Counting Problem of *PureCircuit*

Christos Demetriou *Department of Computer Science*
University of Warwick
Coventry, UK
u2018918@live.warwick.ac.uk, u2018918

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

I. Introduction

Over the last couple of years, there has been a revolutionary initiative in the field of combinatronics. Combinatronics has been a field of study in mathematics that primarily focused on the notion of counting objects with certain properties. Over time, this notion has shifted, especially in the subfield of algebraic combinatorics, where there is no clear notion of the object that we are counting, and the numbers express something more abstract [1]. This gave a need to be able to assign a combinatorial interpretation to such numbers, or more simply, do these numbers correspond to some counting over a set of objects. Being able to find such definitions or interpretations can be very important, it allows us to utilise tools from combinatorics as well as allow us to understand and reveal hidden structures and properties for such numbers [1]. Moreover, there are several problems or numbers such as *Kronecker coefficients*, whose combinatorial interpretation, would give a step towards the resolution of the $P \neq NP$ conjecture [2].

To reiterate the previous statement, we can understand combinatorial interpretation as the process of: given a sequence of numbers $\{a_x\}$, find a set of combinatorial objects A_x such that $|A_x| = a_x$ To formalise the current idea, Igor Pak et al. has concluded that $f \in$ *#P*, implies that f has a combinatorial interpretation [1], [2]. We will explore this idea in much greater detail in the upcoming section, but the main benefit is the ability to use a very expressive but formal language that encapsulates this notion of a combinatorial interpretation.

In our current work, we focus on extending the work done by Ikenmeyer et al., where they focused on the creation of frameworks that determine whether $f \in^{?}$ **#P**, by looking at the complexity class of ***#****TFNP -1*. This is a class of problems that are guaranteed to have a solution and their solutions are verifiable in polynomial time. In their paper, they were able to show that for a subclass of problems, also known as *PPAD* $\subseteq$ *TFNP*, different *PPAD-complete* problems, may or may not have a combinatorial interpretation. Our contribution, comes to the analysis of a specific problem, known as PureCircuit, which utilises Kleene logic, to find satisfying assignments in sequential circuits. We hope to demonstrate that such problem could help us bound, the counting complexity limits of #PureCircuit $- 1$.

A. Project objectives

Below we will present our table of objectives. We will denote updated objectives with (*), new objectives with (!), completed objectives with (✓), deleted objectives with (-).

R.1) (✓) Find a parsimonious reduction from the *EndOfLine* to *EndOfLine*.
R.2) (!) Improve the combinatorial bound between the reduction from EndOfLine to PureCircuit
R.3) (!) Demonstrate that #PPAD$(PureCircuit) - 1 \nsubseteq$ **#P**
R.4) (*) Prove or disprove the following $\forall n \in \mathbb{N}_{\geq 2}$:

$$\exists c \in \mathbb{N} : \#\textsc{SourceOrExcess}(n, 1) \subseteq^{c} \#\textsc{PureCircuit}$$

Below we will be representing the development portion of the project.

S.1) Visualise a pure circuit
S.2) Generate a solution
S.3) Count number of solutions for smaller scales

We will compile the rest of the report, based on our current findings, how we modified our objectives to the current ones as well as general reflections.

B. Research Question

Our main objectives lead to the following conjecture:

Conjecture I.1: #PPAD $-$ 1 **hardness**

We conjecture that proposition that every language in #PPAD $-$ 1 can be parsimoniously reduced up to some polynomial factor to #PURECIRCUIT $-$ 1. We express that idea more formally as such:

$$\forall L \in \text{PPAD}, \exists f \in n^{O(1)} : \#L - 1 \subseteq^{f} \#\text{PURECIRCUIT} - 1$$

The conjecture I.1 captures the essence of our work. Being able to finding such relations will allow us to understand in greater depth the complexities behind #TFNP $-$ 1 class.

II. Preliminaries and Background review

A. Important Notation

To avoid ambiguity, we introduce the following notation conventions used throughout the paper. For any $n \in \mathbb{N}$, we write $[n] \triangleq \{1, \ldots, n\}$ and $[n]_0 \triangleq [n] \cup \{0\}$. We define the Boolean domain as $\mathbb{B} \triangleq \{0, 1\}$ and three value domain $\mathbb{T} \triangleq \{0, 1, \perp\}$.

B. Background overview

1) Counting Complexity and Combinatorial Interpretations: Counting complexity looks into complexity of counting solutions using notions such as **#P**. **#P** was created by [3], to demonstrate the difficulty of counting the number of solutions to a problem, even they are polynomially verifiable.

Definition II.1: #P Complexity Clas [3]

#P is a class of functions $f : \{0,1\}^* \to \mathbb{N}$ such that: there exists a polynomial-time deterministic TM M, and $p : \mathbb{N} \to \mathbb{N}$ such that $p \in n^{O(1)}$, we have:

$$f(w) = \left| \left\{ v \in \{0,1\}^{p(|w|)} \mid M(w, v) = 1 \right\} \right|$$

As we can see, **#P** allows us to define a set of objects, whose cardinality equals $f(w)$. The reason for choosing **#P** to define combinatorial interpretations is [4]:

1) By polynomially bounding words, we avoid cases such as: $f(w) = |\{1, \ldots, f(w)\}|$.
2) Current framework allows us to work with $f(\cdot)$, whose direct computation can be computationally hard.

The current framework was used in several papers such as [2] and [4] where they were able to use tools from complexity theory to show that many structures do or do not have a combinatorial interpretation. For the purposes of the current project, we are focusing on [2], and TFNP problems. Lastly we introduce the idea of correlating two counting problems with the help of *parsimonious reductions* II.2.

Definition II.2: Parsimonious reductions

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

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

Below we are introducing a variant of parimonious reductions that allows for one-to-many reductions.

Definition II.3: Poly-Function Bounded Parsimonious Reductions

Given two counting problems $A, B : \{0,1\}* \to \mathbb{N}$ and a function $f : 0,1^* \to \mathbb{N}$ such that $f \in n^{O(1)}$, we say that:

$$A \subseteq^{f} B$$

If for input $w \in \{0,1\}^*$, if a represent the number of solutions for problem A and b the number of solutions for problem B, we have:

$$a \leq b \leq f(|w|) \cdot a$$

If $\forall x : f(x) = c$ for $c \in \mathbb{N}$ then we will just say

$$A \subseteq^{c} B$$

2) Total Search Problems and PPAD: We give a brief overview of **TFNP** and **PPAD**.

Definition II.4: Search Problems and Total Search Problems

Search problems can be defined as relations $R \subseteq \{0,1\}^* \times \{0,1\}^*$, where given $x \in \{0,1\}^*$, we want to find $y \in \{0,1\}^*$ such that xRy.
Total Search problems are search problems such that:

$$\forall x \in \{0,1\}^*, \exists y \in \{0,1\}^* : xRy$$

Definition II.5: FNP **and** TFNP

FNP are *search problems* such that there exists poly-time TM $M : \{0,1\}^* \to \{0,1\}$ and a poly function $p : \mathbb{N} \to \mathbb{N}$ such that:

$$\forall x \in \{0,1\}^*, y \in \{0,1\}^{p(|x|)} : xRy \iff M(x,y) = 1$$

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

Our current work focuses on a specific subclass of **TFNP** problems which is defined as follows:

Definition II.6: *EndOfLine* problem [5]

Given circuits $S, P \in \{0,1\}^n \to \{0,1\}^n$ such that $S, P \in n^{O(1)}$ we define a directed graph $G = (V, E)$, such that $V = \{0,1\}^n$ and E defined as:

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

We define source or sinks $\forall v \in V : deg(v) = (0,1)$ or $deg(v) = (1,0)$, respectively. We also syntactially 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 if and only if $deg(v)$ is either $(0,1)$ or $(1,0)$.

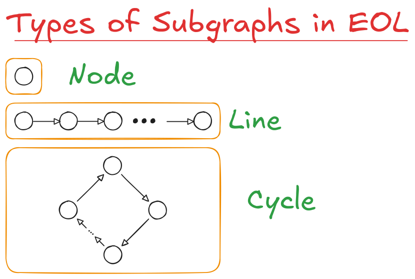


Fig. 1: Types of subgraphs in ENDOFLINE

An illustrative example of an instance can be seen in the figure 1. Using the ENDOFLINE problem, we define the PPAD complexity class II.7

Definition II.7: PPAD complexity class [5]

PPAD is defined as the set of search problems that are reducible to the ENDOFLINE problem II.6.

PPAD has been created by Papadimitriou [5] to demonstrate a subset of problems in **NP** that are guaranteed to have a solution but can be very difficult to find. In the next section we will introduce the relevant of PPAD problems.

a) The PureCircuit problem: The PURECIRCUIT was created by Deligkas et al. [6] to demonstrate the hardness of approximating PPAD problems. it uses kleene-logic based circuits and with continuity of arguments, they demonstrated PPAD-COMPLETENESS.

Definition II.8: PURECIRUIT Problem Definition [6]

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$(v) \leq 1$. A solution to input instance (V, G) is denoted as an assignment $\mathbf{x} : V \to \{0, \perp, 1\}$ such that for all nodes we have:

1) if v is the output of a $(\textit{NOR}, u, v, w)$ gate:

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

2) *Purify*:

$$\forall b \in \{0, 1\} : \mathbf{x}[u] = b \implies \mathbf{x}[v] = b \wedge \mathbf{x}[w] = b$$
$$\mathbf{x}[u] = \perp \implies \{\mathbf{x}[v] \cup \mathbf{x}[w]\} \cap \{0, 1\} \neq \emptyset$$

The definition of PURECIRCUIT that we will be using for the standard set of gates $\{\wedge, \vee, \neg\}$, is based Kleene's three-valued strong logic of indeterminancy which extends the traditional $\mathbb{B}$ logic [7]. Their behaviour can be described in the tables shown in I In addition to that, we will make use of the *Copy* gate which we can define as *Copy*$(x) = \neg(\neg x)$. This allows our circuits to be robust and easier to work with.

`not`	
0	1
$\perp$	$\perp$
1	0

(a) `not` gate

`and`	0	$\perp$	1
0	0	0	0
$\perp$	0	$\perp$	$\perp$
1	0	$\perp$	1

(b) `and` gate

`or`	0	$\perp$	1
0	0	$\perp$	1
$\perp$	$\perp$	$\perp$	1
1	1	1	1

(c) `or` gate

TABLE I: Three-valued logic [7]

For the *Purify* gate, Deligkas et al. showed that the only solutions that are essential for *Purify* are $\{(0, \perp), (\perp, 1), (0, 1)\}$ with the help of continuity arguments [6]. Adding more solutions does not change the complexity as he showed in the original variant of the problem. We acknowledge that the solution set will be different from the original one but, one can observe that #PURECIRCUIT-SIMPLIFIED $\subseteq$ #PURECIRCUIT, and therefore any proposition or argument of the sort: #A $\subseteq$ #PURECIRCUIT-SIMPLIFIED $\implies$ #A $\subseteq$ #PURECIRCUIT. For the purposes of the report, any solution change will be made explicit and we will refer to all such simplified variants as #PURECIRCUIT to avoid confusion.

Definition II.9: SOURCEOREXCESS problem

We define as *SourceOrExcess*$(k, 1)$ for $k \in \mathbb{N}_{\geq 2}$ the search problem as such: Given a poly-sized successor circuit $S : \{0, 1\}^n$ and a set of predecssor poly-sized circuits $\{P_i\}_{i \in [k]}$, we define the graph $G = (V, E)$ such that, $V = \{0, 1\}^n$ 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 sink, meaning deg$(0^n) = (0, 1)$. A valid solution is a vertex v such that *in-deg*$(v) \neq$ *out-deg*(v)

b) Sperner problems: Below we will refer to the notion of Sperner problems which involve the idea of using the topology of a problem and a colouring scheme to ensure that a substructure is panchromatic. There two variants of colouring scheme that are used: one of them will be referred to as the **linear** colouring where for dimension d, assign $d + 1$ distinct colours to each point [8], [9]. Below we will refer to **bipolar** colouring, which has been used grid-like topologies of the Sperner property [6], [10], [11]. It has to be noted that these are not their official names, but we have decided to use this naming scheme for clarity.

Definition II.10: Bipolar colouring

Given dimension d, we refer to the bipolar colouring C of a point $v \in S^d$ where S is some arbitrary set in d dimensionality the following:

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

Essentially a point is a associate with a d dimensionaly binary vector. We say that a set of points $A \subseteq S^d$ **cover all the labels** if:

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

Definition II.11: StrongSperner problem

Input: A boolean circuit that computes a bipolar labelling $\lambda : [M]^N \to \{-1, 1\}^N$ II.10 satisfying the following boundary conditions $\forall i \in [N]$:

- 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:

- *Closessness condition*: $\forall i, j \in [N] : \|x^{(i)} - x^{(j)}\|_\infty \leq 1$
- *Covers all labels* as defined in II.10

The above is a generalised variant of the tradtional Sperner problem to a grid of dimensions N and width of M. Throughout literature the same variants of the problem or specifications have been defined using sperner or discrete brouwer [6], [8]–[10]. For the sake of clarity, we will stick to StrongSperner.

Definition II.12: nD-StrongSperner problem

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

$$\{\alpha + x \mid x \in \{0, 1\}^n\} \text{ cover all the labels II.10}$$

We use A to avoid edge cases. We assume dimensionality $n \geq 2$.

Chen et al. [10] demonstrated that all nD-StrongSperner are PPAD-complete. We mainly use this to show counting arguments with respect to the EndOfLine as people have indicated a parsimonious reduction between 2D-StrongSperner and 3D-StrongSperner to the EndOfLine problem but with *linear* colouring. The authors of these papers use these problems indifferently when talking about the reductions and therefore we can assume that either colouring will create a correct reduction.

c) Other PPAD problems: We will define several problems in PPAD that are related with our project and demonstrate the counting complexity of PureCircuit.

Definition II.13: SourceOrExcess problem

We define as *SourceOrExcess*$(k, 1)$ for $k \in \mathbb{N}_{\geq 2}$ the search problem as such: Given a poly-sized successor circuit $S : \{0, 1\}^n$ and a set of predecssor poly-sized circuits $\{P_i\}_{i\in[k]}$, we define the graph $G = (V, E)$ such that, $V = \{0, 1\}^n$ 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 sink, meaning $\deg(0^n) = (0, 1)$. A valid solution is a vertex v such that *in-deg*$(v) \neq$ *out-deg*(v)

Lastly we will introduce Tarski **??**, which is a problem in PLS $\cap$ PPAD where PLS is a class of problems based on the idea of local search [12] and uses the Knaster-Tarski fixed point theorem II.1. We will use this problem in later sections to connect Kleene algebra with PPAD.

Definition II.14: Monotone functions

Given two posets $(L_1, \preceq_{L_1})$ 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 II.1: Knaster Tarski Fixed point theorem [13], [14]

Given a lattice $(L, \wedge, \vee)$ and a *monotone* $f : L \to L$ II.14

$$\exists c \in L : f(c) = c$$

Using the Tarski theorem, Fearnley et al. [14] created a TFNP variant of the problem which finds fixed points or identifies points that break the monotone argument. We will be using Tarski with Kleene logic to connect the two.

d) Counting complexity of PPAD: The current project looks on the counting complexity of such problems as despite two problems being PPAD-COMPLETE, Ikenmeyer et al. [1] has showed examples where their counting difficulty can differ vastly as it can be seen in the figure 2.

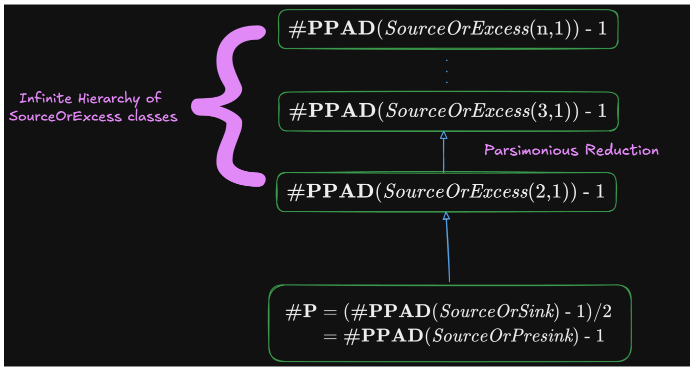

Infinite Hierarchy of SourceOrExcess classes
$\#\mathbf{PPAD}(\mathit{SourceOrExcess}(n,1))$ - 1
$\#\mathbf{PPAD}(\mathit{SourceOrExcess}(3,1))$ - 1
Parsimonious Reduction
$\#\mathbf{PPAD}(\mathit{SourceOrExcess}(2,1))$ - 1
$\#\mathbf{P} = (\#\mathbf{PPAD}(\mathit{SourceOrSink})$ - 1)/2
$= \#\mathbf{PPAD}(\mathit{SourceOrPresink})$ - 1

Fig. 2: Hierarchy graph between #PPAD problems. Figure from [2]

3) Kleene Logic and Hazard-Free Circuits: Kleene logic uses the boolean system with an additional *unstable* value [7]. Study of Kleene logic aided in the development of robust physical systems [15], as well as giving definitions to *Alternative Turing mahcines* [16], and showing correlations complexity of monotone circuits sizes [17]–[20].

Our current report will focus on specific concepts with Kleene logic as PURECIRCUIT instances use the underlying logic directly. Below we introduce important concepts [21].

Definition II.15: Kleene value ordering [21]

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

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

Definition II.16: Kleene Resolutions [18], [21]

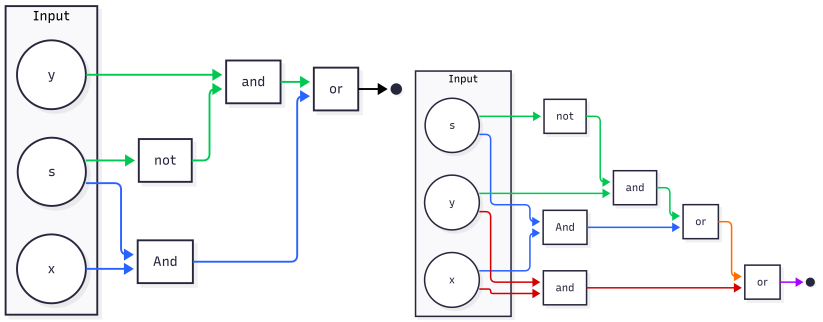
Given $x \in \mathbb{T}^n$, we define the *resolutions* of x, using the following notation:

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

Detecting hazards is a concept that is analysed heavily when talking about Kleene logic. The idea is ensuring robustness of our circuits, meaning if all resolutions of $x \in \mathbb{T}^n$ give the same value, then we should expect circuit to behave the same way II.17. An example of hazard values can be seen in the figure 3.

Definition II.17: Hazard [17], [18]

A *circuit* C, on n inputs has **hazard** at $x \in \mathbb{T}^n \iff C(x) = \perp$ 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.



(a) Multiplexer with hazard at $(1, 1, \perp)$ (b) Hazard free multiplexer

Fig. 3: Hazard circuit and hazard-free circuit. Figure by [18]

Definition II.1 (K-bit Hazard)**.** *For $k \in \mathbb{N}$ at $x \in \mathbb{T}^n \iff C$ has a hazard at x and $\perp$ appears at most k times in x*

III. Current Progress

A. *Current Timetable*

B. *Theory Progress*

In the current section we will be focusing on progress made. We experimented with various approach such as investigating easier variants of PureCircuit to find reductions to the EndOfLine. Moreover, we looked into using the Purify gate or a variant where we have more control over the $\perp$ value but we realised that under Marino et al. [22] continuity arguments, such variants are not trivial to find. Our core breakthrough, came when experimenting with Hazard-Free circuits and EOPL problems. Specifically, we looked at a problem by Fearnley et al. [23] where they introduced a variant brouwer problems known as OPDC, which is a local descent problem with fixed point topologies. Combining ideas of the two lead to the main find of our project in III-B1

1) nD-StrongSperner *Constant Parsimonious Reduction:* Deligkas et al. [6] proved PPAD-hardness by reducing from StrongSperner II.11, but with unbounded dimensions of polynomial width. Our contribution parsimoniously constant reduces II.3 every nD-StrongSperner problem to PureCircuit. We will demonstrate our reduction for 3 dimensions and then we will generalise.

Theorem III.1: 3D-StrongSperner to PureCircuit

$$\#\text{3D-StrongSperner} \subseteq^{19} \#\text{PureCircuit}$$

When referring to the fact that x is a solution, we imply the definition described in II.12. Additionally we will restrict the solution set of the *Purify* gate of PureCircuit to $(0,\perp),(\perp,1),(0,1),(1,\perp)$. We explain in II-B2a why this does not affect the PPAD-hardness of the problem nor any counting arguments.

Proof. The first step of the proof requires constructing a colouring function $\Lambda : (\{0,1\}^{n+1})^3$ such that:

$$\forall (i,j,k) \in (\{0,1\}^{(n+1)})^3 : \Lambda(i,j,k) \triangleq \lambda\left(\left\lfloor \frac{i+1}{2} \right\rfloor, \left\lfloor \frac{j+1}{2} \right\rfloor, \left\lfloor \frac{k+1}{2} \right\rfloor\right)$$

Conversely we can say that we map from our original domain to our new domain as such

$$\forall x \in \{0,1\}^{3n}, i \subseteq \{0,1,2\} : \lambda(x_{-i}, x_i) \to \Lambda(2x_i, 2x_{-i} - 1) \tag{1}$$

The transformation can be visualised in the figure 4. The reasoning for doubling the dimensions will be explained in the later sections.

Given that S is the original set of solution, we define the claim III.1.

Claim III.1: Transformation claim

Given S_{even} the set of even solutions or more formally:

$$S_{\text{even}} \triangleq \left\{(i0, j0, k0) \in (\{0,1\}^{(n+1)})^3 \mid (i0, j0, k0) \text{ covers all labels by II.10}\right\}$$

We claim $|S| = |S_{\text{even}}|$

Proof. We argue that we can bijectively map between $S \leftrightarrow S_{\text{even}}$. To show the left to right direction, assume $(i,j,k) \in S$. Using equation 1, only one transformation maps all points to even coordinates. Additonally, we can observe its neighbourhood set:

$$N = \left\{\Lambda(2i + x_1, 2j + x_2, 2k + x_3) \mid x \in \{0,1\}^3\right\}$$

We can observe that WLOG $\Lambda(2i+1,\cdot,\cdot) = \lambda(i+1,\cdot,\cdot)$, which implies:

$$N = \left\{\lambda(i + x_i, j + x_j, k + x_k) \mid x \in \{0,1\}^3\right\}$$

And therefore the the point is also a solution. Conversely we can use the previous neighbourhood argument to show the opposite direction. □

We will redefine $n+1$ as n to keep the notation consistent. Given the above we provide a main sketch of our tranformation as such:

1) **Input bits**: We initialize input nodes s_1, s_2, s_3
2) **Bit generator**: We apply the *Bit Generator gadget* $\hat{P}$ for each s_i, to create numbers $u^1, u^2, u^3 \in \{0,1\}^k$

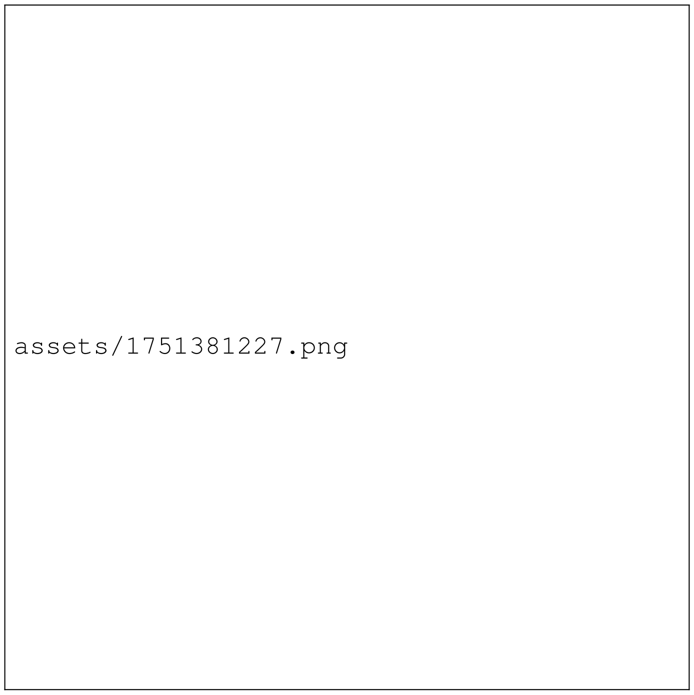


Fig. 4: Transformation of solutions. In the original cube we have as solutions $(0, i, j)$ and $(1, i, j)$. In our new mapping, these solutions correspond to $(0, i, j), (2, i, j)$

3) **Circuit application**: We apply circuit $\bar{\Lambda}(u^1, u^2, u^3)$ to create output values o_1, o_2, o_3, where $\bar{\Lambda}$ is a 3-bit hazard-free variant of Λ.
4) **Validation**: Copy the results to s_1, s_2, s_3

The transformation can be summarised in the figure 5. The goal of the above transformation is to show that if $o_1 = o_2 = o_3 = \perp$, then we have a solution to the original instance.

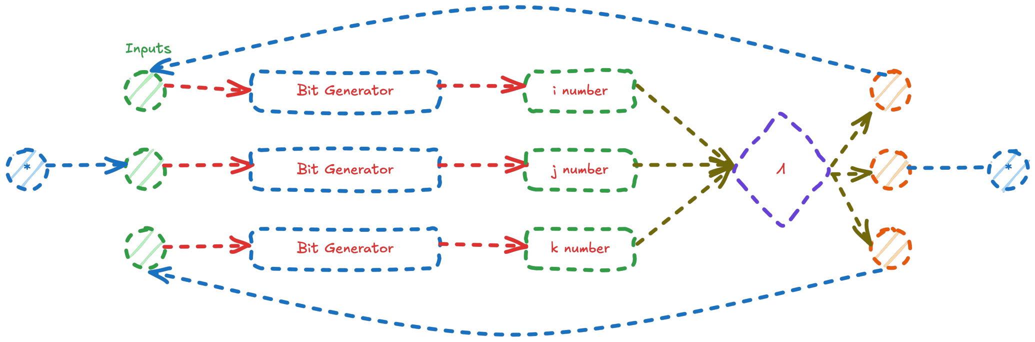


Fig. 5: Testing

a) Bit generator: We create the gadget described in the figure 6.

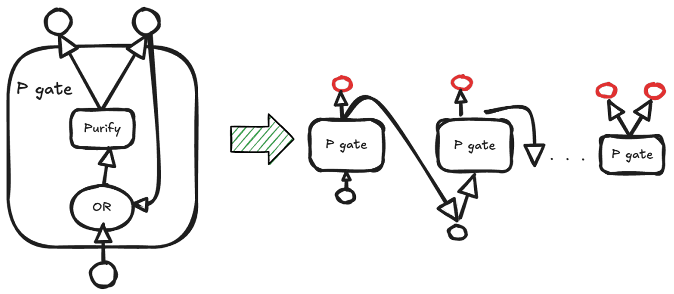


Fig. 6: Bit generator gadget which is defined as a linear stack of P gates.

Given that construction we make the following lemma

Lemma III.1 (Bit Generator Lemma)**.** *The following hold true in our construction:*

1) *if* $s_i = b \implies u^i = b^n$, *given* $b \in \mathbb{B}$
2) *if* $s_i = \perp \implies \forall j \in [n-1] : u^i_j \in \mathbb{B}$ *and* $u^i_n \in \{1, \perp\}$

The above translates to: if we have $\perp$ *in our number, it can only be found in the LSB.*

Proof. The first part follows trivially from the defintion of the PURIFY gate. To prove the second part of lemma, WLOG we choose u^i out of the three outputs and assume $\exists j \in [n-1] : [u^i]_j = \perp$. It implies that one of the purify gates had an output of $(\perp, 1)$. But due to our OR gate, the input would be 1 which implies the output is $1, 1$, which leads to a contradiction. ☐

b) Circuit Application: In the current phase, we apply a polynomial transformation to Λ to be 3-bit hazard-free using the construction by Ikenmeyer et al. [18] *Corollary 10* or by [20] *Corollary 1.9*. By lemma III.1, we can have at most 3 $\perp$ values in our input. Using that property we create the lemma below:

Lemma III.1: Circuit Application Lemma

Given $o_1 = o_2 = o_3 = \perp$ it implies u^1, u^2, u^3 correspond to a solution $(i0, j0, k0) \in S_{\text{even}}$

Proof. First we argue that if $i\perp, j\perp, k\perp = u^1, u^2, u^3$, then its its resolutions can represent hypercubes as shown in figure 7.

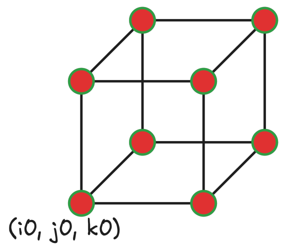


Fig. 7: Resolutions of $i\perp, j\perp, k\perp$

If $(i0, j0, k0)$ is a solution that implies:

$$\forall i \in \{1,2,3\}, b \in \{-1,1\} \exists c \in \text{RES}(i\perp, j\perp, k\perp) : [\bar{\Lambda}(c)]_i = b$$

Since $\bar{\Lambda}$ is 3-bit hazard free, if all outputs are $\perp$, we satisfy the colouring criteria. ☐

The above proof also demonstrates why the domain transformation we did in our first step is needed as we only capture cubes with even smallest corners. We can observe by fixing the LSB of the input to some value, we extract a specific edge or face of the cube.

Lemma III.2: Correctness lemma

A valid assignment x of the current instace, corresponds to a point in S_{even}

Proof. It suffices to show that a valid assignment only occurs when $o_1 = o_2 = o_3 = \perp$. Assume $\exists i \in \{1,2,3\}$ such that label i is not covered, meaning $o_i \neq \perp$. WLOG assume that $o_i = 0$. Our verification stage, will copy 0 onto s_i. By lemma III.1, $u^{(i)} = 1^n$. From the boundary conditions of our circuits and the k-bit hazard-freeness construction, we know that $\Lambda(*, 0^n, *) = \{*, 1, *\}$. But that implies $o_i = 1$ which leads to a contradiction. We can make a similar argument to when

$o_i = 1$. Therefore, by III.1, if $o_1 = o_2 = \perp$, we have a found a panchromatic cube. Lastly, every assignment u^1, u^2, u^3k corresponds to a cube with even corners. $\square$

$\square$

Counting Argument: We observe that the LSB can only be $1, \perp$. Moreover, we double count any edges or faces of the cube that cover all labels, which gives the equation:

$$\underbrace{\binom{3}{1}}_{\text{One of the sides is odd and covers all labels}} + \underbrace{\binom{3}{2}}_{\text{One of the edges is odd and covers all labels}} + \underbrace{1}_{\text{All LSBs are } \perp} = 2^3 - 1 = 7$$

Therefore, we can bound the number of solutions of 3D-STRONGSPERNER by a factor of 7.

ENDOFLINE problem parsimoniously reduces to 2D-STRONGSPERNER and 3D-STRONGSPERNER under **linear** colouring [8], [9], but more work has to been done in order to determine whether this still holds for bipolar colouring.

The reduction described above can work with any dimensionality. and we know $foralln \in \mathbb{N}_{\geq 2}$, Chen et al. showed that nD-STRONGSPERNER is still *PPAD-Hard* in [10]. Therefore the following corollary holds III.1

Corollary III.1: nD-STRONGSPERNER **parsimonious reduction bounds**

$$f(\cdot) = 1 + \sum_{i=1}^{n-1} \binom{n}{i} = 2^n - 1 = a_n$$

$$\#\text{nD-STRONGSPERNER} \subseteq^{a_n} \#\text{PURECIRCUIT}$$

2) Hazard Circuits and Tarski: We define the following subclass of TARSKI **??**:

Definition III.1: KLEENETARSKI **problem definition**

Given $F : \mathbb{T}^n \to \mathbb{T}^n$, where F is a *natural* function, we want to find the set of points **Fix*** such that

$$\forall x \in \textbf{Fix}^\star : F(x) = x$$

We assume F will be represented as a circuit, that uses $\{*, +, \neg, \mathbf{0}, \mathbf{1}\}$.

Proposition III.2 (Parsimonious Reduction between KLEENETARSKI and PURECIRCUIT)**.**

$$\#\text{KLEENETARSKI} \subseteq \#\text{PURECIRCUIT}$$

Proof. Given an input circuit C, consturct PURECIRCUIT instance:

1) Initiate a vector of nodes s
2) Apply $C(x)$ and store result on output vector o.
3) Copy the results back into s

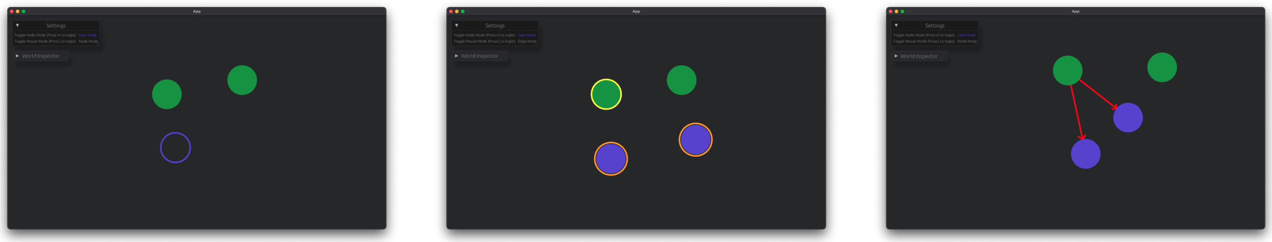
We can observe that for any valid assignment, $\mathbf{x}[s] = \mathbf{x}[o]$. Therefore, our construction can find all fix points for Kleene Circuits. $\square$

C. Software Progress

In our project we are also focused on developing a visualisation tool for PURECIRCUIT instances. In our introduction we referred to three main objectives. As of time of writing w are close to the completion of the first one. We will provide a table of objectives that have been achieved, as well as the remaining requirements needed to complete the first milestone in III. Lastly, we provide a figure of our current visualisation in 8

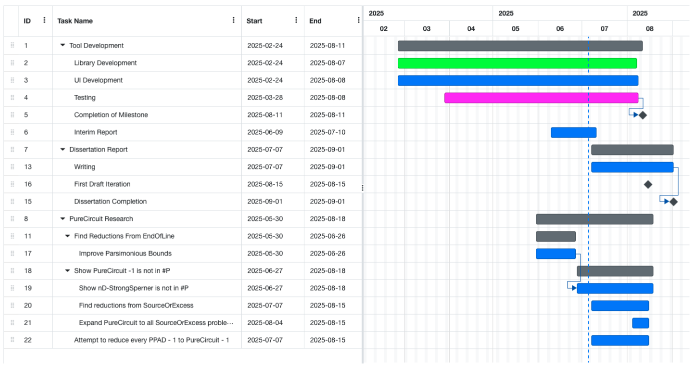
Accomplished	Finished
Ability to add and move nodes	Add values to value nodes
Toggle between value nodes and gate nodes	Specify the type of gate to add
Add edges. Ensure that edges can only be added between heterogeneous nodes	Add indicators as to how many edges are excess/remaining for the gate to be valid
Move whole graph	Inclusion of a status bar as to whether the current assignment is correct, wrong or syntactically incorrect
Create panel to show current state as well as some indicator and guides	

TABLE III: Finished and remaining issues



(a) Add nodes (b) Add edges (c) Final state

Fig. 8: Screenshots of different states of the visualisation tool



ID	Task Name	Start	End
1	Tool Development	2025-02-24	2025-08-11
2	Library Development	2025-02-24	2025-08-07
3	UI Development	2025-02-24	2025-08-08
4	Testing	2025-03-28	2025-08-08
5	Completion of Milestone	2025-08-11	2025-08-11
6	Interim Report	2025-06-09	2025-07-10
7	Dissertation Report	2025-07-07	2025-09-01
13	Writing	2025-07-07	2025-09-01
16	First Draft Iteration	2025-08-15	2025-08-15
15	Dissertation Completion	2025-09-01	2025-09-01
8	PureCircuit Research	2025-05-30	2025-08-18
11	Find Reductions From EndOfLine	2025-05-30	2025-06-26
17	Improve Parsimonious Bounds	2025-05-30	2025-06-26
18	Show PureCircuit -1 is not in #P	2025-06-27	2025-08-18
19	Show nD-StrongSperner is not in #P	2025-06-27	2025-08-18
20	Find reductions from SourceOrExcess	2025-07-07	2025-08-15
21	Expand PureCircuit to all SourceOrExcess proble…	2025-08-04	2025-08-15
22	Attempt to reduce every PPAD - 1 to PureCircuit - 1	2025-07-07	2025-08-15

Fig. 9: Updated gantt chart

IV. Next Steps

A. Theory Crafting Next steps

For the remaining duration, we will focus on three main objectives: One will be to demonstrate that, the *bipolar* colouring of ND-STRONGSPERNER will keep the reduction parsimonious. Second, will be to demonstrate hardness over #P class either by findning reductions from the SOURCEOREXCESS problem. And lastly, we will be to generalise our reduction to all $\#PPAD$ problems.

B. Software Next Steps

In the next steps we aim to complete the rest of the milestones. We believe that the usage of methods by Eichelberger et al. [17] and or polynomial algorithm for specific cases by [6] will allow us to identify solutions. With regards to counting, we can only hope to count solutions for small instances.

C. Timetable

The figure 9 depicts how the remaining project will progress with regards to the milestones we aim to achieve. Overall we modified our timeline to prioritise theory crafting and therefore we hope this adjusted timeline will capture accurately the remaining duration of the project.

V. Project Management

In order to refer to our project management, we will talk about the toolings and methods we used, as summarised in the figure 10.

Our project also includes the development of a visualisation tool for creation and verification of several PURECIRCUIT instances. We opted with *Rust*, as the main language of development, due to its high and low level features. From the one hand, *Rust* can efficiently handle memory allocation safely with its clever usage of the Borrower-Ownership framework. Conversely, it implements a strongly typed system with help of generics, associated types and algebraic types allowing us to create a versatile and compact library. Given the above, we utilise techniques such as `Proptest`, where we create strategic randomized tests that check whether function follows an expected property. On the other hand we make use of `Petgraph`, which is a sophisticated library that handles graph-like structures in *Rust*.

For visualisation we decided to go with *Bevy*. *Bevy* is a game engine that uses the *ECS* software architecture. A general workflow of an *ECS* system can be described as such: a state contains entities, each of which is composed of components

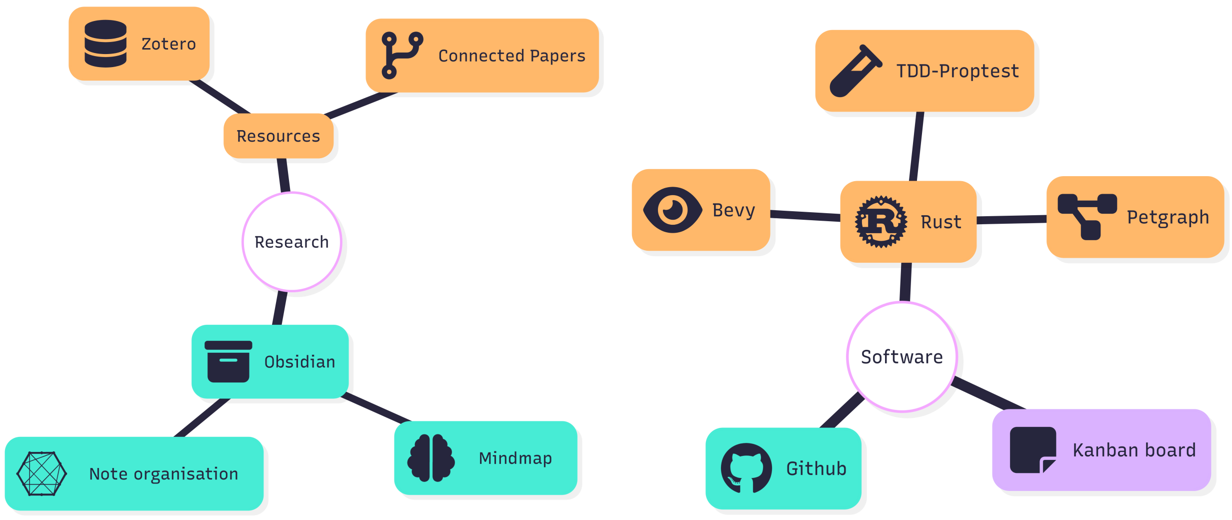


Fig. 10: Tooling

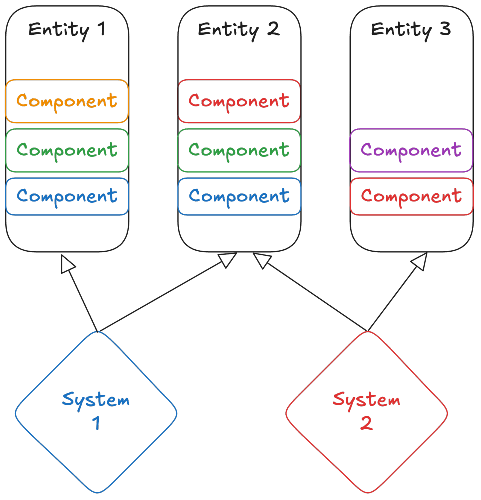


Fig. 11: ECS workflow visualisation

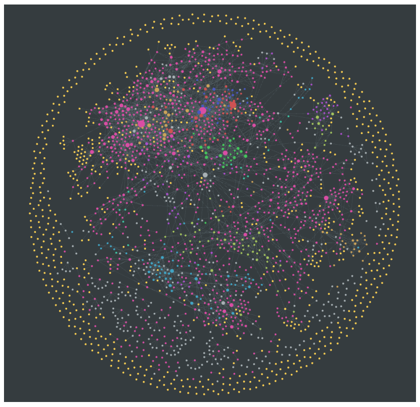
or properties. A system is a specialised method that gathers entities based on their components and describes an interaction between them. This whole process can be visualised in the figure 11

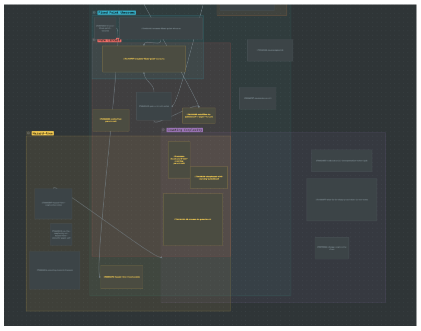
With regards to the our research management, our core tool came through the usage of obsidian. As we can see in the figure 10, `Obsidian` beyond its traditional usage of note taking, it comes with several handy tools such as note organisation and mind-mapping. These can be seen in the figure 12, where we utilised connections across notes as well as its drawings or other thought organisation tools in order experiment and manage ideas.

A. *Risk Management*

With regards to risks and mitigations, the biggest risk we face is the inability to resolve our main question. Due to the limited literature surrounding PURECIRCUIT and its counting nature as well as the recency in the development of $TFNP-1$, our question could go into many directions. The discovery of our reduction allowed us to step closer to our question and in our table V, we will analyze this is greater depth. The general mitigation strategy we can do is to keep researching, keep finding connections until we can connect all the necessary pieces to do the jump.

Severity	**Probability**	**Mitigation**	**Mitigation**	**Address**
High	Medium	Software may not be feasible within the remaining time frame.	Focus on the the first objective where the project. Restrict to solution finding or counting when the number of nodes is small.	Not yet
Low	Medium	Software not identifying a correct solution	Usage of TDD techniques and property testing to ensure correctness. Comparison with hand-made instances	Not yet


(a) Obsidian Graph


(b) Obsidian Canvas

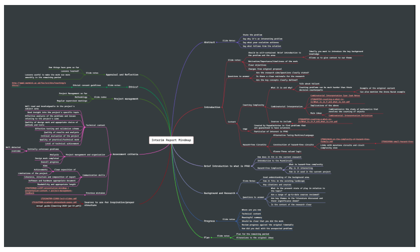

(c) Obsidian mindmap

Fig. 12: Usages of Obsidian

From our table, it is worth expanding on some of the points. The best method we found when tackling this problem is when we try to uncover reductions between other PPAD problems or when trying to incorporate gadgets from Kleene theory. We hope that by expereminting enough we will be able to get close to resolve our conjectures. In addition, for the last point, we made several efforts to ensure our problem is PPAD-complete and combinatorial friendly, by trying to restrict the PURIFY gate. We soon realised that any methods that detect the $\perp$ value or try to eliminate ultimately fail. The observation can be based on a continuity argument that Marino created [22] and in general any possible extensions or gates we can add seem to adhere continuity. (TODO)

VI. Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

References

[1] I. Pak, "What is a combinatorial interpretation?" Sep. 2022.

[2] 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)*, Oct. 2022, pp. 860–871.

[3] L. G. Valiant, "The complexity of computing the permanent," *Theoretical Computer Science*, vol. 8, no. 2, pp. 189–201, Jan. 1979.

[4] 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*, vol. 2024, no. 10, pp. 8442–8458, May 2024.

[5] C. H. Papadimitriou, "On the complexity of the parity argument and other inefficient proofs of existence," *Journal of Computer and System Sciences*, vol. 48, no. 3, pp. 498–532, Jun. 1994.

[6] A. Deligkas, J. Fearnley, A. Hollender, and T. Melissourgos, "Pure-Circuit: Tight Inapproximability for PPAD," *J. ACM*, vol. 71, no. 5, pp. 31:1–31:48, Oct. 2024.

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

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

[9] X. Chen and X. Deng, "On the complexity of 2D discrete fixed point problem," *Theoretical Computer Science*, vol. 410, no. 44, pp. 4448–4456, Oct. 2009.

[10] X. Chen, X. Deng, and S.-H. Teng, "Settling the complexity of computing two-player Nash equilibria," *J. ACM*, vol. 56, no. 3, pp. 14:1–14:57, May 2009.

[11] 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*, ser. STOC 2021. New York, NY, USA: Association for Computing Machinery, Jun. 2021, pp. 1466–1478.

[12] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, "How easy is local search?" *Journal of Computer and System Sciences*, vol. 37, no. 1, pp. 79–100, Aug. 1988.

[13] K. Bronisław, "Un theoreme sur les functions d'ensembles," vol. 6, pp. 133–134, 1928.

[14] J. Fearnley, D. Pálvölgyi, and R. Savani, "A Faster Algorithm for Finding Tarski Fixed Points," *ACM Trans. Algorithms*, vol. 18, no. 3, pp. 23:1–23:23, Oct. 2022.

[15] S. Friedrichs, M. Függer, and C. Lenzen, "Metastability-Containing Circuits," *IEEE Transactions on Computers*, vol. 67, no. 8, pp. 1167–1183, Aug. 2018.

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

[17] E. B. Eichelberger, "Hazard Detection in Combinational and Sequential Switching Circuits," *IBM Journal of Research and Development*, vol. 9, no. 2, pp. 90–99, Mar. 1965.

[18] C. Ikenmeyer, B. Komarath, C. Lenzen, V. Lysikov, A. Mokhov, and K. Sreenivasaiah, “On the complexity of hazard-free circuits,” *Journal of the ACM*, vol. 66, no. 4, pp. 1–20, Aug. 2019.
[19] C. Ikenmeyer, B. Komarath, and N. Saurabh, “Karchmer-Wigderson Games for Hazard-free Computation,” Nov. 2022.
[20] J. Bund, C. Lenzen, and M. Medina, “Small Hazard-Free Transducers,” *IEEE Transactions on Computers*, vol. 74, no. 5, pp. 1549–1564, May 2025.
[21] M. Mukaidono, “On the B-ternary logic function,” *Trans. IECE*, vol. 55, pp. 355–362, Jan. 1972.
[22] L. R. Marino, “General theory of metastable operation,” *IEEE Transactions on Computers*, vol. C-30, no. 2, pp. 107–115, Feb. 1981.
[23] J. Fearnley, S. Gordon, R. Mehta, and R. Savani, “Unique end of potential line,” *Journal of Computer and System Sciences*, vol. 114, pp. 1–35, Dec. 2020.