf{p}_1^1, \mathbf{p}_1^2\} \leqslant k \leqslant \max\{\mathbf{p}_1^1, \mathbf{p}_1^2\}\} & \text{if } \mathbf{p}_2^1 = \mathbf{p}_2^2 = \alpha \\ \varnothing & \text{otherwise} \end{cases}$$

Definition 2.9: E_{ij}

Given an edge of the original graph $(i, j) \in E$, we define the set of edges E_{ij} as:

$$\begin{aligned}
\mathbf{p}^1 &= (0, 6i) \\
\mathbf{p}^2 &= (3(ni + j), 6i) \\
\mathbf{p}^3 &= (3(ni + j), 6j + 3) \\
\mathbf{p}^4 &= (0, 6j + 3) \\
\mathbf{p}^4 &= (0, 6j) \\
E_{ij} &\triangleq E(\mathbf{p}^1\mathbf{p}^2) \cup \cdots \cup E(\mathbf{p}^4\mathbf{p}^5)
\end{aligned}$$

To formally define E_n, we first split E_n into two disjoint sets E_n^1 and E_n^2. We define E_n^1 as $\bigcup_{ij \in E} E_ij$. The graph generated by (V_n, E_n) has some special properties such as, $\forall v \in V_n$: in-deg(v) + out-deg$(v) \leqslant 4$ and $\forall v \in V_n$: deg$(v) = 4 \implies$ in-deg$(v) = 2 =$ out-deg(v). For every 4-degree vertex, we add 4 additional edges as depicted in the figure 2.2 depicted in blue and add them to E_n^2.

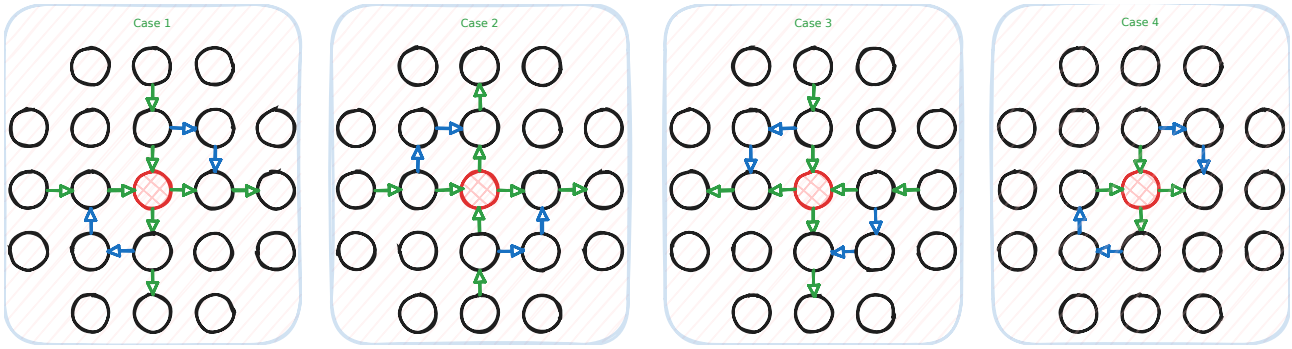

Figure 2.2: Figure redrawn from [7]

Given the above we have a full description of G_n. A visual explanation can be seen in figure 2.3 and an example of the K_3 complete graph in figure 2.4

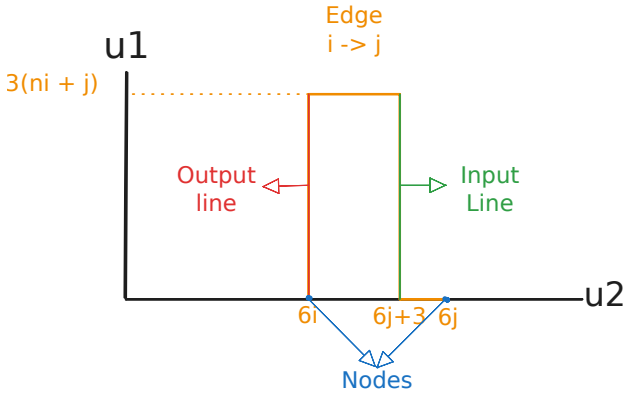

Figure 2.3: Edge between points i, j of the original graph. Main observation is the fact that each point comes with a pair of exponentially long lines, which we refer to as the input output lines.

RLEAFD **problem** We denote C_n as the set of planar graphs $G_n = (V_n, E_n)$ such that $\forall \mathbf{u} \in V_n$: *in-deg*$(v) \leqslant 1 \wedge$ *out-deg*$(v) \leqslant 1$. From that we define the class of problems RLEAFD as in

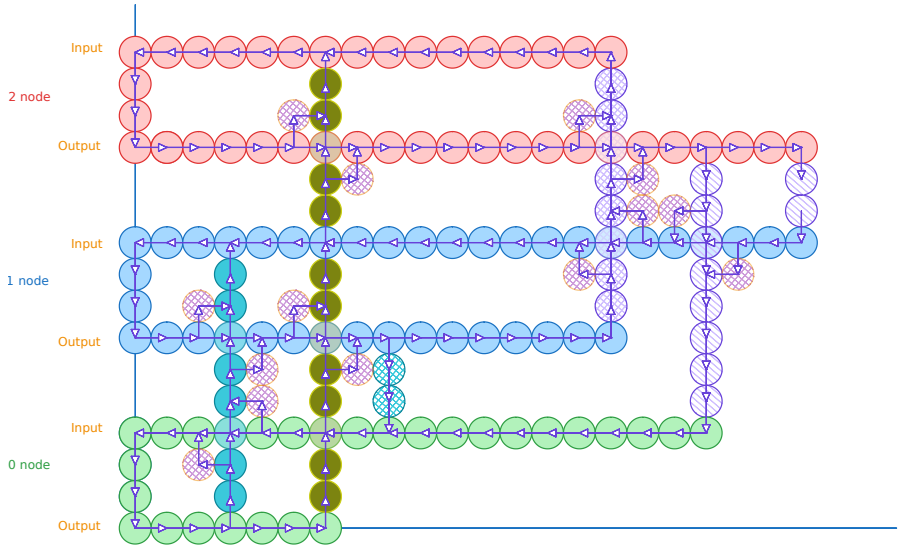

Figure 2.4: K_3 graph construction. Combinations of colours indicate an edge between the two nodes. Solid dots indicate edges from 0, crossed lines indicate edges from 1 and stripped lines indicate edges from 2. We Add two coloured stripped circles on the 4 degree nodes to indicate the E_n^2 edges.

definition 2.1.1.

Definition 2.10: RLeafD problem [7]

Input: A tuple $(K, 0^k)$ such that $k \in \mathbb{N}$ and K is a poly-time TM which describes a graph in $G \in C_n$. More specifically, K is a function $V_{2^k} \to V_{2^k} \cup \{\perp\}$ such that

$$(a, b) \in E_n \iff K(a) = (\cdot, b) \wedge K(b) = (a, \cdot)$$

Output: $v \in V_{2^k} \backslash \{0^n\}$ such that $\exists a \in V_{2^k} : K(v) = (\cdot, a) \vee K(v) = (a, \cdot)$

Embedding EndtofLine Chen et al. [7] demonstrated how every EndOfLine can be reduced to a RLeafD problem. Key insight, is the observation that every planar embedding of a graph of an EndOfLine has the following property:

$$\forall v \in V_n : deg(v) \in \{(0,0), (1,1), (1,0), (0,1), (2,2)\}$$

Using as reference the figure 2.3, we remove all the edges of nodes with degree of $(2, 2)$. This in a way creates a different graph but the key insight is that the number of sources or sinks remain the same as we are essentially redirecting the lines as we can see from the figure 2.5. We will use these reductions in later sections to demonstrate reductions with excess problems as well.

The PureCircuit problem The *PureCircuit* problem is a discretised version of the ϵ-*GCircuit* which was first introduced by Chen et al. [14, 8]. General idea of ϵ-*GCircuit* is that we have a directed graph $T = (V, G)$ where V denote value nodes and G denote gate nodes. Each value node is assigned a value in $[0, 1]$ and the gates denote can denote the standard set of arithmetic operations $\{+, -, *\}$ with other nodes or some constant, or boolean operators $\{\wedge, \vee, \neg, <, >, =\}$ with other nodes or some constant. The goal of the problem is to assign every problem with a value such that the assingment is correct up to a factor of $\pm\epsilon$. This problem was crucial to demonstrate *PPAD*-hardness for constant $\epsilon > 0$ by Rubinstein [14, 29].

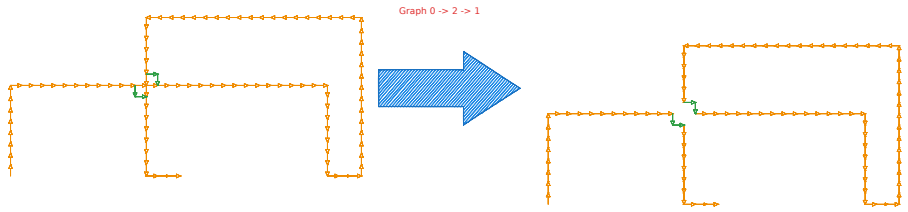

Figure 2.5: Depiction of the edge removal. Main idea is that this we keep the number of sources or sinks as well as invertible (we can trace the source or sink from the original instance), despite losing a lot of graph information.

PureCircuit is the problem that we get when taking the limit $\epsilon \to \infty$ of the ϵ-*GCircuit* problem. This results in a circuit which use the *Kleene valued logic*which extends the traditional binary logic [24]. We will explore the above algebra in greater depth in section ... This problem was created by Deligkas et al. [14] to demonstrate the hardness of approximating PPAD problems and was shown to be PPAD-COMPLETE.

Definition 2.11: PURECIRUIT **Problem Definition [14]**

An instance of *PureCircuit* is given by vertex set $V = [n]$ and gate set G such that $\forall g \in G : g = (T, u, v, w)$ where $u, v, w \in V$ and $T \in \{\text{NOR}, \text{Purify}\}$. Each gate is interpreted as:

1. *NOR*: Takes as input u, v and outputs w
2. *Purify*: Takes as input u and outputs v, w

And each vertex is ensured to have in-deg$(v) \leqslant 1$. A solution to input instance (V, G) is denoted as an assignment $\mathbf{x} : V \to \{0, \bot, 1\}$ such that for all gates $g = (T, u, v, w)$ we have:

1. *NOR*:

$$\begin{gathered}\mathbf{x}[u] = \mathbf{x}[v] = 0 \implies \mathbf{x}[w] = 1 \\ (\mathbf{x}[u] = 1 \vee \mathbf{x}[v] = 1) \implies \mathbf{x}[w] = 0 \\ \text{otherwise} \implies \bot\end{gathered}$$

2. *Purify*:

$$\begin{gathered}\forall b \in \mathbb{B} : \mathbf{x}[u] = b \implies \mathbf{x}[v] = b \wedge \mathbf{x}[w] = b \\ \mathbf{x}[u] = \bot \implies (\mathbf{x}[v], \mathbf{x}[w]) \in \{(0, \bot), (\bot, 1), ((0, 1))\}\end{gathered}$$

In addition to the gates above, we will make use of the standard set of gates $\{\vee, \wedge, \neg\}$, whose behaviour is depicted in the tables 2.1. Moreover, we will make use of the *Copy* gate which we can define as $\mathit{Copy}(x) = \neg(\neg x)$. These gates are well defined based on the *NOR* gate as shown by Deligkas et al. [14] and do not directly affect the complexity of the problem. Furthermore, with respect to the *Purify* gate, the original problem states that one of two outputs must contain a pure value. We restrict the solution set to the one written in the definition as Deligkas et al. [14] showed that the only *Purify* solutions essential for **PPAD**-completeness are $\{(0, \bot), (\bot, 1), (0, 1)\}$, and the addition of more solutions does not affect its complexity.

2.1.2 Topological problems

This section refers to types of problems which take advantage of some combinatorial property of their topology in order to guarantee a solution.

not	
0	1
⊥	⊥
1	0

(a) `not` gate

and	0	⊥	1
0	0	0	0
⊥	0	⊥	⊥
1	0	⊥	1

(b) `and` gate

or	0	⊥	1
0	0	⊥	1
⊥	⊥	⊥	1
1	1	1	1

(c) `or` gate

Table 2.1: Three-valued logic [24]

Sperner Problems Below we will refer to the notion of Sperner problems, which are types of problems that are based around *Sperner's Lemma*. The idea is as follows, given a triangle that has subdivided into smaller subtriangles, we first define the index of each point using the following set:

$$\forall n \in \mathbb{N} : \mathcal{T}_n \triangleq \{(x, y, z) \in [n] \mid x + y + z = n\}$$

We say $\lambda : \mathcal{T}_\setminus \to [3]$ is a valid colouring if and only if it satisfies the following conditions:

$$\forall (x_1, x_2, x_3) \in \mathcal{T}_n : \lambda(x) \in \begin{cases} \{2, 3\} & \text{if} x_1 = 0 \\ \{1, 3\} & \text{if} x_2 = 0 \\ \{1, 2\} & \text{if} x_3 = 0 \end{cases}$$

Given a valid colouring λ, we can always find a small triangle τ where all 3 colours are assigned to one of its vertices. A visual example can be seen in figure 2.6.

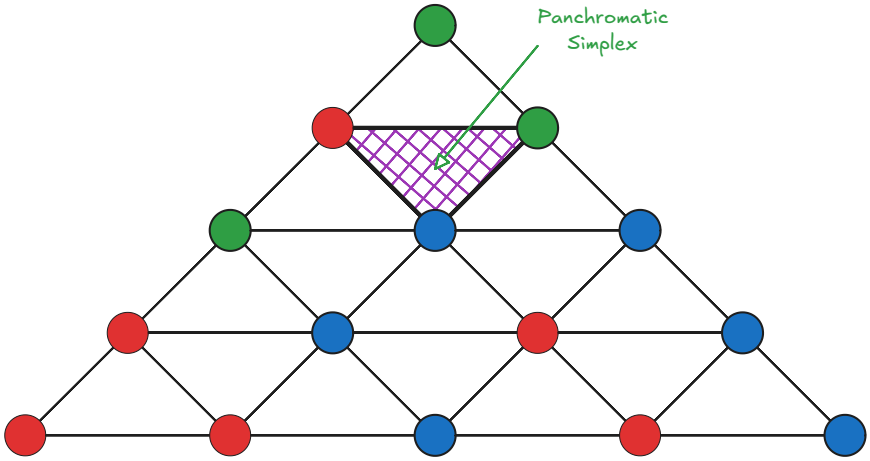

Figure 2.6: A colouring example of a subdivided simplex. As we can observe, no matter how we modify colourings, there will always be a small simplex that contains all colours

This type colouring will be referred to as the **linear** colouring where for dimension d, assign $d + 1$ distinct colours to each point [10, 7], and it was shown that one can define a **PPAD**-complete variant of this lemma property. There several variants of the problem, where we can either work in a triangle with an exponential amount of subdivisions or polynomial subdivision but on higher dimensional simplexes. Both of these problems are **PPAD**-complete.

We introduce an alternative variant of the Sperner problems, introduced by Daskalakis et al. [12], known as *StrongSperner* or *BiSperner*. Instead of assigning $d + 1$ colours to find a panchromatic simplex, each vertex is assigned d colours rather than one. We refer to the figure for a visual interpretation of the colouring 2.7. Essentially for each point in some hypercube $[N]^d$, each vertex is assigned d colours, where for colour i have choices either $\{i^+, i^-\}$. The goal is to find a cube where in every dimension we have both colours. This has been used in many

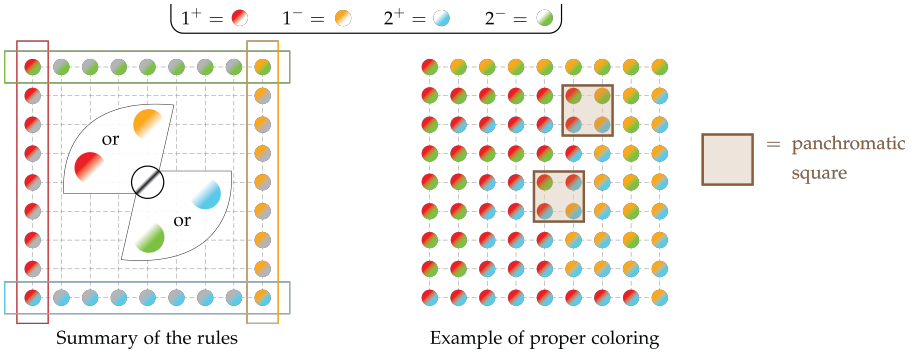

Figure 2.7: Visual interpretation of the *StrongSperner* problem. Figure from paper [11]

problems throughout literature as seen in the papers [8, 14, 12] to demonstated *PPAD*-hardness of many problems.

We will refer to this type of colouring as **bipolar** colouring, which has been used in grid-like topologies of the Sperner property We note that these terms are not standard in literature; we adopt this naming convention for of clarity.

Definition 2.12: Bipolar colouring

Given a set S^d, where S is some arbitrary set, we refer to bipolar colouring C as:

$$\forall v \in S^d, j \in [d] : [C(v)]_j \in \{-1, 1\}$$

We say that a set of points $A \subseteq S^d$ **cover all the labels** if:

$$\forall i \in [d], \ell \in \{-1, +1\}, \exists x \in A : [\lambda(x)]_i = \ell$$

Deligkas et al. [14, 13] showed that $\forall A \subseteq S^d : |A| \geqslant d + 1$ and satisfies the colouring, $\exists D \subseteq A : |D| \leqslant d$ where D also satisfies the colouring condition.

Definition 2.13: StrongSperner problem

Input: A boolean circuit that computes a bipolar labelling $\lambda : [M]^N \to \{-1, 1\}^N$ 2.12 satisfying the following boundary conditions for every $i \in [N]$:

- if $x_i = 1 \implies [\lambda(x)]_i = +1$
- if $x_i = M \implies [\lambda(x)]_i = -1$

Output: A set of points $\{x^{(i)}\}_{i \in [N]} \subseteq [M]^N$, such that:

- *Closeness condition*: $\forall i, j \in [N] : \|x^{(i)} - x^{(j)}\|_\infty \leqslant 1$
- *Covers all labels* as defined in 2.12

The above is a generalised variant of the traditional Sperner problem to a grid of dimensions N with width of M. Throughout literature the same variants of the problem or specifications have been defined using the names Sperner or discrete Brouwer [8, 7, 10, 14]. This is very similar to a different type of problem known as *StrongTucker*, which can be thought of as the *PPA* analog of *StrongSperner*. The two problems satisfy different boundary conditions but due to their similar labelling nature, we will borrow some techniques used in later sections. For clarity, the subsequent analysis adopts *StrongSperner*.

Definition 2.14: nD-StrongSperner **problem**

Input: A tuple $(\lambda, 0^k)$ of a StrongSperner instance but for only n dimensions, such that $\lambda : (\mathbb{B}^k)^n \to \{-1, +1\}^n$.
Output: A point $\alpha = (a_1, \ldots, a_n) \in (\mathbb{B}^k \backslash \{1^k\})^n$ such that

$$\{\alpha + x \mid x \in \mathbb{B}^n\} \text{ covers all the labels 2.12}$$

We assume dimensionality $n \geqslant 2$.

The authors refer to both colouring schemes interchangeably when discussing their reductions, so we can assume that all reductions (not necessarily counting reductions) work similarly for either colouring [8, 14, 10, 7].

Other than topological colouring, we will also give a small introduction, to a tiling problem known as mutilated chessboards. Tiling spaces has been used throughout literature [18, 5]. We will use as reference the **PPAD**-complete 1-MC problem. We refer to the problem introduced by Blank [2] as follows: assume we have a $2n \times 2n$ chessboard. If we are provided with 1×2 pieces, we can trivially tile the board by filling each row with n horizontal pieces. The idea is when we remove the bottom left corner, the board becomes impossible to tile. The argument can be thought of as such: each tile piece covers a bright square and an odd square, which implies that it has to be the case that one of the squares has to remain untilled. An example can be seen in the figure 2.8. More formally we use the following definition 2.1.2, and Hollender et al. [18] demonstrated that this problem

Definition 2.15: 1-MC **[18]**

Input: Given a circuit $C : 2^{2n} \to 2^{2n}$, such that $C(0,0) = (0,0)$, such that tiling between two squares is indicated such that:

$$\forall x \in \{0,1\}^{2n} : C(C(x)) = x \wedge x - C(x) \in \{(0,1), (1,0), (0,-1), (-1,0)\}$$

Output: We want to find a point $x \in \{0,1\}^{2n} \backslash \{(0,0)\}$ such that we have no tiling.

Lastly, we will introduce Tarski 2.17 which uses the Knaster-Tarski fixed point theorem 2.1 and belongs in PLS $\cap$ PPAD, where PLS is a class of problems based on the idea of local search [23].

Definition 2.16: Monotone functions

Given two partially ordered sets $(L_1, \preceq_{L_1})$ and $(L_2, \preceq_{L_2})$, a function $f : L_1 \to L_2$ is **monotone** if and only if:

$$\forall x, y \in L_1 : x \preceq_{L_1} y \implies f(x) \preceq_{L_2} f(y)$$

Theorem 2.1: Knaster Tarski Fixed point theorem[3, 16]

Given a *monotone* function $f : L \to L$ 2.16 of a complete lattice $(L, \wedge, \vee)$, $\exists c \in L : f(c) = c$.

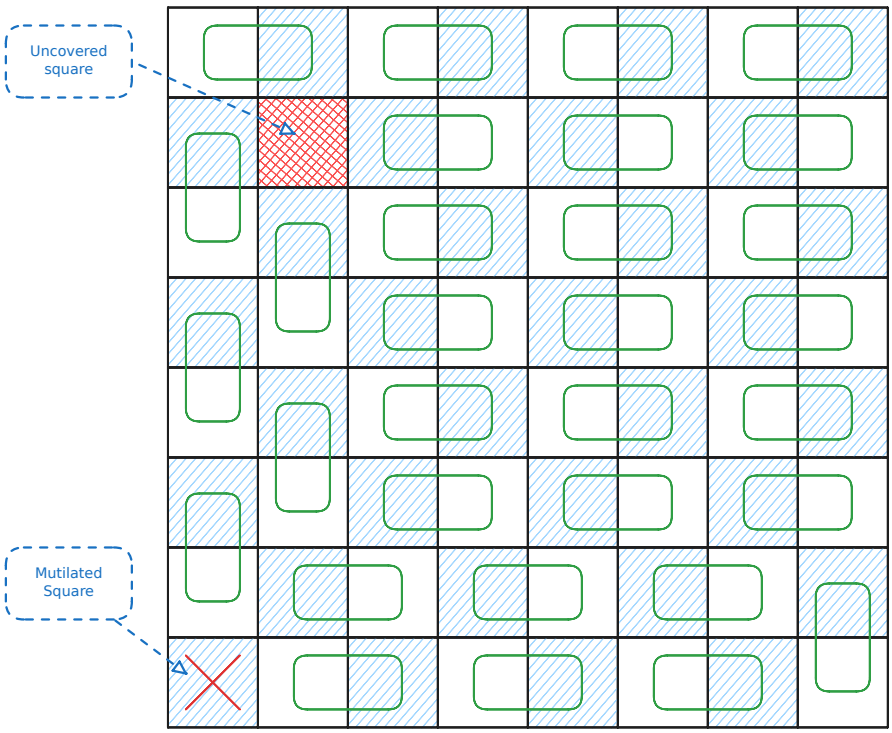

Figure 2.8: Example on an 8×8 chessboard with a missing square. As we can observe, there will always be one square we cannot tile

Definition 2.17: TARSKI **problem definition [16]**

Given a *monotone* function $f : L \to L$ 2.16 of a lattice $(L, \wedge, \vee)$ we define solutions to the problem as:

1. Find $x \in L : f(x) = x$
2. Find $x, y \in L$ such that $x \preceq y$ and $f(x) \not\preceq f(y)$

We assume that f is represented by boolean circuits.

Counting complexity of PPAD Ikenmeyer et al. [21] demonstrated that several PPAD-COMPLETE problems behave differently under #PPAD-1. More specifically #ENDOFLINE$-1 \subseteq$ #P but $\exists A$: #SOURCEORXCESS$^A - 1 \notin$ #P^A. We estimate the counting difficulty of the aforementioned problems based on their position in the hierarchy as seen in the figure **??**.

2.2 Counting Complexity and Combinatorial Interpretations

Decision problems are problems that ask whether an input problem has a solution. Search problems as we observed earlier ask what is a valid solution given the current input instance like what is a valid colouring of a graph, satisfying assignment to a 3-SAT problem etc. Counting complexity, or counting problems take it a step further and usually ask how many valid solutions does a problem have. Valiant formalised this idea with the help of **#P** complexity class 2.2, in which he demonstrated that even if a problem is polynomially solvable in its decision/search variant, can be much harder. A much simpler example can be found in the book by Barak et al. [1] where they provide a reduction from HAM-CYCLE to #CYCLE.

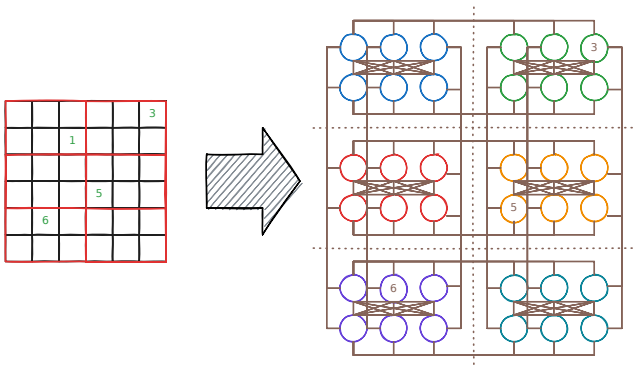

Figure 2.9: Example of reducing a 6x6 sudoku puzzle, into a graph colouring problem

Definition 2.18: #P Complexity Class [30]

A function $f : \mathbb{B}^* \to \mathbb{N} \in \#\mathrm{P}$, if there exists a poly-function $p : \mathbb{N} \to \mathbb{N}$ and a poly-time TM M such that:

$$f(w) = \left|\left\{v \in \mathbb{B}^{p(|w|)} \mid M(w, v) = 1\right\}\right|$$

Alternative but equivalent definition is as follows: A function $f \in \mathbf{\#P}$, if there exists a non-deterministic poly-time TM N such that:

$$f(w) = \mathbf{\#acc}_N(w)$$

Where $\mathbf{\#acc}_N(w)$ denotes the number of accepting paths of N on w.

In hindsight, we can observe that functions in **#P** express functions of the form $\forall w \in \{0,1\}^* : f(w) = |A_w|$, where A_w is a set of objects of size polynomial to the input. Given the above definition, Pak et al. [27, 22] used **#P** to decide whether a class of numbers or objects have or do not have a combinatorial interpretation. Moreover, they argue that the usage of #P has several benefits:

1. By polynomially bounding objects, we avoid cases such as: $f(w) = |\{1, \ldots, f(w)\}|$. In other words, we create a set of combinatorial objects.
2. We can work with $f(\cdot)$, even if its direct computation is hard.
3. #P is a formal system that is highly flexible.

Lastly, we introduce the idea of correlating two counting problems with the help of *parsimonious reductions*. Simply, we say that a problem A can be parsimoniously reduced to a problem B, if for inputs of A, one can create instances of B such that the set of solutions between the two problems have the same cardinality. More formally we refer to the defintion 2.19. An example of how such reduction may look like, can be seen in figure 2.9.

Definition 2.19: Parsimonious reductions

Let R, R' be search problems, and let f be a reduction of $S_R = \{x \mid R(x) \neq \varnothing\}$ to $S_{R'} = \{x \mid R'(x) \neq \varnothing\}$. We say f is **parsimonious** if:

$$\forall x \in S_R : |R(x)| = |R'(f(x))|$$

Where for relation R we say $R(x) \triangleq \{y \in \{0,1\}^* \mid xRy\}$

We introduce a more relaxed variant of parsimonious reductions in 2.2 for one-to-many reductions, such that the two solutions sets are bounded by some polynomial with respect to the size of the input. We observed that for several problems, it can be difficult to obs

Definition 2.20: Poly-Function Bounded Parsimonious Reductions

Given two counting problems $A, B : \mathbb{B}^* \to \mathbb{N}$ and a poly-function $f : \mathbb{N} \to \mathbb{N}$, we say that:

$$A \subseteq^f B \iff \forall x \in \mathbb{B}^n : A(x) \leqslant B(x) \leqslant f(|x|) \cdot A(x)$$

If $\forall x : f(x) = c$ for $c \in \mathbb{N}$, we use the abbreviation:

$$A \subseteq^c B$$

2.3 Kleene Logic and Hazard-Free Circuits

Kleene logic uses the boolean system with an additional *unstable* value [24]. The study of Kleene logic assisted in the development of robust physical systems [17] and aiding in the development of lower bounds for monotone circuits [15, 19, 20, 4]. The current section offers a brief summary to relevant notions and concepts in Kleene logic.

Definition 2.21: Kleene value ordering [26]

The instability ordering for $\leqslant^u$ is defined as: $\bot \leqslant^u 0, 1$. The n-dimensional extension of the ordering is:

$$\forall x, y \in \mathbb{T}^n : x \leqslant^u y \implies \forall j \in [n] : (x_j \in \mathbb{B} \implies x_i = y_i)$$

Definition 2.22: Kleene Resolutions [26, 19]

Given $x \in \mathbb{T}^n$, we define the *resolutions* of x, using the following notation:

$$\textsc{Res}(x) \triangleq \left\{ y \in \mathbb{B}^n \mid x \leqslant^u y \right\}$$

Unless otherwise specified, we assume that all Kleene functions are *natural* 2.23, since they can be represented by circuits 2.1. Detecting hazards is a concept that is analysed heavily when talking about Kleene logic. The idea is ensuring robustness in our circuits, meaning if all resolutions of $x \in \mathbb{T}^n$ give the same value, then we should expect the circuit to behave the same way 2.24. An example of hazard values can be seen in the figure **??**.

Definition 2.23: Natural functions [19]

A function $F : \mathbb{T}^n \to \mathbb{T}^m$ for some $n, m \in \mathbb{N}$ is *natural* if and only if it satisfies the following properties:

1. *Preserves stable values*: $\forall x \in \mathbb{B}^n : F(x) \in \mathbb{B}^m$
2. *Preserves monotonicity* 2.16 2.22

Proposition 2.1: Natural functions and Circuits [26, 19]

A function $F : \mathbb{T}^n \to \mathbb{T}^m$ can be computed by a circuit *iff* F is *natural* 2.23

Definition 2.24: Hazard [19, 15]

A *circuit* C, on n inputs has **hazard** at $x \in \mathbb{T}^n \iff C(x) = \bot$ and $\exists b \in \mathbb{B}, \forall r \in \textsc{Res}(x) : C(r) = b$. If such value does not exists then we say that C is hazard-free.

Definition 2.25: K-bit Hazard [19]

For $k \in \mathbb{N}$ a circuit $C : \mathbb{T}^n \to \mathbb{T}$ has a *k-bit hazard* at $x \in \mathbb{T}^n \iff C$ has a hazard at x and $\perp$ appears at most k times.

We also wish to introduce the following corollary by Ikenmeyer et al. [19] which allows us to construct k-bit hazard free circuits

Corollary 2.1: K-bit Hazard Construction [19]

Given circuit $C : \mathbb{T}^n \to \mathbb{T}$, one can construct a k-bit hazard free circuit of C denoted as C' such that:

$$\begin{aligned} \text{size}(C') &= (\frac{ne}{k})^k(\text{size}(C) + a6) + O(n^{2.71k}) \\ \text{depth}(C') &= \text{depth}(C) + 8k + O(k \log n) \end{aligned}$$

Where size(P) denotes the number of gates of P and depth(P) denotes the longest path from the input bit to the output bit

2.3.1 Kleene logic and Complexity Theory

Lastly we want to introduce the notion of *Alternating Turing Machines*, which are meant to represent a generalisation of non-deterministic Turing machines. These have been introduced by Chandra et al. and the idea is that we introduce universal and existential quantifiers to alternate the course of computation. The informal presentation can be thought as follows: In a non-deterministic TM, given a single state q or configuration c, we execute $\beta_1, ..., \beta_k$ separate processes. We say that q will accept if at least one of the processes $\{\beta_i\}_{i\in[k]}$ accepts. Alternating Turing machines introduce additional conditions where one accepts all $\{\beta_i\}_{i\in[k]}$ to accept or in case of a negation state, if β_1 accepts, then q rejects or vice versa. More formally, we offer the definition in 2.3.1.

Definition 2.26: Alteranting Turing Machines(ATM) [25, 6]

An alternating TM is a seven tuple

$$M = (k, Q, \Sigma, \Gamma, \delta, q_0, g)$$

Where each term denotes:

1. K: The number of work tapes
2. Q: Finite set of states
3. Σ: *Finite* input alphabet. Moreover, $\star \notin \Sigma$ denotes the end of the input
4. Γ: *Finite* work tape alphabet. Moreover $\# \in \Gamma$ to denote blank symbol.
5. δ transition relation which is defined as:

$$\delta \subseteq \Big(Q \times \Gamma^k \times (\Sigma \cup \{\star\})\Big) \times \Big(Q \times (\Gamma \setminus \{\#\})^k \times \{L, R\}^{k+1}\Big)$$

   Important to observe that for a single state, input symbol and work tape symbols, we can take multiple types of transitions.
6. $q_0 \in Q$: Is the initial state
7. $g : Q \to \{\vee, \wedge, \neg, \mathbf{1}, \mathbf{0}\}$: Where g denotes the type of state and $\mathbf{1}, \mathbf{0}$ denote whether it is accepting or not.

The machine essentially contains a read-only input tape with endmarkers and k work tapes which are initially blank. A *step* of m consists of reading a symbol from the input tape, writing a symbol to each work tape, moving the writing head in each head left or right while transitioning into a new state.

Definition 2.27: ATM configurations [25, 6]

A configuration $\mathcal{C}$ of an ATM M is a element of the set $\mathcal{C}_M$

$$\mathcal{C}_M \triangleq Q \times \Sigma^* \times (\Gamma \setminus \{\#\})^k \times \mathbb{N}^{k+1}$$

Which represent, the current state, the input, the current contents of each work tape, and the $k+1$ head positions.

We say that $\mathcal{C}$ that *configuration* is universal (existential, negating, accepting or rejecting) if q is also universal (existential, negating, accepting or rejecting). The initial configuration of M on input x is

$$\sigma_M(x) = (q_0, x, \lambda^k, 0^{k+1})$$

Where λ is a null string.

Definition 2.28: Successor of configurations [6]

Given M, we write $\mathcal{C} \vdash \mathcal{C}'$ to say that $\mathcal{C}'$ is the successor of $\mathcal{C}$, if we can go from one configuration to another with a single δ rule. We require for δ that, for accepting or rejecting configurations to have no successors, for universal or existential to have at least one, and for negating configurations to have only one. We can define the **computation** of a path on x as a sequence $a_0 \vdash a_1 \vdash ... \vdash a_n$ where $a_0 = \sigma_M(x)$

Denoting Accepting or Rejecting computations As mentioned previously, we start from the original configuration. If a process is in an existential configuration $\mathcal{C}$ and $\beta_1, ..., \beta_m$ are all possible successors, then we spawn a process for each configuration and run each one concurrently. If at least one process is accepting we report back to the parent and terminate the rest. Same line of reasoning can be done with the universal quantifiers. Ideally we would like a labelling function f which is defined as

$$f(\mathcal{C}) = \begin{cases} \bigwedge_{\mathcal{C} \vdash \beta} f(\beta) & \text{if } \mathcal{C} \text{ is universal} \\ \bigvee_{\mathcal{C} \vdash \beta} f(\beta) & \text{if } \mathcal{C} \text{ is existential} \\ f(\beta) & \text{where } \mathcal{C} \vdash \beta \text{ if } \mathcal{C} \text{ is negating} \\ \mathbf{1} & \text{if } \mathcal{C} \text{ is accepting} \\ \mathbf{0} & \text{if } \mathcal{C} \text{ is rejecting} \end{cases}$$

But some processes may never halt or require a much greater computation than necessary. We therefore use Kleene logic $\mathbb{T}$ where $\perp$ denotes configurations that we do not know if they are accepting or not. The quantifiers use the same logic as described in the tables 2.1. We now say that a labelling of configurations is a map $\lambda : C_M \to \{0, \perp, 1\}$. We define an operator τ to be an operator of mappings such that:

$$\tau(\ell)(\mathcal{C}) = \begin{cases} \bigwedge_{\mathcal{C} \vdash \beta} \ell(\beta) & \text{if } \mathcal{C} \text{ is universal} \\ \bigvee_{\mathcal{C} \vdash \beta} \ell(\beta) & \text{if } \mathcal{C} \text{ is existential} \\ \ell(\beta) & \text{where } \mathcal{C} \vdash \beta \text{ if } \mathcal{C} \text{ is negating} \\ \mathbf{1} & \text{if } \mathcal{C} \text{ is accepting} \\ \mathbf{0} & \text{if } \mathcal{C} \text{ is rejecting} \end{cases}$$

We can check that τ is monotone with respect to $\preceq$ since if $l \preceq l'$, then $\tau(l) \preceq \tau(l')$. By 2.1.2 there must exist maximum labelling such that

$$\ell^* = \sup_{m \in \mathbb{N}} \tau^m(\Omega)$$

where the supremum is with respect to $\preceq$, where

$$\tau^0(\ell) = \ell$$
$$\tau^{m+1}(\ell) = \tau(\tau^m(\ell))$$

where Ω is denoted as the minimal labelling function $\lambda x.\perp$. In a way that the above procedure describes is the "propagation" process from the leaf nodes up to the root of the tree where the leaf nodes are assigned a label of $\perp$ or accepting. An example can be seen in the figure Lastly given the above we can finally give a definition of halting states

Definition 2.29: Halting states[6]

We say that M halts on x iff $\lambda^*(\sigma_M(x)) \in \{0, 1\}$ and it accepts or rejects respectively iff $\lambda^*(\sigma_M(x)) = 1$ or $\lambda^*(\sigma_M(x)) = 0$

We aim for this ATM to work for only recursively enumerable sets and therefore we aim to limit to the depth of the recursion. Therefore, we define operator τ_C where $C \subseteq C_M$ such that

$$\tau_C(\ell)(a) \triangleq \begin{cases} \tau(\ell)(a) & \text{if } a \in C \\ \perp & \text{otherwise} \end{cases}$$

Where we can say that C is the set of configuration that are reachable within some $f(|x|)$ steps or require at most $f(|x|)$. An example of a computation tree can be observed in the figure 2.10.

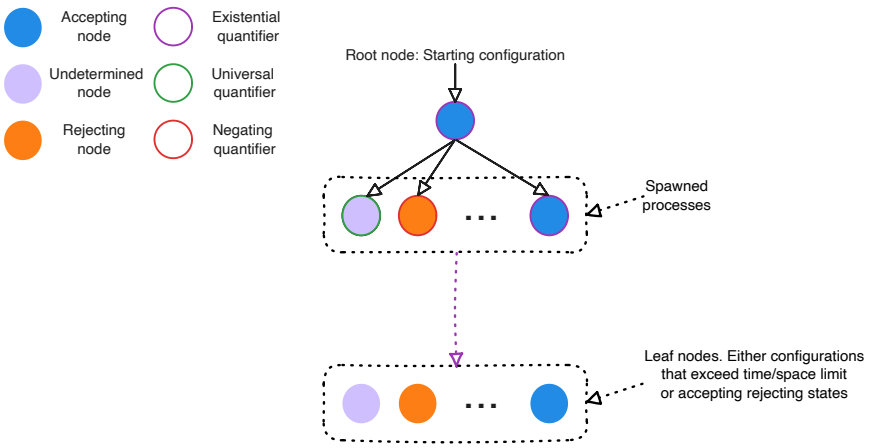

Figure 2.10: Computation tree of an ATM with labellings

We can observe that one can encode Kleene logic into the computation of Turing machines. We use this framework as one of the main motivations of the project.

3 Literature Review

Our work is primarily focused around the paper by Ikenmeyer et al. [21] and the parsimonious reductions discovered across various **PPAD** problems. We introduce the notation $\#\mathbf{PPAD}(L)$, where this indicates the class of problems which are parsimoniously reducible to L. In fact, we extend this notation with $f(\#\mathbf{PPAD}(L))$ to indicate reductions of the form, given L', we say that $L' \in f(\#\mathbf{PPAD}(L))$ if

$$\forall x \in S_{L'} : |R_{L'}(x)| = f(|R_L(g(x))|)$$

We demonstrate an example of the above notation using a finding from the paper by Ikenmeyer et al. [21]: $\mathbf{PPAD}(\mathit{EndOfLine} - 1)/2 = \#\mathbf{P}$. To show inclusion it is easy to see that one can accept all the non-zero sources, since each source is paired with a sink, and we are able to ignore the sink paired with the 0^n source. To show the other direction, they demonstrate an example where they reduce $\mathit{CircuitSAT} \leqslant_C (\#\mathit{EndOfLine} - 1)/2$ by creating a graph, where for each input x, we create a source and sink pair $(0x, 1x)$.

Ikenmeyer et al. [21] developed tools to demonstrate how can one show whether a problem is in $\#\mathbf{P}$ or not, or more specifically, how do demonstrated whether certain functions exhibit combinatorial interpretations. In their paper, they established that for some **PPAD** problems, $\#\mathbf{PPAD}(\cdot) - 1 \subseteq \#P$ but for some other this statement did not hold true under relativising reductions, such as the *SourceOrExcess*. The aforementioned statement oracle separation is known as an oracle separation. The details of how it works are beyond the scope of the current project and therefore we focus on the key takeaway that $\#\mathbf{PPAD}(\mathit{SourceOrExcess}(2, 1)) - 1$ is a harder problem than $\#\mathbf{PPAD}(\mathit{EndOfLine}) - 1$.

But one can ask, how does $\#\mathbf{PPAD}(\mathit{SourceOrExcess}(k, 1)) - 1$ behave for some $k \in \mathbb{N}_{\geqslant 3}$ and in general we will demonstrate that there are many other problems that have interesting counting properties. This leads us directly to the question of, can we somehow quantify each problem in **PPAD**, or more specifically, is there a problem that can "everything" else?

3.1 Project Question and Objectives

As mentioned in the previous section, we observed how different **PPAD** problems can behave differently. The main motivation comes from the observation of how *Kleene logic* can be used to simulate the computation paths of NTMs. We can parallelise this to the boolean problem of *SAT* which given the *Cook-Levin* theorem [9] one can convert every **NP** problem into a *SAT*(problem of satisfying assingments for boolean expression) problem by creating a boolean expression for each snapshot of the Turing Machine such that it returns an accepting state. Another property of the *Cook-Levin* is that every problem can be parsimoniously reduced to a *SAT* assignment, meaning:

$$\forall L \in \mathbf{NP} : \#\mathbf{NP}(L) \subseteq \#\mathbf{NP}(SAT)$$

A reasonable conjecture could also be if one can exploit Kleene-circuit nature of *PureCircuit* to create a *Cook-Levin* type equivalent reduction for **PPAD** problems, or more formally:

$$\forall L \in \mathbf{PPAD} : \#\mathbf{PPAD}(L) \subseteq \#\mathbf{PPAD}(PureCircuit)$$

If the above statement were indeed true, then it would give us insights as to how counting problems behave in *PPAD* as in the section .., we highlighted that even if two problems are *PPAD*-complete, it does not necessarily imply they are counting equivalent.

As established previously and as of the time of writing this paper, the *PureCircuit* was mainly established for inapproximability bounds of **PPAD** problems. Therefore, to tackle the aforementioned conjecture we established the following set of milestones.

1. Establish a chain of parsimonious reductions from the *EndOfLine* to the *PureCircuit* problem.
2. Establish a chain of parsimonious reductions from the $SourceOrExcess(2, 1)$ to the *PureCircuit* problem. Ideally, we would like to extend this to the entire hierarchy of $SourceOrExcess(k, 1)$ problems for some $k \in \mathbb{N}$ or ideally. We will introduce these problems more formally in the upcoming sections *PolySourceOrExcess*.
3. Establish a *Cook-Levin* type theorem for the all **PPAD** problems or more formally

$$\forall L \in \mathbf{PPAD} : \#\mathbf{PPAD}(L) \subseteq \#\mathbf{PPAD}(PureCircuit)$$

To assist with the theory development, we set out to develop a visualisation tool that will be able to visualise instances of the problem as well as find solutions or enumerate through solutions for small instances. We will a give a greater in-depth overview of the tool in the section .., its requirements and its current state.

4 Current findings

4.1 PureCircuit and EndOfLine

We will present several attempts and efforts we made across several problems and their parsimonious reductions. First we want to introduce combine Kleene Logic with **PPAD**. We define the following series of Problems

Definition 4.1: Kleene Unary Numbers

Given a number $p \in M$, its unary representation can be depicted as a binary number $\{0,1\}^M$ such that: $k \in M \to 1^k \in \bar{M}$. Under Kleene algebra, we mainly refer to notations M_k^n where $k, n, M \in \mathbb{N}_0$ such that:

$$M_n^k \triangleq \{1^j \perp^k \mid \forall j \in \mathbb{N}_0 : j + k \leqslant n\}$$

Definition 4.2: HF-StrongSperner **problem**

Input: A hazard-free circuit $\lambda : [M]^n \to \{-1,+1\}^n$, where $M \in n^{O(1)}$, that describes a *StrongSperner* labelling, as explained in 2.13, where the inputs are represented using Kleene Unary numbers 4.1 and $\#1(p) = \bar{p}$.
Output: A point $p \in \times_{i\in[n]} M_n^1 \cup M_n^0$ if and only if: $\lambda(p) = \perp^n$.

We will mainly refer to a specific subset of these problems, which we will refer to as HF-DiscreteStrongSperner. The general idea is that we exploit the hazard-free property of the Strong Sperner function In the original problem, we request for $d+1$ points as a solution to the problem. Due to the assumption that $\forall p, p' \in S : \|p - p'\|_\infty \leqslant 1$, we can assume they are all part of some high-dimensional cube. We abuse this notation by making the observation that given $p \in [M^1]^n$, all the resolutions of p, resolve in a hypercube, as seen in the figure one can observe that, we only need two points to create a panchromatic solution, where $\lambda(p) = \overline{\lambda(p')}$. These can create a lot of duplicate solutions and therefore can behave unpredictably. We will mainly focus on subproblems of the HF-StrongSperner where two cubes do not overlap. More formally we define

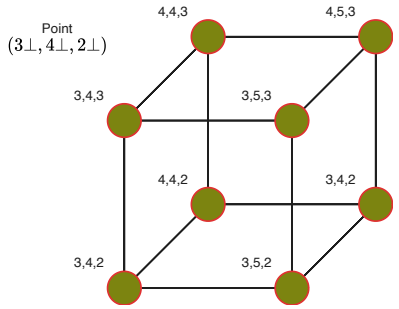

Figure 4.1

Definition 4.3: HF-DiscreteStrongSperner **problem**

An HF-StrongSperner instance λ such that $\forall p \in [M^1]^n$, if $\lambda(p) = \perp^n$, then

$$\forall j, i \in [n], [\phi_i^b(x)]_j \triangleq \begin{cases} x_i & \text{if } i \neq j \\ b & \text{if } i = j \end{cases}$$

$$\forall i \in [n], b \in \{0,1\}, \exists j \in [n] : \forall x, x' \in \mathbf{Res}(\phi_i^b(x)) : [\lambda(x)]_j = [\lambda(x')]_j$$

Lastly we introduce a specific subset of the above solutions where introduce antipodal panchromatic cubes. In the specific type of solution said we argue that two opposite corners of a cube contain opposite colourings. This notion of antipodal cubes is used in the majority of reductions across literature. More formally we say

Definition 4.4: MinRes min and max resolution functions

We use the **MinRes**, to denote the smallest resolution of a kleene string. More formally we can use the following equivalent definitions:

$$\forall x \in \{0,1,\bot\}^*, j \in [|x|] : [\mathbf{MinRes}(x)]_j \triangleq \begin{cases} x_i & \text{if } x_i \in \{0,1\} \\ 0 & \text{otherwise} \end{cases}$$

For the maximum resolution we will use the **MaxRes** function where we instead replace all $\bot$ with 1 instead.

Definition 4.5: HF-AntipodalStrongSperner **problem**

An HF-DiscreteStrongSperner instance λ such that $\forall p \in [M^1]^n$, if $\lambda(p) = \bot^n$, and denote $\mathbf{MinRes}(x) = \hat{x}$ as the *anchor* of the cube, then:

$$\forall b \in \{0,1\}^n, j \in [n] : [\lambda(x+b)]_j = -[\lambda(x + 1^n \oplus b)]_j$$

Claim 4.1: Antipodal discrete squares

Given a panchromatic solution of a HF-AntipodalStrongSperner p, it must be the case then that:

$$\forall j \in \{-1,1\}^n, \exists! x \in \mathbf{Res}(p) : \lambda(x) = j$$

Proof. We proof by induction on n. Lets assume base case $n = 2$. We can observe that the only satisfying cube that is both antipodal and discrete, is some rotation of the following matrix

$$\begin{pmatrix} \mp & + \\ - & \pm \end{pmatrix}$$

Where we denote $+ = (+,+), - = (-,-), \pm = (+,-)$ and $\mp = (-,+)$, we can observe indeed that the panchromatic square contains all the colourings. For the induction step, we can observe that if we want our colouring function to be both discrete and antipodal, then the following must suffice: given k, k' the two resolutions of dimension $k+1$, we know that $[\lambda(p_{-(n+1)}(k))] = (b, \bot^n)$ for some $b \in \{0,1\}^n$. But that implies $[\lambda(p_{-(n+1)}(k'))] = (\bar{b}, \bot^n)$, which indicates a discrete antipodal panchromatic solution. Moreover if $p_{-(n+1)}$ cover 2^n labels, then the whole cube covers all 2^{n+1} labels, which by pigeonhole principle means that we must cover all possible combinations of colourings. □

All the problems above are clearly in **PPAD** as all of them reduce to the *StrongSperner* problem, under only binary inputs, since we make the assumption that all such circuits are inherently *natural*. For the purposes of our current research, we demonstrate a parsimonious reduction from HF-DiscreteStrongSperner to PureCircuit.

Theorem 4.1

$$\#\mathbf{PPAD}(\text{HF-DiscreteStrongSperner}) \subseteq \#\mathbf{PPAD}(\text{PureCircuit})$$

Proof. Given an input instance Λ of a HF-DiscreteStrongSperner, we create a PureCircuit *Input nodes*, *Output nodes*, *Bit generator gadget* and *Circuit application gadget*. To

visualise our construction we will refer to the figure in 4.2. Each of these parts describe the following:

1. **Input nodes**: Vector of n value nodes. Will be represented as $(x_i)_{i\in[n]} \in \mathbb{T}^n$.
2. **Bit generator gadget**: Given a value node x_i, apply the *Bit generator* gadget such that we create Kleene unary number as defined in 4.1, where $x^i \in M_n^{\leqslant 1}$. We will refer to the bit generator circuit as C_i.
3. **Circuit application gadget**: Since the above numbers form coordinates in a $[M]_0^n$ space, we apply the Λ function to extract a set of output nodes $(u^i)_{i\in[n]} \in \mathbb{T}^n$.
4. **Output nodes**: We use the `Copy` gate to copy $u^i \to x_i$, for all $i \in [n]$.

The 1st and last part of our construction is self-explanatory, we will analyse the other two sections in greater depth and prove some claims.

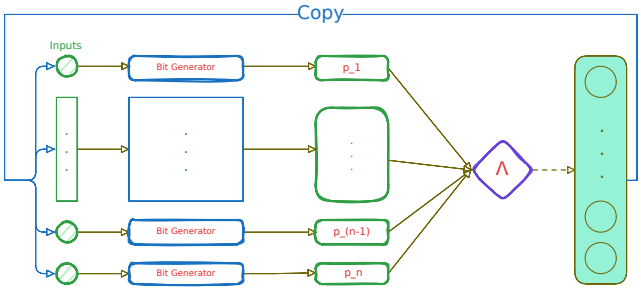

Figure 4.2: Pure Circuit construction visualisation

Bit generator gadget In order to create Kleene unary numbers we make use of a binary tree of `Purify` gates as seen in the figure 4.3. We now claim that the current construction has certain desirable properties.

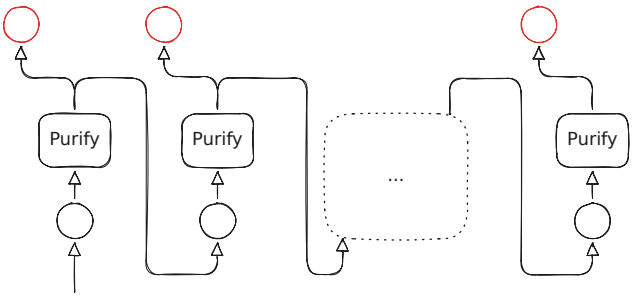

Figure 4.3: Purification gadget. The red nodes denote the outputs of the purification gadget.

Claim 4.2: Purification Claim

Given the description of the bit generator gadget, we make the following two claims:

1. $b \in \{0,1\}^n : C_i(b) = b^n$
2. $C_i(\bot) \in M_n^{\leqslant 1}$

Proof. From the definition of the `Purify` gate we know that $\forall b \in \{0,1\} : \texttt{Purify}(b) = (b, b)$, and therefore any subtree that takes a pure input, will copy its value to all its leafs, which follows

our first claim directly. To the second part of our lemma, we can observe that the only valid outputs are $(0, \perp), (\perp, 1), (0, 1)$. Based on our previous observation, if a purify gate outputs a $(0, \perp)$ value, it will propagate $\perp$ to the rest of the tree and return 0 in the current level, if the value is $(\perp/0, 1)$, the tree will return $\perp/0$ at the specific level, and then propagate 1 to the rest of the leaves. This shows that all possible strings generated are of the form $1^k \perp^{0|1} 0^*$. □

Correctness and parsimonious argument One can observe that the current construction is a valid construction, as we satisfy the arities of all gates and ensured that every node has an in-degree of 1. We now make the following lemma

Lemma 4.1: Correctness Lemma

A correct assignment corresponds to panchromatic cube in the original problem instance.

It suffices to show that every vector $x^i \in M_\perp^1$ and $\forall i \in [n] : \mathbf{x}[u^i] = \perp$. Assume that we have a correct assignment, and WLOG $\mathbf{x}[u^i] = 0$ for some $i \in [n]$. By our copy that, we know that $\mathbf{x}[u^i] = \mathbf{x}[x_i]$. By our purification lemma .. we know that $\mathbf{x}[x^i] = 0^i$. Since our circuit is hazard-free and applies the boundary conditions of the sperner problem, we know that $[\Lambda(x_{-i}(0))]_i = 1 \neq \mathbf{x}[u^i]$. We can make a similar argument for $\mathbf{x}[u^i] = 1$. Therefore, a satisfying assignment corresponds to some panchromatic solution. Given that our instance is in HF-DiscreteStrongSperner, it has to be the case that all $x_i \in M_n^1$, otherwise that would imply that there exists a $(n-1)$-subspace that also covers all the labels. This also shows that there exists a 1-to-1 mapping between a panchromatic cube and a satisfying assignment. □

Although the above reduction also works for the general **HF-StrongSperner**, the reduction is not parsimonious since the PureCircuit will accept cubes in $(n-1)$ or lower subspaces.

4.2 From the EndOfLine

Although we have not managed to create a full reduction from the *EndOfLine* to *PureCircuit*, we will demonstrate the closest to what we can hope. First we acknowledge will mainly focus on the *RLeafD* problem, as it has been shown to be parsimonious to the *EndOfLine* problem. Chen et al. [7], has showed how to convert the problem from a graph problem to a sperner problem but with linear colouring. We will first demonstrate how to do this with bipolar instead. We will demonstrate how to apply the snake embedding technique to go reduce the problem to an *AntipodalStrongSperner* instance.

Bipolar colouring of *RLeafD*

Proof. We construct the same reduction as in the *RLeafD* but with slight differences: We will use points $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V_{2^k}$ and $\mathbf{p}, \mathbf{q}, \mathbf{r} \in B_{2^{2k+5}}$ where

$$B_n \triangleq \left\{\mathbf{p} \in [n]^2\right\}$$

Moreover a polychromatic square will be denoted if all the points in the neighbouring squares satisfy the bipolar colouring. Morevoer below we will denote our colouring scheme for clarity:

$$\begin{aligned} + &= (+1, +1) \\ \pm &= (+1, -1) \\ \mp &= (-1, +1) \\ - &= (-1, -1) \end{aligned}$$

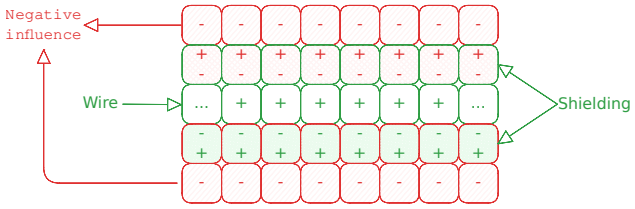

Figure 4.4: Shielding method on 2D-EoL embeddings.

First, we define mapping $\mathcal{F} : V_{2^k} \to B_{2^{2k+5}}$ such that

$$\mathcal{F}(\mathbf{u}) = \begin{pmatrix} 6\mathbf{u}_1 + 6 \\ 6\mathbf{u}_2 + 6 \end{pmatrix} \in B_{2^{2k+5}}$$

Since $G \in C_{2^k}$, its edge-set can be decomposed into a collection of paths and cycles $(P_i)_{i\in[m]}$. By using $\mathcal{F}$, every P_i is mapped to a set $I(P_i) \subset B_{2^{2k+5}}$ We will colour every point on $I(P_i)$ with $+$

$$P = \mathbf{u}^1, \ldots, \mathbf{u}^t, \text{ if P is a cycle } \mathbf{u}^1 = \mathbf{u}^t \tag{4.1}$$

$$I(P) \triangleq \bigcup_{i\in[t-1]} I(u^t u^{t-1}) \tag{4.2}$$

A lot of *Line reductions* under colouring use the notion of shielding. Essentially we embed our lines and cycles in a grid, everything not a line is denoted as a negative charge $-$. All lines are denoted as positive charge $+$. To protect our lines from creating solutions, we cover them with $\pm, \mp$ lines. A visualisation can be seen in figure 4.4. We define the following set to denote the set of shielding points around our paths.

$$O(P) = \{\mathbf{p} \in B_{2^{2k+5}} \backslash I(P) :|: \exists \mathbf{p}' \in I(P), \|p - p'\|_\infty = 1\}$$

If P is a simple path we can decompose $\{\mathbf{s}_p, \mathbf{e}_p\} \cup L(P) \cup R(P)$ such that: $\mathbf{s}_p = \mathcal{F}(\mathbf{u}^1) + \Delta + \Delta^T$, where $\Delta = (\mathbf{u}^1 - \mathbf{u}^2)$ $\mathbf{e}_p = \mathcal{F}(\mathbf{u}^1) + \bar{\Delta} + \bar{\Delta}^T$, where $\bar{\Delta} = (\mathbf{u}^t - \mathbf{u}^{t-1})$ Starting from $\mathbf{s}_p$, enumerate vertices in $O(P)$ clockwise as $\mathbf{s}_p, \mathbf{q}^1, \ldots, \mathbf{q}^{n_1}, \mathbf{e}_p, \mathbf{r}^1, \ldots, \mathbf{r}^{n_2}$ such that

$$L(P) = \{\mathbf{q}^i\}_{i\in[n_1]} \ ,$$
$$R(P) = \{\mathbf{r}^i\}_{i\in[n_2]}$$

If P is a simple cycle, then can can decompose $L(P) \cup R(P)$, where $L(P)$ contains all the vertices on the left side of the cycle and $R(P)$ all the vertices on the right side If G is specified by $(K, 0^k) \subset C_{2^k}$, we can uniquely decompose the edge set into $P_1, \ldots P_m$ Every path is either a maximal path, or a cycle in G We can prove the following:

Lemma 4.2: Disjoint Lemma

$$\forall i, j : 1 \leqslant i \neq j \leqslant m : \Big(I(P_i) \cup O(P_i)\Big) \cap \Big(I(P_j) \cap O(P_j)\Big) = \varnothing$$

Proof. □

We can use the following to describe our colouring algorithm:

Algorithm 4.1 Turing machine F with input $(p_1, p_2) \in B_{2^{2k+5}}$

```
if p_1 = 0 then
    if p_2 ⩽ 6 then
        return +
    else
        return ±
else if p_1 < 6 then
    if p_2 = 6 then
        return +
    else if p_2 < 6 then
        return ∓
    else if p_2 = 7 then
        return ±
    else
        return −
else
    return M_K(p)
```

Where $M_k(\mathbf{p})$ is can be described with the following description:

$$M_k(\mathbf{p}) = \begin{cases} + & \text{if } \exists i, \mathbf{p} \in I(P_i) \\ \pm & \text{if } \exists i, \mathbf{p} \in L(P_i) \vee p_1 = 0 \\ \mp & \text{if } \exists i, \mathbf{p} \in R(P_i) \vee p_2 = 0 \\ - & \text{otherwise} \end{cases}$$

We will call the path of nodes $(0,0) \to (0,6) \to (6,6)$ as the tail. An example of how such reduction looks like can be seen in figure 4.5.

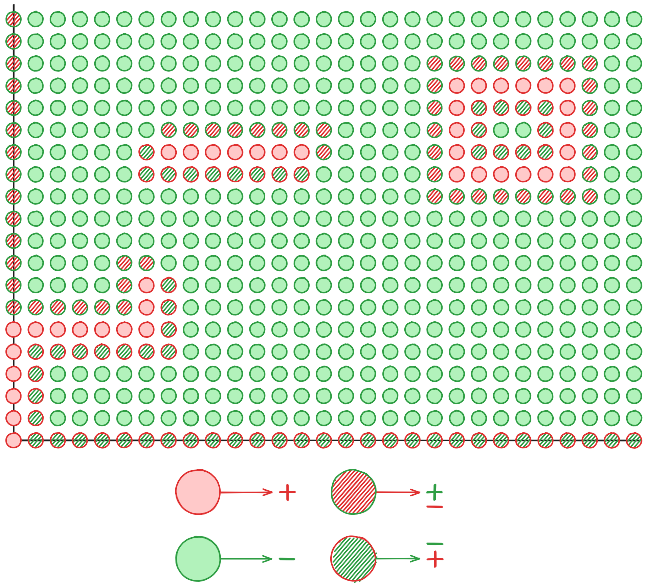

Figure 4.5: 2D EndOfLine Bipolar Colouring

Lemma 4.3: Correctness Lemma

The following statements hold correct:

1. $\forall i, j : \min_{x \in P_i, y \in P_j} \|\mathcal{F}(x) - \mathcal{F}(y)\|_\infty > 4$
2. All polychromatic squares represent solutions

To prove the first statement, assume we have $a, b \in V_{2^k} : \|a - b\|_\infty > 1$. WLOG assume $a - b = (x, y)$ for some $x, y \in \mathbb{N}$ which implies the following:

$$\mathcal{F}(a) = \mathcal{F}\begin{pmatrix} b_x + x \\ b_y + y \end{pmatrix} = \begin{pmatrix} 6b_x + 6x + 6 \\ 6b_x + 6y + 6 \end{pmatrix}$$

$$\mathcal{F}(a) - \mathcal{F}(b) = \begin{pmatrix} 6x \\ 6y \end{pmatrix}$$

Since $x + y > 0 \implies \|\mathcal{F}(a) - \mathcal{F}(b)\|_\infty \geqslant 6 > 4$. This statement is meant space all lines far enough such that the colouring of each one won't affect the other.

To show our second part of our solution, we can observe that: All nodes that are not boundary nor not in $\bigcup_i I(P_i) \cup L(P_i) \cup S(P_i)$ are coloured $-$. All $+$ coloured nodes are on the tail and in the line, all non-tail nodes are cushioned by $\pm, \mp$ and every non-tail node is not poly chromatic. All $-$ elements do not affect $\pm, \mp$ coloured nodes. Since all lines are sufficiently apart from each other, it suffices to check only the ends of paths. One can observe that the square anchored at s_{P_i}, and the square that contains e_{P_i} will be polychromatic. Since only one of each $1 - 1$, therefore we have a valid and parsimonious reduction. □

We call this coloured instances as *2D-StrongRLeafD*. We now want to introduce the Snake embedding method developed by Chen et al. [8]. The idea is folding the space in higher and higher dimensions with the goal of going from an $2^n \times 2^n \to f(n)^m$ where $m, f(n) \in O(n)$. The original reduction, expresses the ability to go from a $2^n \times 2^n$ space, to a $f(n)^d$ where $d = \lceil \frac{n}{f(n)} \rceil$ space. We must also keep in mind the corollary 10 by [19], where they demonstated that the size of a circuit scales exponentially with the number of potential hazard. This implies the best we can hope for when using the vanilla method is:

$$M = n^k, f(n) = \log_2(n)^k \implies d = \frac{n}{log(n)^k}$$

The above demonstrates that the resulting circuits are superpolynomial with respect to the input size.

In the current duration of the project, despite not being able to achieve a generalised reduction, we want to demonstrate a specific circuit that can be constructed. This circuit contains properties that may lead to a full reduction in the future. We demonstrate this over a simplified version of the snake embedding, first employed by .. on the *StrongTucker* version of the problem, which reduced the space from $2^n \times 2^n \to 7^m$, where $m = O(n)$. We will show how to expend this for the *StrongSperner* problems as well as demonstrate that such reduction will to *AntipodalStrongSperner* instansces.

Theorem 4.2

#PPAD(2D-StrongEoL) $\subseteq$ **#PPAD**(AntipodalStrongSperner)

Proof. General idea is that we have these twisted sheets that capture the n-1 embedding onto the higher on We are given as an input a $2^n \times 2^n$ grid representing any *2D-StrongEoL* problem.

We will reduce to a hypercube $[7]^m$ where $m = O(n)$. The procedure is summarised as follows: Given dimension n, apply the **padding operation** such that $n' = 3 \cdot s + 1$ for some $s \in \mathbb{N}$. Then apply the *folding* operation, which reduces "width" of the space to $s + 3$ and compresses the space into a higher dimension. We will now go into a greater detail as to how each operation works. To make notation simple and consistent, we define M^t as

$$M^t \triangleq \bigtimes_{i \in [n_t]} m_i^t$$

Where m_i^t denotes the width of dimension i at iteration t, and n_t as the number of dimensions.

Padding Dimensions We will use the definition 4.2 to define our operation.

Definition 4.6: Padding dimension $P(T, i, u)$

Given labelling function $\Lambda^t : M_d^T \to \{-1, +1\}^{n_t}$, we define the padding operation $P(\Lambda^t, i, u)$ for $i \in [n_t]$ and $u \in \mathbb{N}$ where $n_{t+1} = n_t$ and

$$\forall j \in [n_{t+1}] : m_j = \begin{cases} m_i & \text{if } i \neq j \\ m_i + u & \text{otherwise} \end{cases}$$

We essentialy add u additional layers to dimension i.

How the operation work is that we append a null dimension such that:

$$\forall x \in M^t, j \in [n] : [\Lambda(x_{-i}(m_i^t + 1))]_j = \begin{cases} -1 & \text{if } x_i > 0 \\ +1 & \text{otherwise} \end{cases}$$

We can be sure that the above will not result in any new solutions since in $[\Lambda^t(\cdot, m_i^t, \cdot)]_i = -1$, and therefore any solution between $(\cdot, m_i^t, \cdot), (\cdot, m_i^t + 1, \cdot)$ will not yield in a solution.

Snake Embedding below we will give a overview of the snake embedding technique. This method is a modification of the method used in the tucker problem, in order to make it compatible with *StrongSperner*.

Before giving a formal definition, we will first give an informal overview of what the technique actually does based on the figure 4.6. where the y-axis denotes the new dimension, denoted as d' and the x axis denotes the folded dimension referred as $\bar{i}$ with width $m_{\bar{i}}^{t+1} = s + 3 = m_{\bar{i}}^{t'}$, given that the original width was $3s + 1$ for some $s \in \mathbb{N}$. Moreover, we will use notation (a, b) as a, the labels in the d subspace and b as the label of the $d + 1$ dimension. From the figure we observe that we have orange, red, green and blue lines. Blue and orange lines denote $(\omega, -)$ and $(\omega, +)$ where ω is the d-dim null space, where for each point > 0, we use label $-$ and for 0 we use label $+$. For the red and green lines, these are denoted as $(\star, -), (\star, +)$ where $\star$ is an n-dim copy of the lower space with some additional points. The core idea is that all solutions, must belong in between the red and green lines, and if we have a solution in the n-space, we should also have a solution in the $n + 1$ space, as shown in the figure 4.7.

To define the above more formally, we define sets $B^U, B^L \subseteq M_{n+1}^{t'}$ which denote the set of boundary indices of the orange and blue lines. In addition, we define functions $\ell^P, \ell^N \subseteq M_{n+1}^{t'}$ to denote the points on the greed and red lines with functions ϕ^P, ϕ^N that are maps $phi^i : \ell^i to M_n^t$ that indicate the indexes in the original space. Given our new circuit Λ^{t+1}, we define the

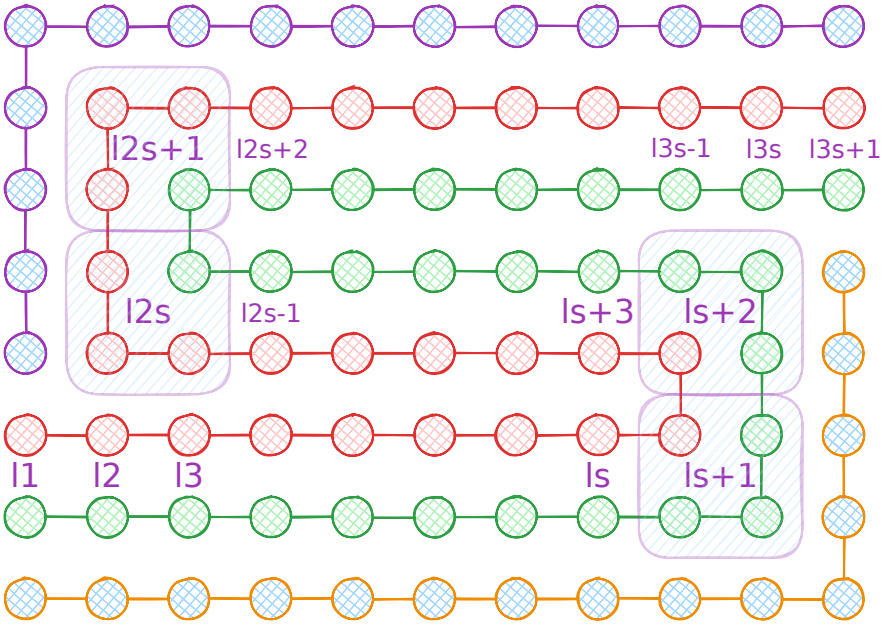

Figure 4.6: Snake Embedding technique between the folding dimension and the new dimension

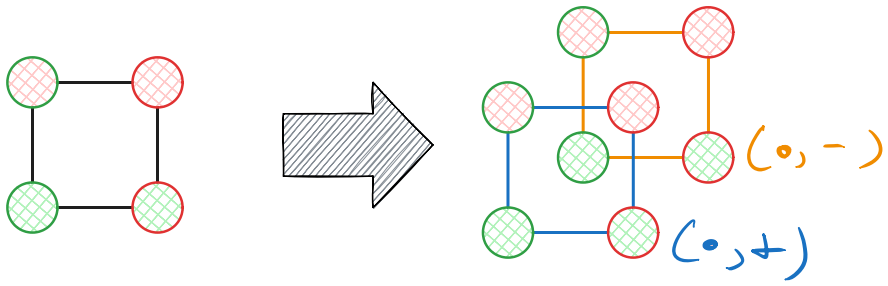

Figure 4.7: Snake Embedding technique between the folding dimension and the new dimension

following:

$$\forall x \in M_{n+1}^{t+1}, j \in [n] : [\Lambda^{t+1}(x)]_j = \begin{cases} -1 & \text{if } x \in B^U \cup B^O \wedge x_j > 0 \\ +1 & \text{if } x \in B^U \cup B^O \wedge x_j = 0 \\ [\Lambda^{t+1}(\phi^P(x))]_j & \text{if } x \in \ell^P \\ [\Lambda^{t+1}(\phi^N(x))]_j & \text{if } x \in \ell^N \end{cases}$$

For the new dimension, we apply the labelling rules as described previously

$$\forall x \in M_{n+1}^{t+1} : [\Lambda^{t+1}(x)]_{n+1} = \begin{cases} -1 & \text{if } x \in B^U \cup \ell^N \\ +1 & \text{if } x \in B^L \cup \ell^P \end{cases}$$

Using the above we formally define the operation the snake embedding operation $S(T, i)$.

Definition 4.7: Snake Embedding operation $S(\Lambda^t, i)$

Given labelling function $\Lambda : M_n^t \to \{-1, +1\}^n$, apply the snake embedding operation on dimension $i \in [n]$, where $m_i \equiv 1 \mod 3$, to a get a new labelling function $\Lambda' : M'_{n+1} \to \{-1, +1\}^{n+1}$ such that $m'_i = s + 3$, where Λ' is defined based on the above description.

Claim 4.3: Antipodality Solution preservation

Given Λ^t which is antipodal, we argue that Λ^{t+1} will remain antipodal.

Proof. We will prove this by induction. We start with a base case of $2 \to 3$ and we can observe that all *2D-StrongEoL* instances preserve the antipodality condition. Assume we apply snake embedding to one of the two dimensions such that we have $2^n \times y \times z$. We can observe that all solutions in the embedded space will be anchored between the red or green lines (using figure 4.6), otherwise the third dimension will not be covered. Moreover one can observe can never be anchored in any of the turning points since we have a copy of the point and therefore x, y dims will not be covered. We define $\perp^n(\cdot)$ as

$$\forall \bar{p} \in [M] : \perp^n(p) \triangleq \begin{cases} 1^p \perp & \text{if } \bar{p} < M \\ 1^M & \text{otherwise} < M \end{cases}$$

G

□

□

References

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge, 2009.

[2] M. Black. *Critical Thinking: An Introduction to Logic and Scientific Method*. Prentice-Hall Philosophy Series. Prentice-Hall, Incorporated, 1946.

[3] K. Bronisław. Un theoreme sur les functions d'ensembles. 6:133–134, 1928.

[4] J. Bund, C. Lenzen, and M. Medina. Small Hazard-Free Transducers. *IEEE Transactions on Computers*, 74(5):1549–1564, May 2025.

[5] S. H. Chan and I. Pak. Computational complexity of counting coincidences. *Theoretical Computer Science*, 1015:114776, Nov. 2024.

[6] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM (JACM)*, 28(1):114–133, 1981.

[7] X. Chen and X. Deng. On the complexity of 2D discrete fixed point problem. *Theoretical Computer Science*, 410(44):4448–4456, Oct. 2009.

[8] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player Nash equilibria. *J. ACM*, 56(3):14:1–14:57, May 2009.

[9] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, May 1971. Association for Computing Machinery.

[10] C. Daskalakis, P. Goldberg, and C. Papadimitriou. *The Complexity of Computing a Nash Equilibrium*, volume 39. Association for Computing Machinery, Jan. 2006.

[11] C. Daskalakis, S. Skoulakis, and M. Zampetakis. The Complexity of Constrained Min-Max Optimization, Sept. 2020.

[12] C. Daskalakis, S. Skoulakis, and M. Zampetakis. The complexity of constrained min-max optimization. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1466–1478, New York, NY, USA, June 2021. Association for Computing Machinery.

[13] A. Deligkas, J. Fearnley, A. Hollender, and T. Melissourgos. Constant inapproximability for PPA. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1010–1023, New York, NY, USA, June 2022. Association for Computing Machinery.

[14] A. Deligkas, J. Fearnley, A. Hollender, and T. Melissourgos. Pure-Circuit: Tight Inapproximability for PPAD. *J. ACM*, 71(5):31:1–31:48, Oct. 2024.

[15] E. B. Eichelberger. Hazard Detection in Combinational and Sequential Switching Circuits. *IBM Journal of Research and Development*, 9(2):90–99, Mar. 1965.

[16] J. Fearnley, D. Pálvölgyi, and R. Savani. A Faster Algorithm for Finding Tarski Fixed Points. *ACM Trans. Algorithms*, 18(3):23:1–23:23, Oct. 2022.

[17] S. Friedrichs, M. Függer, and C. Lenzen. Metastability-Containing Circuits. *IEEE Transactions on Computers*, 67(8):1167–1183, Aug. 2018.

[18] A. Hollender and P. Goldberg. The Complexity of Multi-source Variants of the End-of-Line Problem, and the Concise Mutilated Chessboard, June 2018.

[19] C. Ikenmeyer, B. Komarath, C. Lenzen, V. Lysikov, A. Mokhov, and K. Sreenivasaiah. On the complexity of hazard-free circuits. *Journal of the ACM*, 66(4):1–20, Aug. 2019.

[20] C. Ikenmeyer, B. Komarath, and N. Saurabh. Karchmer-Wigderson Games for Hazard-free Computation, Nov. 2022.

[21] C. Ikenmeyer and I. Pak. What is in #P and what is not? In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 860–871, Oct. 2022.

[22] C. Ikenmeyer, I. Pak, and G. Panova. Positivity of the Symmetric Group Characters Is as Hard as the Polynomial Time Hierarchy. *International Mathematics Research Notices*, 2024(10):8442–8458, May 2024.

[23] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, Aug. 1988.

[24] S. Kleene and M. Beeson. *Introduction to Metamathematics*. Ishi Press International, 2009.

[25] D. Kozen. *Theory of Computation*. Texts in Computer Science. Springer London, 2006.

[26] M. Mukaidono. On the B-ternary logic function. *Trans. IECE*, 55:355–362, Jan. 1972.

[27] I. Pak. What is a combinatorial interpretation?, Sept. 2022.

[28] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, June 1994.

[29] A. Rubinstein. Inapproximability of Nash Equilibrium. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 409–418, New York, NY, USA, June 2015. Association for Computing Machinery.

[30] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, Jan. 1979.

Appendices

A The Flux Sketch