# Counting Problem of *PureCircuit*

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

**Abstract**

Test.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.

**Index Terms**

Complexity Theory, Counting Complexity, Hazard Free Circuits, TFNP, PPAD, Search Problems, Kleene Logic

## 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 combinatronics, 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 combinatronics 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 assingments 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 (-). 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. Core set of results

Below we will present our main list of findings up to this point:

**Theorem I.1** (ND-Brouwer to PureCircuit)**.** *Given* $f(x) \triangleq 20$

$$\#\textsc{Brouwer}^{f_3} \subset^f \#\textsc{PureCircuit}$$

**Corollary I.1.1.** *For any* $d \in \mathbb{N}_{\geq 2}$*, we can define* $f(\cdot) = \sum_{i=1}^{d-1} \binom{d}{i} 2^i$ *such that:*

$$\#\textsc{Brouwer}^{d} \subset^f \#\textsc{PureCircuit}$$

**Theorem I.2.** *For any* $d \in \mathbb{N}_{\geq 2}$*, we can define* $f(\cdot) = 5$ *such that:*

$$\#\textsc{Brouwer}^{d} \subseteq^f \#\textsc{PureCircuit}$$

Throughout our search, we stumbled upon various variants of the problem as well as reductions and claims from other subclasses of PPAD or Hazard-Free logic

**Proposition I.3.** *Weaker variants of PureCircuit can result to parsimonious reductions to the EndOfLine problem as such:*

$$\#\textsc{AcyclicBPureCircuit} \subseteq \#\textsc{EndOfLine}$$
$$\#\textsc{PermutationFreeBPureCircuit} \subseteq \#\textsc{EndOfLine}$$

In addition, we were able to show that for different promise problems, we can find reductions to the PureCircuit problem

**Proposition I.4.**

$$\#\textsc{PUnsatHazard} \subseteq \#\textsc{PureCircuit}$$

Lastly we introduce the following proposition

**Proposition I.5.** *Given function* $F : \mathbb{T}^n \to \mathbb{T}^n$ *where* $\mathbb{T} \triangleq \{0, 1, \bot\}$ *and* $F$ *is a* ***natural*** *function:*

$$\exists x^\star \in \mathbb{T}^n : F(x^\star) = x^\star$$

The idea above is based on the Tarski Fixed point theorem. Using fixed points with the Kleene logic set, has been used in the past by Kozer as seen in his book [3], but we are extending such ideas to any possible functions or gates that use monotone gates. Doing that allows us to make the following observation.

**Proposition I.6.**

$$\#\textsc{HazardTarski} \subseteq \#\textsc{PureCircuit}$$

## II. Preliminaries and Background review

### A. Important Notation

Here we will introduce important notation to avoid confusion:

- $\forall n \in \mathbb{N} : [n] \triangleq \{1, ..., n\}$
- $\forall n \in \mathbb{N} : [n]_0 \triangleq [n] \cup \{0\}$
- $\forall n \in \mathbb{N} : \bar{n}$ implies the unary representation of the number $n$ and $n$ by default implies the binary
- $A^B \triangleq \{f \mid f : A \to B\}$
- $\mathbb{B} \triangleq \{0, 1\}$
- $\mathbb{N} \triangleq \mathbb{Z}_{\geq 1}$
- $\mathbb{N}_0 \triangleq \mathbb{Z}_{\geq 0}$

### B. Background overview

As mentioned in the introduction, the current project works as an interaction of three different fields: 1. counting complexity and combinatorial interpretation, 2. total search problems and PPAD and 3. Kleene logic.

*1) Counting Complexity and Combinatorial Interpretations:* As we previously mentioned, we say that an object or a structure $f$ has a combinatorial interpretation if $f \in$ **#P**. **#P** is a complexity class created by Valiant [4], to define a formal combinatorial framework in complexity theory.

**Definition II.1** (**#P** Complexity Class)**.** ***#P*** *is a class of functions* $f : \{0, 1\}^* \to \mathbb{N}$ *such that: there exists a polynomial time non-deterministic Turing machine* $N : \{0, 1\}^* \to \{0, 1\}$ *such that*

$$\forall w \in \{0, 1\}^* : f(w) = \#acc_N(w)$$

*Where* $\#acc_N(\cdot)$*, denotes the number of accepting paths. Equivalently, we may also use the definition of: There exists a polynomial deterministic TM* $M$*, and* $p : \mathbb{N} \to \mathbb{N}$ *such that* $p \in n^{O(1)}$*, we have:*

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

**#P** was initially created by [4], to demonstrate, that even if we have a problem $L \in P$, $\#L$ can be computationally hard to compute, by providing an example of computing the permanent of a 01-matrix, with number of perfect matchings.

As we can see, **#P** allows us to define a set of objects, whose cardinality equals $f(w)$. Core reasoning for choosing **#P** to define combinatorial objects is mainly for the following two reasons [5]:

1) By polynomially bounding words, we avoid cases such as: $f(w) = \{1, \dots, f(w)\}$
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 [6] and [5] 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 [6], where they demonstrated how several TFNP problems, change in complexity as we ignore one of its solutions.

Any class of problems in complexity theory utilises the notion of reduction to indicate the similarity between problems. For counting complexity this is referred to as *parsimonious reductions*.

**Definition II.2** (Parsimonious reductions)**.** *Let $R, R'$ be search problems and let $M$ be a Karp 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:*

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

*2) Total Search Problems and PPAD:* When talking about search problems, we are using the following definition:

**Definition II.3** (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:*

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

Using the above, we can define the following complexity classes

**Definition II.4** (FP, FNP, TFNP)**.** ***FP*** *are search problems such that there exists poly-time TM $M$ such that $M(x) = y$ where $xRy$.*

***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.5** (*EndOfLine* problem)**.** *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 are source or sinks $\forall v \in V$ : deg$(v) = (0,1)$ or deg$(v) = (1,0)$, respectively. We want to output $v$ that is either a source or a sink.*

**Definition II.6** (Levin Reductions)**.** *Given a pair of search problems $R_A, R_B$, a pair of computable time functions $(f, g)$ is called a Levin reduction from $R_A \to R_B$*

$$S_R \triangleq \{x \mid \exists y : xRy\}$$
$$R(x) \triangleq \{y \mid xRy\}$$
$$\forall x \in S_{R_A}, y_b \in R(f(x)) : (x, g(x, y_b)) \in R_A$$

**Definition II.7** (**PPAD** complexity class)**.** ***PPAD*** *is defined as the set of search problems that are levin reducible to the* ENDOFLINE *problem*

**PPAD** has been created by Papadimitriou [7] to demonstrate a subset of problems in **NP** that are guaranteed to have a solution but can be very difficult to find. We will define several problems of interest as they will be referenced in later sections.

**Definition II.8** (*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:*

$$\forall x, y \in V^2 : (x,y) \in E \iff (S(x) = y) \wedge \bigvee_{i \in [k]} P_i(y) = x$$

*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$(v)$*

**Definition II.9** (StrongSperner problem)**.** ***Input****: A boolean circuit computing a labelling $\lambda : [M]^N \to \{-1,1\}^N$ 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: $\forall i, j \in [N] : \|x^{(i)} - x^{(j)}\|_\infty$*
- *Covers all labels:*

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

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. For the sake of clarity, we will stick to StrongSperner and we will refer to StrongSperner$^n$ as the problem with a fixed dimensionality of $n$. This problem is crucial as it show **PPAD**-Hardness as well as relating to the reduction of various problems. We mainly use this to show counting arguments with respect to the EndOfLine as people have indicated a parsimonious reduction between StrongSperner$^2$ to a variant of the EndOfLine and StrongSperner$^3$ to the EndOfLine problem but with different colourings.

These problems have found connections with various other problems such as Nash Equilbria, Fixed point theorems or Sperner Lemmas. The current project looks on the counting complexity of such problems as despite two problems being PPAD-complete, Ikenmeyer et al. has showed several examples where their counting difficulty can differ vastly. Examples of such statements can be summarised as:

$$\textbf{\#PPAD}(\text{SourceOrSink}) - 1/2 = \textbf{\#P}$$
$$\textbf{\#P}^A \not\subseteq \textbf{\#PPAD}(\text{SourceOrExcess(2,1)})^A - 1$$

Currently we are mainly focused into the study of a certain **PPAD**-*complete* problem, known as PureCircuit

*3) Kleene Logic and Hazard-Free Circuits:* Kleene logic, is the type of system that extends the traditional binary system to: $\mathbb{T} = \{0, \perp, 1\}$. Study of such systems has an impact to both the development of robust systems and from a theoritical point of view, people developed systems such as *Alternative Turing mahcines*, as well as showing important correlations between the complexity of monotone circuits and the robustness of hazard free circuits.

We want to introduce the following concepts:

**Definition II.10** (Kleene Resolutions)**.** *Given $x \in \mathbb{T}^n$, we define the resolutions of $x$, using the following notation:*

$$\text{Res}(x) \triangleq \left\{y \in \mathbb{B}^n \mid \forall j \in [n] : x_i \in \mathbb{B} \implies x_i = y_i\right\}$$

**Definition II.11** (Hazard circuit)**.** *A circuit $C$, on $n$ inputs has **hazard** at $x \in \mathbb{T}^n \iff C(x) = \perp$ and $\exists b \in \mathbb{B}, \forall r \in \text{Res}(x) : C(r) = b$*

**Definition II.12** (Natural function)**.** *A function $F : \mathbb{T}^n \to \mathbb{T}$ can be computed by stable values and is monotone with respect to $\preceq$*

**Definition II.13** (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. Theory Progress

*1) EndOfLine to PureCircuit Constant Parsimonious Reduction:* We will define the necessary prerequisites for the current section:

**Definition III.1** (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:*

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

**Proposition III.1** (EndOfLine to 3D-StrongSperner Parsimonious reduction)**.**

$$\#\text{EndOfLine} \subseteq \#\text{3D-StrongSperner}$$

The above has been shown by Our main contribution comes from the following theorem

**Theorem III.2** (3D-StrongnSperner to PureCircuit)**.** *Given $f(\cdot) = 20$, we argue*

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

*Where* 3D-StrongSperner *is represented as a tuple $(\lambda, 0^n)$ where $n$ corresponds to grid size of $2^n$.*

To show the above holds true we will first show the construction and prove its correctness. To simplify the counting argument, we modify a solution of 3D-StrongSperner as such: Assuming $S \subseteq \{0,1\}^n$ is the set of all solutions, such that we define $S$ as such:

$$S \triangleq \Big\{(i,j,k) \in (\{0,1\}^n)^3 \mid \{\lambda(i+x_1, j+x_2, k+x_3) \mid x \in \{0,1\}^3\}\Big\} \text{ covers all labels}$$

We will create $\Lambda : (\{0,1\}^{(n+1)})^3 \to \{-1,+1\}^3$ 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

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

The above transformation can be visualised using the following figure

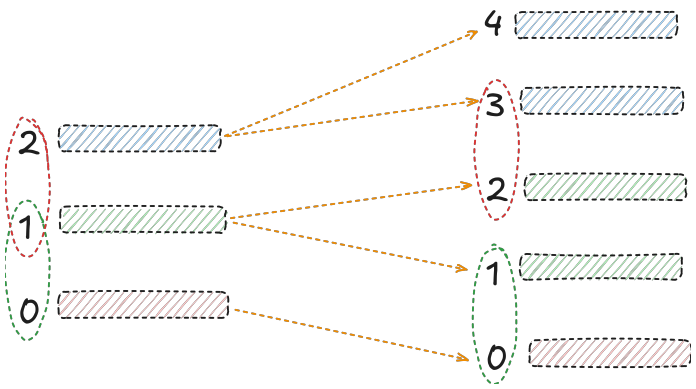



Fig. 1: The current figure allows depicts how solutions or pairs of solutions are mapped. For example for cube starting at $i = 0$ and $i = 1$, we now just have to look at $i = 0$ and $i = 2$.

**Claim III.3** (Transformation claim)**.** *We claim that $|S_{even}| = |S|$ where $S_{even}$ is defined as:*

$$S_{even} \triangleq \Big\{(i0, j0, k0) \in (\{0,1\}^{(n+1)})^3 \mid \{\Lambda(2i + x_1, 2j + x_2, 2k + x_3) \mid x \in \{0,1\}^3\}\Big\} \textit{ covers all labels}$$

*Proof.* We argue that we have a bijective transformation $S \leftrightarrow S_{\text{even}}$. To do that we will first show that every point in $x \in S$ is mapped to a single point in $x' \in S_{\text{even}}$. Assume $(i,j,k) = x$ for some $i,j,k \in \{0,1\}^n$. For a point to be in $S_{\text{even}}$ has to be the case all 3 coordinates are even. From our transformation, there is only one mapping that achieves that, which is when $j = \{1,2,3\}$. We have to verify that this point corresponds to the same set of points or when $x' = (2i, 2j, 2k)$. We can observe its neighbourhood set:

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

We can observe that if $x_i = 1$ for any $i \in 1,2,3$, that points corresponds to the same point as in $i + 1$. This implies that the set above corresponds to the neighbourhood:

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

And therefore the points match. We can conversely use the same argument to show the opposite direction and therefore we can conclude that $|S| = |S_{\text{even}}|$ □

For simplicity's sake I will rename $n + 1$ 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 $u^1, u^2, u^3 \in \{0,1\}^k$
3) **Circuit application**: We apply circuit modified circuit $\bar{\Lambda}$ to $u^1, u^2, u^3$ to create output values $o_1, o_2, o_3$
4) **Validation**: Copy the results back to $s_1, s_2, s_3$

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. The current construction allows us to match a single box of the original problem to up to 20 solutions of the PureCircuit one. We can visualise it as such:

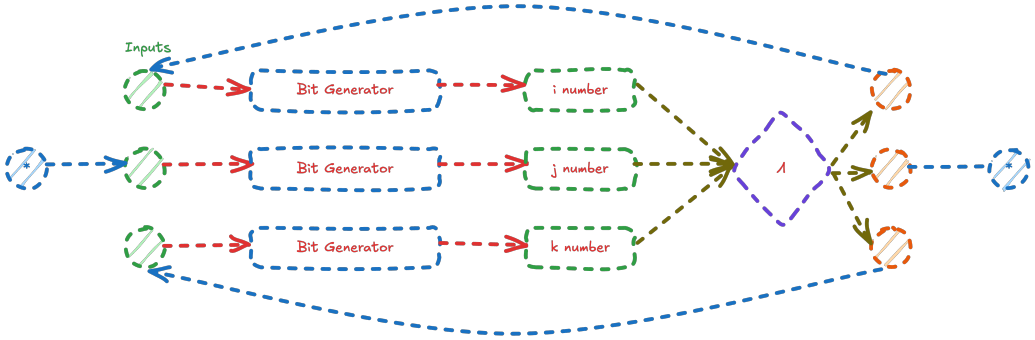



Fig. 2: Testing

Before we look at the construction, we will refer to the standard set of gates $\{*, +, \neg\}$. These can be trivially build using NOR gates. Lastly we only accept the following as valid outputs of the *Purify* gate: $(0, \perp), (1, \perp), (\perp, 1), (0, 1)$. It is easy to see that for these set of outputs, *PureCircuit* remains **PPAD**-complete, by using the continuity argument.

*a) Bit generator:* In order to generate our number we use the construction shown in the label 3,

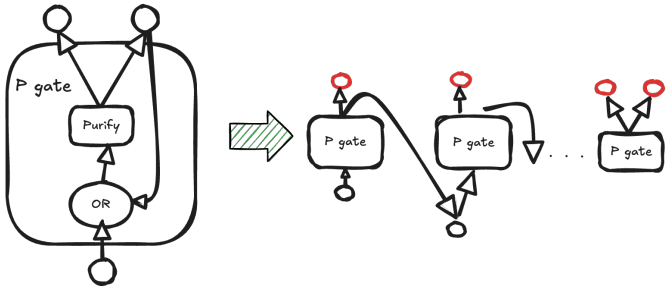



Fig. 3: Our bit generator bit is stacked version of a lot of $\hat{P}$ circuits. Be stacking them as shown in the figure and choosing the red nodes, we get our binary number.

Given that construction we make the following lemma

**Lemma III.4** (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* $u^i_n \in \{1, \perp\}$

*We essentially ensure that if there exists a* $\perp$ *in our numbe, it should only be found in the last digit*

*Proof.* The first part follows trivially from the defintion of the PURIFY gate, therefore if our input is a pure bit, we are copying it to all the inputs. To prove the second part of lemma, we use assume contradiction. Assume $\exists j \in [n-1]$. That would imply that one of the purify gates had an output of $(\perp, 1)$. But due to our OR gate, that would force the input to be 1 which would imply that the output is $1, 1$. This implies that the only bit that can be $\perp$ is the last bit. □

*b) Circuit Application:* In the current phase, we will modify the circuit $\Lambda$ to be hazard-free using the construction from the *Corollary 10* of [8]. They showed that we can ensure a k-bit hazard-free circuit with the following properties:

$$\text{size}(C) = \left(\frac{ne}{k}\right)^{2k} (|C| + 6) + O(n^{2.71k})$$
$$\text{depth}(C) = D + 8k + O(k \log n)$$

Using the bit generator lemma III.4, we can observe that we can have at most 3 $\perp$ values in our input. This implies that we can create a circuit of polynomial size such that it is hazard-free. Using that property we create the lemma below:

**Lemma III.5** (Circuit Application Lemma)**.** *If $o_1 = o_2 = o_3 = \perp$ implies that we covered all the labels and found a solution in $(i0, j0, k0) \in P_{even}$*

*Proof.* First part is to understand what our $u^{(1)}, u^{(2)}, u^{(3)}$ represent. To do that, we use the following visualisation on the figure 4 The first $n-1$ bits denote the $i, j, k$ indexes of our original solution. The cube essentially represents all possible realisations

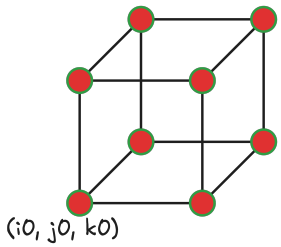



Fig. 4: Representation of our solutions

of $i\perp, j\perp, k\perp$. We know that the neighbouring corners correspond to vertices in our original instance. When all LSBs are $\perp$, we can observe that we are calculating our $\Lambda(\cdot)$ for all corners of the cube simultaneously. If $(i0, j0, k0)$ is a solution that implies:

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

That implies if $\bar{\Lambda}$ outputs $\perp$ in some dimensions, then it found two points with different labels. We can therefore argue that if the output it $\perp^3$, then we covered all the labels. □

The above section conclude our construction. We will now demonstrate the correctness argument:

**Lemma III.6** (Correctness lemma)**.** *We argue that every correct assingment of the above circuit, corresponds to a point in $P_{even}$*

From the transformation claim III.3, we know that if our circuit can find all corresponding points in $P_{\text{even}}$, we can find the all the solutions of the original circuit.

*Proof.* To prove our lemma, it suffices to show that $o_1 = o_2 = o_3 = \perp$. Let's assume by contradiction that $\exists i \in \{1,2,3\}$ 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 our bit generator lemma III.4, we have that $u^{(i)} = 1^n$. From the boundary conditions of our circuits, and the k-bit hazard-freeness construction by III.5, 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 $o_i = 1$. □

*Counting Argument:* To find how many solutions correspond to a single solution of the original instance, we can make the following observation: Looking at our cube 4, if a side of the cube contains all labels, or if an edge contains all labels, we are counting those as additional solutions. We can find an upper bound by making the following observation:

$$\underbrace{\binom{3}{1} \cdot 2}_{\substack{\text{One of the sides is odd} \\ \text{and covers all labels}}} + \underbrace{\binom{3}{2} \cdot 2^2}_{\substack{\text{One of the edges is odd} \\ \text{and covers all labels}}} + \underbrace{1}_{\text{All LSBs are } \perp} = 20$$

Therefore, we can bound the number of solutions of 3D-StrongSperner by a factor of 20. Core breakthorugh came from the utilisation of the hazard-freeness from [8], as well as studying some problems in EOPL and UEOPL or more specifically the One-Permutation Discrete map. Athough in the end we were not able to extract any significantlly useful results out of it, its key observations were enough.

*Useful Corollaries:* We can easily observe that our reduction above can work with any dimensionality. In fact for any $\forall n in \mathbb{N}_{\geq 2}$, Chen et al. showed that nD-StrongSperner is still *PPAD-Hard*. Additionally based on our proof, we can make the following corollary:

**Corollary III.6.1.** *Given* nD-StrongSperner*, we define $f$ as:*

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

*Such that we have the following relationo*

$$\#\text{ND-STRONGSPERNER} \subseteq^f \#\text{PURECIRCUIT}$$

In addition we know that (INSERT CITATION), found a parsimonious reduction from the *EndOfLine* to 3D-STRONGSPERNER, using a slighgly different colour scheme. One can use the same construction that was done on the 2D-STRONGSPERNER to show PPAD-HARDNESS by Chen et al. by using a different but very similar variant to the ENDOFLINE. Lastly we conjecture that we can use the snake embedding technique by Chen Et Al. to bound all the ND-STRONGSPERNER to the same bounds as the 2D-STRONGSPERNER. More formally we conjecture the following to be true

**Conjecture III.7.**

$$f(\cdot) = \binom{2}{1} * 2 + 1$$

$$\forall n \in \mathbb{N}_{\geq 2} : \#\text{ND-STRONGSPERNER} \subseteq^f \#\text{PURECIRCUIT}$$

*2) Hazard-Free Logic Findings:* Whilst trying to tackle the main objectives of the dissertation, we stumbled upon several interest relations across Hazard. We will use the notion of promise problems to show the following relation. First we will define a variant of UnSAT hazard as explained in [8].

**Definition III.2.** *Given a circuit $C$ that computes $f : \mathbb{B}^n \to \mathbb{B}$ such that $\forall x \in \mathbb{B}^n : f(x) = 0$, find $\bar{x} \in \mathbb{T}^n$ such that $C(\bar{x}) = \perp$. We assume that $\forall x \in \mathbb{B}^n : C(x) = 0$.*

Given the idea above we propose the following:

**Proposition III.8.**

$$\#\text{PROMISEUNSATHAZARD} \subseteq \#\text{PURECIRCUIT} - 1$$

For the purposes of the current reduction, we will use the same PURIFY values $(0, 1), (0, \perp), (1, \perp), (\perp, 1)$

*a) Construction:* We start with a single node $s$. We create $n$ copies of PURIFY, where all use $s$ as the input. From each purify gate we use the left output and create a vector $\hat{x} \in \mathbb{T}^n$ and pass them onto $C$ which will output onto a node $o$. We copy the output onto $s$. The above description can be summarised in the following figure 5.

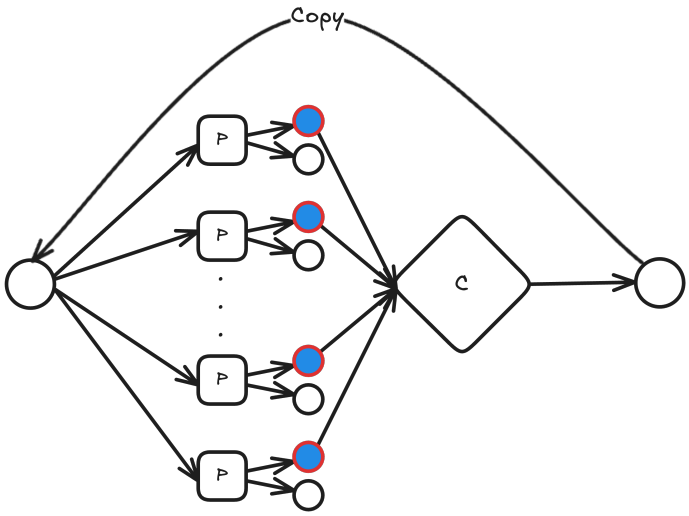



Fig. 5: Pure Circuit construction

*Proof.* To prove our counting argument, we can make the following observation: If $o = 0$, we are computing $C(0^n) = 0$, which is the one guaranteed solution. We know that $o \neq 1$, therefore the only other possible solutions are $\bar{x} \in \mathbb{T}^n : C(\bar{x}) = \perp$. We also know from our construction of our PURIFY gate values, that our highlighted nodes can take values $\{0, 1, \perp\}$. Therefore, given that $\hat{x} \in \mathbb{T}^n$, our construction can find all possible solutions. □

*3) Hazard Circuits and Tarski:* The following finding was derived when looking into alternative fixed point problems. In the following proposition we found a pretty interesting connection between *natural* functions and TARSKI. More specifically, TARSKI is a complexity problem in CLS. First it is essential to define monotone functions:

**Definition III.3** (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:*

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

**Definition III.4** (TARSKI problem definition)**.** *Given a lattice $(L, \wedge, \vee)$, and a monotone function $f : L \to L$, 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 can use the information ordering to impose a partial order on $(\mathbb{T}, \preceq)$ as such:

$$\forall x, y \in \mathbb{T}^n : x \preceq y \implies \forall j \in [n] : x_j \in \mathbb{B} \implies x_j = y_j$$

Therefore we create the following subclass of TARSKI

**Definition III.5** (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**$^\star$ such that*

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

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

**Proposition III.9** (Parsimonious Reduction between KLEENETARSKI and PURECIRCUIT)**.**

$$\#\text{KLEENETARSKI-1} \subseteq \#\text{PURECIRCUIT} - 1$$

The above reduction can be done trivially using the following three steps:

1) Initiate a vector of nodes $s$
2) Pass the inputs into a circuit $C$ that computes the input into an 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.

*4) Reductions to the EndOfLine:* One can also attempt to make some reductions to the ENDOFLINE. It has to be noted, that we assume that there are no parsimonious reductions to the ENDOFLINE, but we were able to construct some variants that allows to make there reductions. They key insights for this comes from the PPAD-inclusion proof by Deligkas et al. [9], where they showed that we can reduce PureCircuit, to *Brouwer* problem by constructing a continuous function $F : [0,1]^n \to [0,1]^n$ as such: For each $v \in V$, we create continuous functions $f_i(\cdot) : [0,1]^n \to [0,1]$ where we take thne input of vector of all the nodes and output the value of the current node

1) If $v \in V$ is the output of a NOR gate with inputs $x_1, x_2$, then we have:

$$f_v(\mathbf{x}) := (1 - \mathbf{x}_{x_1}) \cdot (1 - \mathbf{x}_{x_2})$$

2) If $y_1, y_2$ are the outputs of a PURIFY gate with $z$ as input, then we have:

$$f_{y_1}(\mathbf{x}) := \max\{2 \cdot \mathbf{x}_z - 1, 0\}$$
$$f_{y_2}(\mathbf{x}) := \min\{2 \cdot \mathbf{x}_z, 1\}$$

If $F(x) = x$, then we also found a solution for PURECIRCUIT by:

$$\forall v \in V : \mathbf{x}[v] = \begin{cases} 0 & \mathbf{x}[v] = 0 \\ \perp & 0 < \mathbf{x}[v] < 1 \\ 1 & \mathbf{x}[v] = 1 \end{cases}$$

Given that as an assumption we can restrict our *Purify* gate to outputs $(0, \perp), (0, 1), (\perp, 1)$. Using that we can define two types of variants of PURECIRCUIT that are parsimoniously reducible to SOURCEORPRESINK. We first define SOURCEORPRESINK as:

**Definition III.6** (SOURCEORPRESINK definition)**.** *A* SOURCEORPRESINK *is defined by the same construction as the* ENDOFLINE *but the solutions are sources and predecessors of sinks. If a node is both a source and a presink, then we count it once.*

We will annotate our *Purify* gates with an additional bit $b \in \mathbb{B}$ such that

$$\forall x \in \mathbb{T}, y_1, y_2 \in \mathbb{T} : \text{PURIFY}^b(x) = (y_1, y_2) \implies \text{PURIFY}^{\neg b}(x) = (y_2, y_1)$$

We can create two types of problems based on the annotation:

1) Acyclic PURECIRCUIT: Our instance is acyclic

2) Permutation-free PURECIRCUIT: The outputs of a pure circuit, are connected to a circuit such that any permutation of the inputs retains the result. More formally:

$$\forall x \in \mathbb{T}^n, \sigma \in \mathcal{A} : C(\sigma(x)) = C(x)$$

Where $\mathcal{A}$ is the set of automorphisms of $[n] \to [n]$. The above description can be depicted in the following figure: (ADD FIGURE)

We can show that above variants we can create the following proposition (ref). To prove the reduction we can need to argue that we can go from one solution to another. The core idea of these problems is: if we find a solution given in a set of configurations $\gamma \in \{0,1\}^p$ where $p$ are the number of purify gates, we can match it with an additional solution where our set of configurations is $\neg\gamma$. This can be visualised as such: If the flip does not affect the inputs then we can create the following graph:

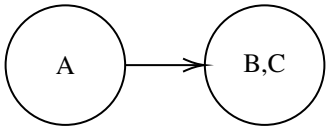



Fig. 6: Equal solution conversion

Otherwise for two distinct solutions we can use the following

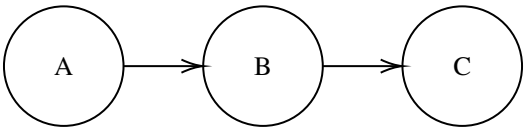



Fig. 7: Not equal solution

*5) Next steps:* Our next steps will focus mainly on two directions. One is lowering the bound as we declare in our conjecture with the usage of snake embeddings. We believe that each dimension can be parsimoniously reduced with another. This implies that, if we can show that $\forall n \in \mathbb{N}_{\geq 3}$ : #ND-STRONGSPERNER $\subseteq$ #2D-STRONGSPERNER we get to prove our conjecture.

With regards to, our second direction, we want to emphasize on the hardness of PURECIRCUIT. Or more specifically we will investigate as to if any of the following statements hold true.

$$\exists n, \alpha \in \mathbb{N}_{\geq 2} : \#\text{SOURCEORXCESS}(\alpha, 1) - 1 \subseteq \#\text{ND-STRONGSPERNER} - 1$$
$$\exists n \in \mathbb{N}_{\geq 2} : \#\text{SOURCEORXCESS}(n, 1) - 1 \subseteq \#\text{PURECIRCUIT} - 1$$

We believe the above two should be our next steps in order to get close to the main question of our project. Ideally generalising, the above reductions to the infinity hierarchy of #SOURCEORXCESS$(\cdot, 1)$ would be ideal as it may lead to finding an overall upper bound towards the entire #PPAD $-$ 1 class.

### *B. Software Progress*

Our project also includes the development of a visualisation tool for creation and verification of several PURECIRCUIT instances. We will use the figure 8 for main reference. 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 achitecture. A general workflow of an *ECS* system can be described as such: a state contains entities, each of which is composed of components 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 9

*1) Current Progress:* In our introduction we referred to three main objectives. As of time of writing we 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 I. Lastly, we provide a figure of our current visualisation in 10

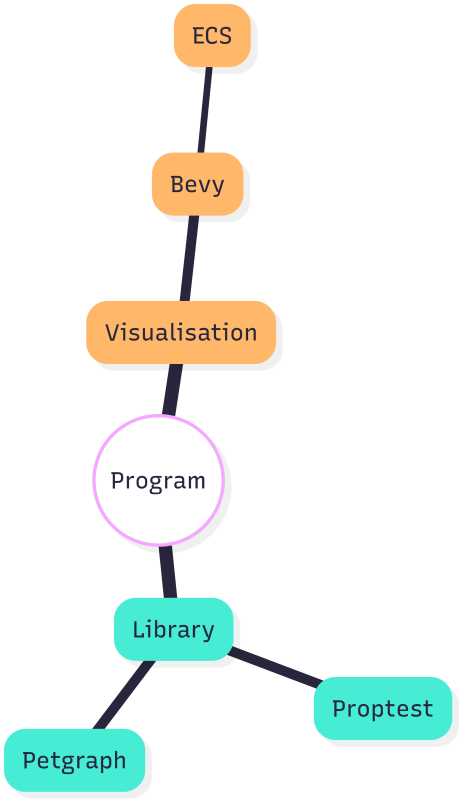



Fig. 8: Application workflow with respect to the libraries and dependencies use

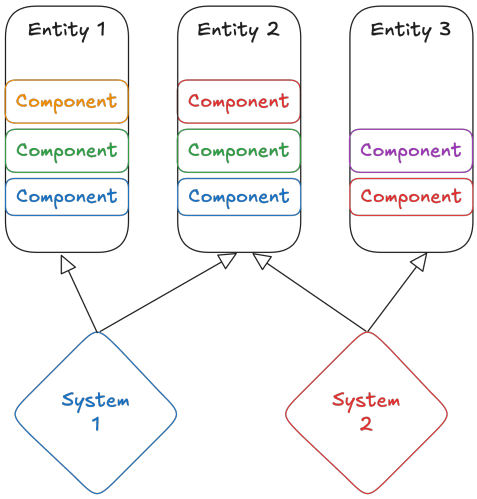
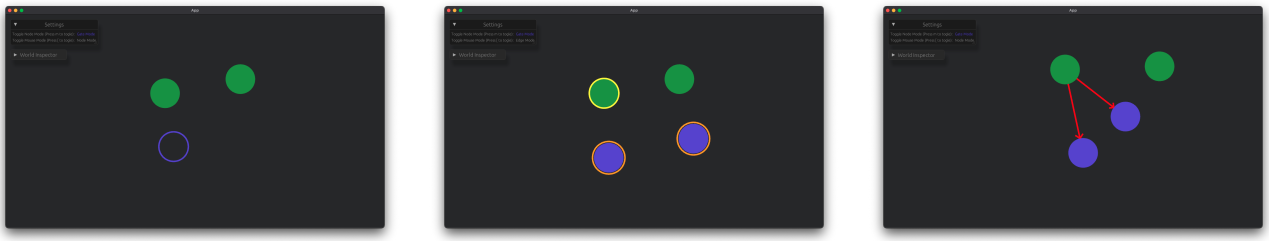



Fig. 9: ECS workflow visualisation

| 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 I: Finished and remaining issues



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

Fig. 10: Screenshots of different states of the visualisation tool

*2) Next steps:* In the next steps we aim to complete the tasks that were mentioned in the above table, as well as finding a solution or even better finding all possible solutions. Of course due to the nature of the problem, finding a single or all solutions for big instances is computationally difficult. We aim to investigate a method introduced by Eichelberger, where he utilised repeated applications of the circuit to detect oscillations and replace them with $\perp$ values. The usage of that procedure was to detect hazard but aim to utilise that procedure as a heuresaaaawtical approach to reach a satisfying assignment. We do not primarily aim to fully optimise these algorithms, as the project focuses more on analysing the theoritical findings of PURECIRCUIT.

## IV. NEXT STEPS

## V. CONCLUSION

## 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] D. Kozen, *Theory of Computation*, ser. Texts in Computer Science. Springer London, 2006.
[4] L. G. Valiant, "The complexity of computing the permanent," *Theoretical Computer Science*, vol. 8, no. 2, pp. 189–201, Jan. 1979.
[5] 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.
[6] C. Ikenmeyer and I. Pak, "What is in #P and what is not?" Apr. 2022.
[7] 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.
[8] 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.
[9] 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.