

Downward self-reducibility in the total function polynomial hierarchy

Karthik Gajulapalli* Surendra Ghentiyala† Zeyong Li‡ Sidhant Saraogi§

July 25, 2025

Abstract

A problem $\mathcal{P}$ is considered *downward self-reducible*, if there exists an efficient algorithm for $\mathcal{P}$ that is allowed to make queries to only strictly smaller instances of $\mathcal{P}$. Downward self-reducibility has been well studied in the case of decision problems, and it is well known that any downward self-reducible problem must lie in PSPACE. Harsha, Mitropolsky and Rosen [ITCS, 2023] initiated the study of downward self reductions in the case of search problems. They showed the following interesting collapse: if a problem is in TFNP and also downward self-reducible, then it must be in PLS. Moreover, if the problem admits a unique solution then it must be in UEOPL.

We demonstrate that this represents just the tip of a much more general phenomenon, which holds for even harder search problems that lie higher up in the total function polynomial hierarchy ($\text{TF}\Sigma_i^P$). In fact, even if we allow our downward self-reduction to be much more powerful, such a collapse will still occur. We show that any problem in $\text{TF}\Sigma_i^P$ which admits a randomized downward self-reduction with access to a Σ_{i-1}^P oracle must be in $\text{PLS}^{\Sigma_{i-1}^P}$. If the problem has *essentially unique solutions* then it lies in $\text{UEOPL}^{\Sigma_{i-1}^P}$.

As an application of our framework, we get new upper bounds for the problems RangeAvoidance and LinearOrderingPrinciple and show that they are both in UEOPL^{NP} , a particularly small subclass of $\text{TF}\Sigma_2^P$. As a corollary of the powerful Range Avoidance framework, we get that a host of explicit construction problems like constructing rigid matrices, Ramsey graphs, hard truth tables against fixed polynomial size circuits are all in UEOPL^{NP} . This appears to be an orthogonal containment to the results by Chen, Hirahara and Ren [STOC 2024], Li [STOC 2024], and Korten and Pitassi [FOCS 2024] that put the LinearOrderingPrinciple and hence RangeAvoidance in FS_2P .

In the third level of the polynomial hierarchy, we show that King, the only candidate problem not known to collapse to any smaller sub-class of $\text{TF}\Sigma_3^P$, indeed collapses to $\text{PLS}^{\Sigma_2^P}$. Even more surprisingly, we give a $\text{ZPP}^{\Sigma_2^P}$ algorithm for King. This refutes the idea proposed in Kleinberg, Korten, Mitropolosky, and Papadimitriou [ITCS 2021] that King is in a class in and by itself.

Along the way, we highlight the power of our framework by giving alternate proofs of PLS and UEOPL membership for several important problems: the P-matrix linear complementarity problem, finding a winning strategy in a parity game, finding a Tarski fixed point, and the

*Georgetown University. Email: kg816@georgetown.edu. Supported by NSF grant CCF-2338730.

†Cornell University. Email: sg974@cornell.edu. This work is supported in part by the NSF under Grants Nos. CCF-2122230 and CCF-2312296, a Packard Foundation Fellowship, and a generous gift from Google.

‡National University of Singapore. Email: li.zeyong@u.nus.edu. Supported by NRF grant NRF-NRFI09-0005.

§Georgetown University. Email: ss456@georgetown.edu.

S – Arrival problem. These proofs only rely on the fact that the natural recursive algorithms for these problems yield downward self-reductions, and therefore classifications into PLS/UEOPL. Consequently, our proofs are shorter, simpler, and take a more algorithmic perspective.

Finally, we highlight the limitations of downward self-reducibility as a framework for PLS membership. We show that unless $\text{FP} = \text{PLS}$, there exists a PLS-complete problem that is not d.s.r. This indicates that traditional d.s.r is not a complete framework for PLS, answering a question left open by Harsha, Mitropolsky and Rosen [ITCS, 2023].

Contents

1	Introduction	1
1.1	Our Results	2
1.1.1	Downward Self Reductions in the Total Function Polynomial Hierarchy . . .	3
1.1.2	New Upper Bounds for AVOID and LINEAR ORDERING PRINCIPLE	4
1.1.3	On the complexity of KING	7
1.1.4	Downward Self Reductions in TFNP	8
1.1.5	The limits of the d.s.r framework	9
1.2	Related Work	9
1.3	Discussion and Open questions	10
2	Preliminaries	11
2.1	The total function hierarchy	11
2.2	TFNP Sub-problems	13
3	Extending Downward self-reducibility	14
3.1	μ -Downward self-reducibility	14
3.2	μ -Downward self-reducibility with essentially unique solutions	18
4	New Upper Bounds for Avoid and the Linear Ordering Principle	19
5	The Complexity of King	21
6	Downward self-reducibility in TFNP	26
6.1	P-LCP	26
6.2	Graph Games	27
6.3	Tarski	29
6.4	S-ARRIVAL	31
7	Limitations of self-reducibility as a framework in TFNP	34
7.1	Not all PLS-complete problems are traditionally d.s.r	34
7.2	A note on random self-reducibility	36
8	Acknowledgements	37

1 Introduction

A problem is considered *downward self-reducible*, if given solutions to smaller sub-problems, one can solve the original problem efficiently. More precisely, a search problem $\mathcal{P}$ is traditionally *downward self-reducible* (d.s.r)¹ if given an instance $x \in \mathcal{P}$, where $|x| = n$, there is a polynomial time reduction to a collection of problems $x_1, \dots, x_t \in \mathcal{P}$ such that for each x_i , we have that $|x_i| < n$, and there is a polynomial time procedure $\mathcal{A}$ that given solutions to all x_i outputs a solution for x .

It turns out that many natural problems are downward self-reducible. In fact downward self-reducibility has found itself as one of the central tools in establishing search-to-decision reductions for a plethora of problems. Perhaps the most well-known example of such a problem is FSAT (functional version of SAT), also known as SEARCH-SAT.

Definition 1. *The problem FSAT takes as input a boolean formula φ over n variables $x_1, \dots, x_n$, and $t = \text{poly}(n)$ clauses. If φ is satisfiable output any satisfying assignment, otherwise, output “UNSAT”*

The downward self-reduction for FSAT proceeds as follows. Let φ_0 denote the sub-formula of φ where x_1 is set to 0 and then all clauses are maximally simplified². Similarly, let φ_1 denote φ where x_1 is set to 1 and then all clauses are maximally simplified. Call the FSAT oracle on φ_0 and φ_1 . If either the oracle call on φ_0 or φ_1 returns a satisfying assignment $x_2 = \alpha_2, \dots, x_n = \alpha_n$, output $(0, \alpha_2, \dots, \alpha_n)$ or $(1, \alpha_2, \dots, \alpha_n)$ respectively. Otherwise, output “UNSAT”. Notice that this reduction is correct since φ is satisfiable if and only if either φ_0 or φ_1 is satisfiable. Furthermore, $|\varphi_0| < t$ and $|\varphi_1| < t$, so the reduction is also downward.

Downward self-reductions similar to the one for FSAT can be shown for a wide variety of different NP-complete problems such as VERTEXCOVER and HAMILTONIAN PATH. In fact, it is known that any problem that is downward self-reducible must be contained in PSPACE. This follows from the existence of the PSPACE algorithm that simply runs the recursive algorithm defined by the downward self-reduction. Moreover, this bound is known to be tight since there exist PSPACE-complete problems which are downward self-reducible (e.g., the TQBF problem).

The recent work of [HMR22] asks about the complexity of a special type of downward self-reducible search problems: TFNP problems. Informally, a search problem is in TFNP (total function nondeterministic polynomial time) if every instance always has a solution (total function) and given a solution it is efficiently verifiable in polynomial time (nondeterministic polynomial time). Restricting TFNP even further, one can define the class TFUP, that include all problems in TFNP that have a unique solution. Perhaps the most useful example to have in mind is the problem FACTOR: given an n bit integer x , output 0^{n-1} if x is prime, otherwise output an $n - 1$ bit integer y such that y divides x . Notice that FACTOR is a total function since every integer x is either prime or has a non-trivial factor of size at most $x/2$. Furthermore, the solutions to FACTOR are efficiently verifiable since checking if x is prime or y divides x can be done in polynomial time. While FACTOR is not in TFUP since an integer may have many factors, it is possible to define a search problem that outputs the full factorization of an n bit input x , which is in TFUP by the unique factorization theorem.

[HMR22] show that any problem that is in TFNP and also downward self-reducible must lie in PLS (Polynomial Local Search), a well-studied TFNP subclass. Moreover, if the problem is in TFUP

¹We often use d.s.r as an abbreviation for both downward self-reduction and downward self-reducible. It should be clear from context which one we mean.

²For example, $(x_1 \wedge x_2) \vee (x_3)$ gets simplified to x_3 since $(x_1 \wedge x_2)$ is false under the assumption that $x_1 = 0$.

and downward self-reducible then it lies in UEOPL (Unique End Of Potential Line), a particularly small TFNP subclass. These results should be somewhat surprising as downward self-reducibility of a problem $\mathcal{P}$ generally only implies that $\mathcal{P} \in \text{PSPACE}$, but adding the minor additional constraint that $\mathcal{P}$ is in TFNP/TFUP (i.e. $\mathcal{P}$ is total and efficiently verifiable) collapses the complexity of $\mathcal{P}$ to relatively small TFNP subclasses PLS/UEOPL. As a consequence [HMR22] gives a barrier for designing recursive algorithms for FACTOR. A recursive algorithm for FACTOR would factor an n bit integer by factoring several integers of bit-length at most $n - 1$, and thus imply a downward self-reduction for FACTOR. This would place FACTOR in UEOPL, resolving long-standing open questions about the complexity of FACTOR within TFNP.

Based on the collapses in TFNP for problems that admit downward self-reductions, it is natural to ask if this is representative of a more general phenomenon. We explore exactly this question in search problems which admit downward self-reductions and lie in TFPH (Total Function Polynomial Hierarchy). In what has now become a seminal paper Kleinberg, Korten, Mitropolsky and Papadimitriou [KKMP21] introduce $\text{TFPH} = \bigcup_{i=1}^{\infty} \text{TF}\Sigma_i^{\text{P}}$, a version of the polynomial hierarchy defined on total search problems. The class $\text{TF}\Sigma_i^{\text{P}}$ defines a total search problem, where given a solution it is verifiable by a Π_{i-1}^{P} machine ($\text{TFU}\Sigma_i^{\text{P}}$ when the problem admits a unique solution). Note that when $i = 1$, this corresponds to TFNP (and resp. TFUP).

In the second level of the polynomial hierarchy $\text{TF}\Sigma_2^{\text{P}}$, [KKMP21] introduce the problem RANGE AVOIDANCE (AVOID) whose study has led to breakthrough results in proving maximum circuit lower bounds, a major open problem in complexity theory [CHR24, Li24]. More generally, [Kor22b] showed that algorithms for AVOID would have many interesting consequences for both derandomization and getting explicit constructions of many combinatorial objects. As one application, [Kor22b] showed an equivalence between the existence of FP^{NP} algorithms for AVOID and the existence of a language in E^{NP} with circuit complexity $2^{\Omega(n)}$. To put this in perspective, the best lower bound currently known for E^{NP} is that it cannot be computed by circuits of size $\sim 3.1n$ [LY22]. However, the best algorithm for AVOID lies in the class FS_2P [CHR24, Li24, KP24] which leaves the following question about the complexity of AVOID open.

Open Problem 1. *Is there a FP^{NP} algorithm for AVOID?*

Going even higher up to the third level of the polynomial hierarchy, [KKMP21] introduce two problems in $\text{TF}\Sigma_3^{\text{P}}$: SHATTERING (the search problem derived from the Sauer-Shelah lemma on the shattering dimension of sets), and KING (the principle that any tournament has a sort of almost-maximal participant). In the case of SHATTERING, they show that the problem collapses to the subclass $\text{PPP}^{\Sigma_2^{\text{P}}}$ ³. However, in the case of KING they posit that unless $\#\text{P}$ is in the polynomial hierarchy, it seems unlikely that KING belongs to $\text{PLS}^{\Sigma_2^{\text{P}}}$. This leads to the natural open question.

Open Problem 2. *Is there a $\text{PLS}^{\Sigma_2^{\text{P}}}$ algorithm for KING?*

1.1 Our Results

In this work, we show that PLS/UEOPL membership theorems of [HMR22] are actually just the tip of the iceberg of a much more general phenomenon. The downward self-reducibility framework for showing PLS membership in fact works for all search problems with *efficiently verifiable solutions* and *promise preserving downward self-reductions*. Thus, we are able to fully recover the results

³PPP is another subclass of TFNP corresponding to the application of the pigeonhole principle.

of [HMR22] as the totality of any $\text{TF}\Sigma_i^P$ problem ensures that all downward self-reductions are promise-preserving⁴, and all solutions are verifiable in P for TFNP problems. We are further able to generalize to randomized reductions, as well as problems which are not strictly downward self-reducible (Definition 3.5). Most interestingly, we show that the downward self-reducibility framework relativizes as we go up the polynomial hierarchy, i.e. a problem $\mathcal{P}$ that is in $\text{TF}\Sigma_i^P$ and also downward self-reducible with access to a Σ_{i-1}^P oracle lies in $\text{PLS}^{\Sigma_{i-1}^P}$.

We then use these ideas to establish surprising containments in $\text{TF}\Sigma_i^P$ subclasses for several total function problems: `AVOID`, `LINEARORDERINGPRINCIPLE`, and `KING`. As a result we make progress on Open Problem 1 and fully resolve Open Problem 2. Along the way, we highlight the power of our framework by giving simple alternate proofs of membership in PLS and UEOPL of some classic TFNP problems.

1.1.1 Downward Self Reductions in the Total Function Polynomial Hierarchy

We first highlight our main technical contribution: a framework (Section 3) for showing collapses in $\text{TF}\Sigma_i^P$ for problems that are downward self-reducible, under very powerful notions of downward self-reducibility. One can interpret our framework as a strong generalization of the main result from [HMR22], which we recall below:

Theorem ([HMR22]). *Every (traditionally) downward self-reducible problem in TFNP is in PLS . Moreover, every downward self-reducible problem in TFUP is in UEOPL .*

While this result is already quite powerful, it is a somewhat restrictive framework. For example, it requires that the downward oracle calls are to instances of strictly smaller “size”. The result holds only for problems in the first level of the polynomial hierarchy, and their framework requires the problems to be total. Moreover, the downward self-reduction must run in deterministic polynomial time. In contrast, we will allow for much more powerful downward self-reducibility and prove the following more general theorem (see Theorem 3.7).

Theorem 1. *Let $\mathcal{R}$ be a search problem in $\text{PromiseF}\Sigma_i^P$ which has a (randomized) μ -d.s.r. If $\mu(x) \leq p(|x|)$ for a polynomial p , then there is a (randomized) reduction from $\mathcal{R}$ to $\text{PLS}^{\Sigma_{i-1}^P}$. Furthermore, if $\mathcal{R}$ has unique solutions, then there is a (randomized) reduction from $\mathcal{R}$ to $\text{UEOPL}^{\Sigma_{i-1}^P}$.*

We now contrast Theorem 1 to the original framework in [HMR22], and describe our relaxations with some observations about our proof that let us generalize the downward self-reducible framework in the following ways.

1. Our theorem applies to $\text{PromiseF}\Sigma_i^P$ problems rather than just TFNP problems. Informally, a $\text{PromiseF}\Sigma_i^P$ is a promise search problem (one where only certain inputs are allowed) which has a Σ_{i-1}^P verifier (see Definition 3.1). We require that our downward self-reduction be promise-preserving⁵, and as a special case of Theorem 1, we get that a downward self-reducible $\text{TF}\Sigma_i^P/\text{TFU}\Sigma_i^P$ problem is in $\text{PLS}^{\Sigma_{i-1}^P}/\text{UEOPL}^{\Sigma_{i-1}^P}$ (see Corollary 3.9). This generalization, allows us to provide a very simple proof that `PROMISE-P-LCP` reduces to UEOPL (see Corollary 6.5).

⁴A total problem satisfies the trivial promise, every input is in the relation.

⁵Our downward self-reduction only maps to instances that also satisfy the promise.

2. The traditional notion of downward self-reducibility encapsulates a set of recursive algorithms, where the recursive calls are made on strictly smaller instances. However, when designing recursive algorithms, the notion of “smaller” is often more flexible than just instance size. To capture this idea, we define μ -d.s.r (see [Definition 3.5](#)). Informally, a problem R is μ -d.s.r if there exists a polynomially bounded function μ defined on instances of R such that solving R on an instance I is polynomial-time reducible to solving R on instance $I_1, \dots, I_t$ such that $\mu(I_i) < \mu(I)$. We believe that the notion of μ -d.s.r is a more accurate abstraction of recursive algorithms since recursive algorithms have some measure on which the instance decreases at each recursive call, but not necessarily the size.

One of the key takeaways from [Theorem 1](#) and this paper as a whole is that *recursive algorithms and inductive proofs of totality are closely related to PLS*, since these correspond to μ -d.s.r, as we will see in [Section 6](#). For a concrete example, consider the problem TARSKI for which we recover an alternate proof of PLS membership. In our reduction, it is crucial for us to have the notion of μ -d.s.r over traditional d.s.r (see [Section 6.3](#)).

3. For problems in $\text{PromiseF}\Sigma_i^P$, we will allow the downward self-reduction to make use of a Σ_{i-1}^P oracle, which makes the reduction very powerful.
4. We further generalize the fact that a μ -d.s.r $\text{TFU}\Sigma_i^P$ problem $\mathcal{R}$ with polynomially bounded μ reduces to $\text{UEOPL}^{\Sigma_{i-1}^P}$. We observe, that when $i \geq 2$, unique solutions are no longer required for this theorem to hold. It suffices that $\mathcal{R}$ has what are dubbed as essentially unique solution [\[KP24\]](#) (see [Definition 3.11](#) and [Lemma 3.14](#)). We use this property, crucially in showing that LINEAR ORDERING PRINCIPLE and AVOID are in UEOPL^{NP} instead of just PLS^{NP} (see [Section 4](#)).
5. Our results apply to randomized reductions.

As a corollary of our framework we show that even though FSAT has a downward self-reduction, PROMISE-FSAT is very unlikely to have a promise-preserving downward self-reduction.

Corollary 1. *If PROMISE-FSAT has a promise-preserving downward self-reduction then $\text{NP} = \text{coNP}$*

This follows from just observing that such a reduction would reduce SAT to PLS, which would imply $\text{NP} = \text{coNP}$. A stronger collapse follows if we consider the problem PROMISEF-UNIQUE SAT (see [Corollary 3.8](#)).

1.1.2 New Upper Bounds for Avoid and Linear Ordering Principle

Equipped with [Theorem 1](#), we make progress towards [Open Problem 1](#). We consider two problems: AVOID and LINEAR ORDERING PRINCIPLE (LOP), and show that they are both in UEOPL^{NP} .

We begin by defining the range avoidance problem (AVOID) introduced in [\[KKMP21\]](#)

Definition 2 (Range Avoidance (AVOID)). *Given a polynomial sized circuit $C : \{0,1\}^n \rightarrow \{0,1\}^{n+1}$, find an element y not in the range of C .*

AVOID is total via the dual pigeonhole principle, and a solution y to AVOID can be verified by the following coNP predicate: $\forall x : C(x) \neq y$. This puts AVOID in $\text{TF}\Sigma_2^P$. Moreover, AVOID has an

abundance of solutions, since at least half the elements in the co-domain are not in the range of C . This abundance guarantees that AVOID is in TFZPP^{NP} , via the trivial algorithm that guesses a random string and checks if it is a valid solution.

AVOID has received a lot of attention in recent literature because of its connection to derandomization [Kor22b, Kor22a] and explicit constructions [Kor22b, RSW22, GLW22, GGNS23, CHLR23, GLV24]. For example [Kor22b] showed that if $\text{AVOID} \in \mathfrak{C}$ for some complexity class $\mathfrak{C}$, then $\text{BPP} \subseteq \mathfrak{C}$. Moreover, the construction of many combinatorial objects known to exist via the probabilistic method like hard truth tables, rigid matrices and Ramsey graphs, just to name a few, all reduce to AVOID. However to get explicit constructions of these combinatorial objects, it is not sufficient to just have an algorithm to AVOID, we also require that our algorithm isolates a single solution for a given instance (we call this problem single-value AVOID). Consider for instance the task of constructing a language that cannot be computed by circuits of size n^2 . If one tries to define a language based on the output of the hard truth table generated from an AVOID algorithm that is not single-valued, then the language would not be well-defined since different hard truth tables will define different languages.

We now show the power of single-value AVOID as a tool for capturing most explicit construction problems via the following folklore theorem:

Theorem ([Kor22b, GLW22]). *If Single-Value AVOID $\in \mathfrak{C}$ for some complexity class $\mathfrak{C}^6$, then we can output the explicit construction of the following objects in $\mathfrak{C}$: Ramsey Graphs, Two Source Extractors, Rigid Matrices, Linear Codes, Hard Truth Tables for any Fixed Polynomial Sized Circuit, K^{poly} -random strings.*

We now state our main result of this section.

Theorem 2. *AVOID is in UEOPL^{NP} .*

In fact, our proof also guarantees uniqueness of the solution. Combined with the theorem on explicit constructions from single-value AVOID we get the following corollary.

Corollary 2. *There is an explicit construction of Ramsey Graphs, Two Source Extractors, Rigid Matrices, Linear Codes, Hard Truth Tables against any Fixed Polynomial Sized Circuit, K^{poly} -random strings in UEOPL^{NP}*

We first give a proof overview of **Theorem 2** (see **Corollary 4.2**), and then compare our results to other upper bounds on AVOID.

It is not clear that AVOID gives us any structure to directly construct a μ -downward self-reduction. So to apply our framework, we consider an intermediate problem called LINEAR ORDERING PRINCIPLE (LOP) introduced in [KP24], where given as input a total order: $\prec$, one must find the smallest element in $\prec$. If it is a total order, LOP has a unique solution, and it is not hard to see that this problem admits essentially unique solutions (**Definition 3.11**).

Definition 3 (Linear Ordering Principle (LOP) [KP24]). *Given $\prec : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$, where $\prec$ is described by a polynomial sized circuit, either:*

1. *Find a witness that $\prec$ does not define a total order i.e. $x, y, z \in \{0, 1\}^n$ such that one of the following holds: (i) $x \prec x$, (ii) $x \neq y$, $x \not\prec y$ and $y \not\prec x$ or (iii) $x \prec y \prec z$ and $x \not\prec z$.*

⁶We only consider classes $\mathfrak{C}$ where $\text{P} \subseteq \mathfrak{C}$.

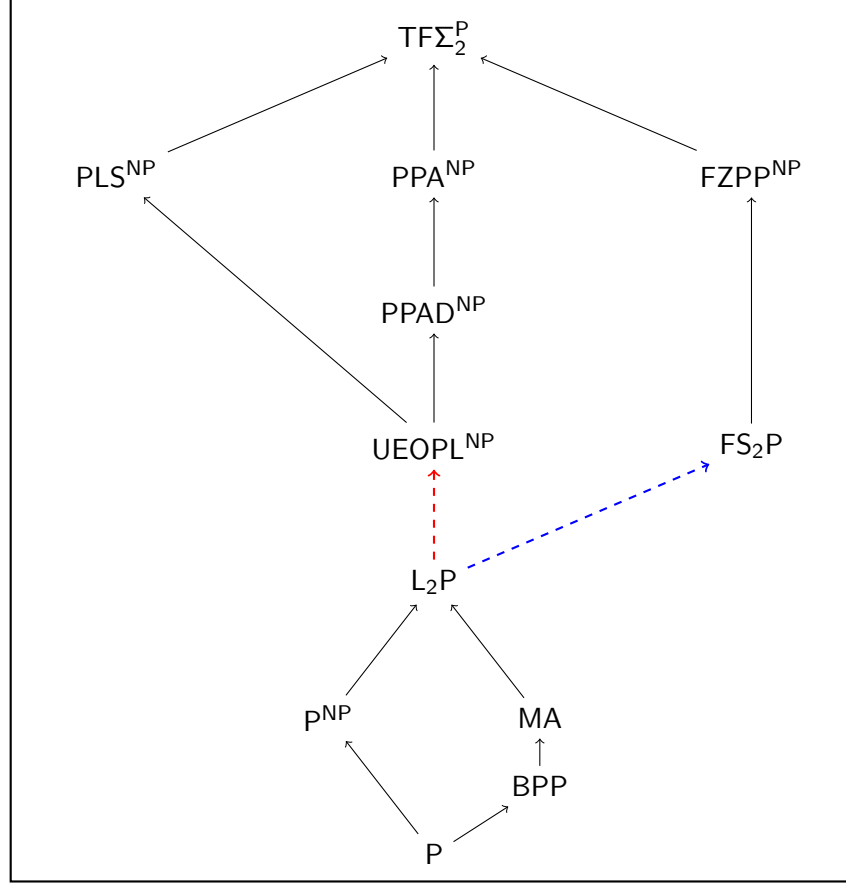


Figure 1: A Fine-Grained $\text{TF}\Sigma_2^P$ Version of the Sipser-Gacs-Lautermann Theorem. Solid arrows denote inclusions and dotted arrows denote decision to search reductions. The inclusion $\text{FS}_2P \subseteq \text{FZPP}^{\text{NP}}$ is due to [Cai07]. L_2P which is a class of decision problems was shown to be equivalent to class of problems that are polynomially many-one reducible to LOP [KP24]. The blue dotted arrow indicates the containment of $\text{LOP} \in \text{FS}_2P$ as shown in [KP24]. The red dotted arrow indicates our new containment of $\text{LOP} \in \text{UEOPL}^{\text{NP}}$.

2. Find the minimal element of the order defined by $\prec$.

Moreover, given a solution y to LOP one can verify that y is minimal by the following coNP predicate: $\forall x : \prec (y, x) = 1$ which puts LOP in $\text{TF}\Sigma_2^P$. We then show that LOP is μ -downward self-reducible (Theorem 4.1), thus by Theorem 3.7 we get the following theorem:

Theorem 3. LOP is in UEOPL^{NP}

Applying the polynomial time many-to-one reduction from AVOID to LOP from [KP24] proves Theorem 2.

We now compare the new containment in Theorem 2 and Theorem 3 to previous results. [CHR24] and [Li24] show that AVOID is in FS_2P . In [KP24] they define L_2P as a new syntactic sub-class of S_2P and show that $\text{MA} \subseteq \text{L}_2P$. Moreover, they show that a language $L \in \text{L}_2P$ is equivalent to L being polynomial time many-one reducible to LOP. Thus, placing LOP in a class of

search problems also implies that L_2P reduces to this class. While [KP24] places LOP in the class FS_2P , our result places both LOP and AVOID in the class $UEOPL^{NP}$. In fact, before Theorem 2 it was open if AVOID or LOP was even in PLS^{NP} or PPA^{NP} . A diagram of our new containment is given in Figure 1.

We now highlight why our new upper bound of LOP and AVOID is interesting.

1. Since AVOID is in $TF\Sigma_2^P$, it is interesting to see what is the smallest standard TFNP sub-class with access to an NP oracle that can solve AVOID. $UEOPL$ is one of the smallest TFNP sub-classes, so getting explicit constructions in $UEOPL^{NP}$ would bring us a step closer to the goal of providing explicit constructions in FP^{NP} . Moreover, a sub-exponential simulation of $UEOPL$ could yield new approaches to getting an E^{NP} construction of rigid matrices and hard truth tables for circuits of size cn for some fixed constant $c > 0$.⁷, both big open problems.
2. The original motivation when defining S_2P by [RS98] and [Can96] was to find the smallest class within the polynomial hierarchy that captures probabilistic classes like BPP and MA. Such a class, inherently captures an abundance of solutions (most random strings are good). And while, AVOID seems to share this property of abundance of solutions, the results of [KP24] indicate that the algorithms for AVOID [CHR24, Li24] isolate a single solution and implicitly solve the more restrictive LOP problem, which has a unique solution. In this regard, it seems that the class $UEOPL^{NP}$ which doesn't correspond to any notion of abundance, and has a unique solution captures the behavior of LOP better than FS_2P .
3. We do not understand how the complexity classes $UEOPL^{NP}$ and FS_2P relate to each other. Under very strong hardness assumptions for derandomization [KVM99] we have that $FS_2P \subseteq FZPP^{NP} \subseteq FP^{NP} \subseteq UEOPL^{NP}$. However, unconditionally they seem incomparable.

1.1.3 On the complexity of King

We now consider the third level of the polynomial hierarchy and look at search problems in $TF\Sigma_3^P$. [KKMP21] illustrated two problems: SHATTERING and KING, that they showed belonged to $TF\Sigma_3^P$. In the case of SHATTERING, they gave a non-trivial application of Σ_2^P oracles to construct a hashing circuit where a collision corresponded to a solution to SHATTERING, thus putting SHATTERING $\in PPP^{\Sigma_2^P}$. However, in the case of KING, no such collapse is known. As a result KING remains the only candidate for a problem in $TF\Sigma_3^P$ that doesn't collapse to a smaller TFNP sub-class relative to a Σ_2^P oracle.

We say that a vertex v in a digraph G is a king if every vertex in G can be reached from v by a path of length at most 2. Moreover, we say that G is a tournament if for every pair of distinct vertices u, v exactly one of (u, v) or (v, u) is present in G . It was shown in [Lan53] that any digraph representing a tournament always has a king. We now define the total problem KING as introduced by [KKMP21]:

Definition 4 (KING). *In the problem KING, given as input a circuit $C : [2^n] \times [2^n] \rightarrow \{0, 1\}$ representing a digraph, either:*

1. *find a distinct $x_1, x_2 \in \{0, 1\}^n$ such that $C(x_1, x_2) = C(x_2, x_1)$,*

⁷Even constructing circuits of size at least $4n$ would be interesting as none of the current techniques seem to be able to beat the $3.2n$ lower bound.

2. find an element $k \in [2^n]$ such that for every $a \in [2^n] \setminus \{k\}$, either $C(k, a) = 1$, or there exists an element $b \in [2^n] \setminus \{k, a\}$, such that $C(k, b) = 1$ and $C(b, a) = 1$.

If one tried to make the proof of totality of KING [Lan53] constructive it seems crucial that one has to count the number of neighbors of a given vertex, thus reducing it to some kind of a generic counting problem. As a result, when discussing the complexity of KING [KKMP21] make the following statement:

“Unless $\#P$ is in the polynomial hierarchy, KING does not appear to belong in $PLS^{\Sigma_2^P}$ ”

Our first result in this section (see Theorem 5.6) seems to completely counter the expectations of [KKMP21].

Theorem 4. KING is in $PLS^{\Sigma_2^P}$

To get this result, we are not able to use Theorem 1 directly. However, we note that our proof directly borrows ideas from the μ -downward self-reducible framework. In Lemma 5.5 we identify a very weak downward self-reducible property relating to tournaments that removes one vertex at a time. While we cannot apply Theorem 1 since our μ here would be exponential, carefully applying ideas from the proof of Theorem 1 allows us to reduce KING to $PLS^{\Sigma_2^P}$ directly.

While Theorem 4 already seems to be counter intuitive, we show perhaps an even more surprising result (see Theorem 5.7).

Theorem 5. KING is in $TFZPP^{\Sigma_2^P}$

To get this result we use the fact that one can sample a coNP predicate uniformly in $ZPP^{\Sigma_2^P}$ [BGP00]. We consider the set W_u that contains all vertices in the graph that witness that u is not a KING. We show (Claim 5.9) that any vertex $v \in W_u$, must have a strictly smaller W_v , moreover for a random $v \in W_u$, W_v will be a constant fraction smaller than W_u with high probability (Claim 5.10). This leads to a natural algorithm that keeps iteratively sampling through W_i till it becomes empty at which point we find a king.

Thus given an oracle Σ_2^P we get the following corollary from Theorem 4 and Theorem 5:

Corollary 3. King is in $PLS^{\Sigma_2^P} \cap TFZPP^{\Sigma_2^P}$

We note that this stands in striking contrast to the fact that KING has no faster randomized algorithm better than 2^n in the black-box query model. Therefore, our result shows that a Σ_2^P oracle likely significantly speeds up algorithms for KING. In fact, even an NP oracle allows us to achieve a significant speedup for KING (to $\text{poly}(n)2^n$ time), which requires $\Omega(2^{4n/3})$ in the black-box query model without a NP oracle (Theorem 5.12).

1.1.4 Downward Self Reductions in TFNP

We apply Theorem 1 to known TFNP problems with the dual purposes of demonstrating how one should use the μ -d.s.r framework and giving alternative proofs of well-known PLS/UEOPL membership results.

We begin with the P-matrix principal linear complementarity (P-LCP) problem. The promise version of P-LCP where the input is guaranteed to be a P-matrix⁸ seems to be of the most

⁸A P-matrix is a square matrix where every principal minor is positive.

interest. [FGMS20] showed that this problem is in UEOPL in two ways. The first proof reduces P-LCP to a different problem UNIQUE-SINK-ORIENTATION, which they also show is in UEOPL. The second proof directly reduces P-LCP to UEOPL but requires the application of Lemke’s algorithm. However, our proof is significantly shorter and simpler (see [Corollary 6.5](#)). It only relies on the observation that P-LCP is downward self-reducible. Interestingly, and quite unusually for proofs of membership in TFNP subclasses, this proof is inherently quite different than the proof that a P-matrix always has a solution to the PROMISE-P-LCP problem.

We then move on to the graph games, which are equivalent to parity games. These types of games were first shown to be in PLS in [BM08] via bounded arithmetic. We are able to give an elementary (not using bounded arithmetic) proof of PLS membership (see [Corollary 6.11](#)).

We next move to the TARKSI problem (see [Corollary 6.13](#)). It was shown in [EPRY20] that TARKSI is in PLS. We give an alternative proof through the μ -d.s.r framework that TARKSI is in PLS. Moreover, our proof is a translation of the classic divide and conquer algorithm for finding Tarski fixed-points, and in this way, more intuitive. The proof seems to rely crucially on the notion of μ -d.s.r as opposed to traditional d.s.r. In this instance, the measure μ captures the size of the lattice that the Tarski instance is defined on. It would be interesting to giving a traditional d.s.r for TARKSI or show that one does not exist.

Finally, we cover the S-ARRIVAL problem. This problem was first shown to be in UEOPL in [FGMS20]. We show that S-ARRIVAL is downward self-reducible. This along with the fact that S-ARRIVAL is in TFUP acts as an alternative proof that S-ARRIVAL is in UEOPL (see [Corollary 6.19](#)). While the original proof of UEOPL membership is quite simple, our proof highlights the recursive nature of the problem.

1.1.5 The limits of the d.s.r framework

Finally, we tackle a problem left open in [HMR22]: is every PLS-complete problem traditionally d.s.r? We give a negative answer to this question (unless $\text{PLS} = \text{FP}$).

Theorem 6. *All PLS-complete problems are traditionally d.s.r if and only if $\text{PLS} = \text{FP}$.*

[Theorem 6](#) follows from a simple padding argument. Though not the focus of this work, we note in [Observation 7.5](#) that a very similar argument shows that all NP-complete/PSPACE-complete problems are traditionally d.s.r if and only if $\text{P} = \text{NP}/\text{P} = \text{PSPACE}$. Interestingly, we note that the above theorem does not provide a barrier to proving that every PLS-complete problem is μ -d.s.r.

Another question asked in [HMR22] is if one can establish a connection between random-self-reducibility and some syntactic TFNP subclass (e.g. PPP) in the same way that downward self-reducibility implies membership in PLS. In [Section 7.2](#), we give some intuition as to why this might be challenging.

1.2 Related Work

The notion of downward self-reducibility has a long history and we refer the interested reader to [All10] as a good starting point.

Downward self-reductions serve as one of the main tools in designing search to decision reductions for a plethora of problems. [HMR22] were the first to consider downward self-reducibility in the context of total problems and observe any interesting consequences. They also showed that

the PLS-complete problem ITER is downward self-reducible, thereby showing that a downward self-reducible TFNP problem is hard if and only if PLS is hard. Along with traditional d.s.r, they also define a more general notion which they refer to as circuit d.s.r. Both these notions are subsumed by μ -d.s.r (Definition 3.5). [HMR22] only used that d.s.r framework to show PLS member for ITER, ITER-WITH-SOURCE, SINK-OF-DAG, and SINK-OF-DAG-WITH-SOURCE, all problems closely related to the canonical PLS-complete problem SINK-OF-DAG. We use our framework to show PLS and UEOPL membership for a wide range of problems. Our framework is more broadly applicable than that of [HMR22]. We are able to use our framework to classify many problems into $\text{PLS}^{\Sigma_i^P}/\text{UEOPL}^{\Sigma_i^P}$, many of which are beyond the scope of [HMR22].

[BCH⁺22] showed another interesting connection between downward self-reducibility and TFNP. They showed that the hardness of any TFNP problem which is d.s.r and has a randomized batching property is sufficient to show the hardness of UEOPL.

[KKMP21] introduced the total function polynomial hierarchy $\text{TF}\Sigma_i^P$. They also introduced the idea of raising a TFNP subclass (like PPP) to a Σ_i^P oracle to create a $\text{TF}\Sigma_{i+1}^P$ subclass (like PPP^{NP}). They showed that the computational problem associated with the Sauer–Shelah lemma, SHATTERING, is in $\text{PPP}^{\Sigma_2^P}$. [KKMP21] also introduced the problem AVOID, which has proven fruitful for understanding several areas of complexity theory [Kor22b, CHR24, Li24, GLV24]. Barriers for designing FP algorithms [ILW23], and FNP algorithms [CL24] for AVOID were shown under cryptographic assumptions. In concurrent work, [HV25] give an upper bound of Ppr^{SBP} for the class L_2P .

Our algorithms for KING are similar in spirit to the work of [JW24, Zha24] where they give exponential time but faster than brute-force algorithms for the TFNP problem PIGEONHOLE EQUAL SUMS. They do this by exploiting the fact that the PIGEONHOLE EQUAL SUMS is total, unlike the related subset sum problem. Our work therefore contributes to the nascent field of designing algorithms for total search problems.

1.3 Discussion and Open questions

We have seen that both LOP and KING are in very small subclasses of $\text{TF}\Sigma_i^P$, i.e. $\text{UEOPL}^{\text{NP}} \cap \text{TFZPP}^{\text{NP}^9}$ and $\text{PLS}^{\Sigma_2^P} \cap \text{TFZPP}^{\Sigma_2^P}$ respectively. There are two ways to interpret our results: we implicitly capture some kind of combinatorial property of the underlying TFNP sub-class via downward self-reductions, or Σ_{i-1}^P oracles are just very powerful.

On one hand, it is possible that the classic TFNP subclasses (e.g, PLS) seem to capture most of the combinatorial principles used to show totality of TFNP problems, moreover the classic TFNP subclasses given access to a Σ_{i-1}^P oracle capture most of the combinatorial principles used to show totality of $\text{TF}\Sigma_i^P$ problems. In some sense, maybe we do not really encounter new combinatorial principles as we go higher in the total function hierarchy.

On the other hand, do we get a UEOPL^{NP} algorithm for AVOID because AVOID is somehow intimately related to some kind of end-of-type arguments, or is it because we believe that AVOID is in FP^{NP} , and we are able to use the NP oracle in some clever way?

We offer no guess as to what the correct answer is, and even formulating this question formally remains an interesting open question.

Finally, we observe that it is not immediate that AVOID even has a downward self-reduction. We are able to get an upper bound on AVOID by showing that AVOID has a polynomial time many-

⁹The containment of LOP in TFZPP^{NP} follows from approximate counting.

one reduction to another problem (LOP) that does have a downward self-reduction. This raises a new approach of showing PLS membership for problems that are not downward self-reducible: try finding an adjacent problem that is downward self-reducible, and to which your original problem reduces to.

Below we list a few open questions that arise from our work.

1. Is there a candidate $\text{TF}\Sigma_3^{\text{P}}$ problem that does not collapse to a sub-class of TFNP with access to a Σ_2^{P} oracle? In the case of $\text{TF}\Sigma_2^{\text{P}}$ the only problems not known to collapse further down are EMPTY [KKMP21] and SHORT-CHOICE [PPY23]. Do these problems also admit further collapses?
2. Can the d.s.r framework be used to show that the $\text{TF}\Sigma_3^{\text{P}}$ problem SHATTERING [KKMP21] is in $\text{PLS}^{\Sigma_2^{\text{P}}}$? In particular, the proof of the Sauer-Shelah lemma, the combinatorial principle behind SHATTERING , employs a divide-and-conquer argument which seems closely related to a downward self-reduction.
3. Does a non-adaptive downward self-reduction for a $\text{TF}\Sigma_i^{\text{P}}$ problem imply membership in a smaller class than $\text{PLS}^{\Sigma_{i-1}^{\text{P}}}$? This is a particularly important question since our downward self-reductions for graph games (Section 6.2), P-LCP (Section 6.1), and LOP (Section 4) are all non-adaptive.
4. Does μ -d.s.r imply traditional d.s.r?
5. Aldous' algorithm [Ald83] gives us a randomized procedure for SINK-OF-DAG for an input $S : \{0, 1\}^n \rightarrow \{0, 1\}^n, V : \{0, 1\}^n \rightarrow \{0, 1\}^n$ that runs in expected time $\text{poly}(n)2^{n/2}$. [FGMS20] and [GHH⁺18] gave $\text{poly}(n)2^{n/2}$ time randomized algorithms for P-LCP and S-ARRIVAL by showing there exist fine grained reductions from those problems to SINK-OF-DAG , and then applying Aldous' algorithm. Our proofs using μ -d.s.r on the other hand incur a large blowup in instance size (except that of KING). Is there any way to apply Aldous' algorithm to achieve a speedup for problems shown to be in $\text{PLS}^{\Sigma_i^{\text{P}}}$ via μ -d.s.r?
6. What conditions can we put on a μ -d.s.r to show membership in TFNP classes between PLS and UEOPL ? For example, how should we extend the μ -d.s.r framework in a way that enables us to show that TARSKI is in CLS ?

2 Preliminaries

For a set $S \subseteq S_1 \times \dots \times S_n$, we let $\pi_i(S)$ denote the projection of S onto its i^{th} coordinate. In particular, $\pi_i(S) = \{e_i : (e_1, \dots, e_i, \dots, e_n) \in S\}$.

2.1 The total function hierarchy

We now formally define a search problem and problems in the total function polynomial hierarchy, i.e. $\text{TF}\Sigma_i^{\text{P}}$.

Definition 2.1. A search problem is a binary relation $\mathcal{R} \subseteq \{0, 1\}^* \times \{0, 1\}^*$ where we say that a y is a solution to x if and only if $(x, y) \in \mathcal{R}$.

Definition 2.2 ([KKMP21]). A relation $\mathcal{R}$ is in $\text{TF}\Sigma_1^P$ if the following conditions hold:

1. $\mathcal{R}$ is total: for all $x \in \{0, 1\}^*$, there exists $y \in \{0, 1\}^*$ such that $(x, y) \in \mathcal{R}$.
2. $\mathcal{R}$ is polynomial: For all $(x, y) \in \mathcal{R}$, $|y| \leq \text{poly}(|x|)$.
3. There exists a polynomial time Turing machine M and polynomials $p_1(n), \dots, p_{i-1}(n)$ such that

$$(x, y) \in \mathcal{R} \iff \forall z_1 \in \{0, 1\}^{p_1(|x|)} \exists z_2 \in \{0, 1\}^{p_2(|x|)} \forall z_3 \in \{0, 1\}^{p_3(|x|)} \dots V(x, y, z_1, \dots, z_{i-1}) \text{ accepts.}$$

Definition 2.3. A relation $\mathcal{R}$ is in $\text{TFU}\Sigma_1^P$ if it is in $\text{TF}\Sigma_1^P$ and for all $x \in \{0, 1\}^*$, there exists exactly one y such that $(x, y) \in \mathcal{R}$.

$\text{TF}\Sigma_1^P$ and $\text{TFU}\Sigma_1^P$ are also referred to as TFNP and TFUP respectively. Informally, TFNP is the set of search problems where a solution always exists and is efficiently verifiable (is in FNP , the search analogue of NP). $\text{TF}\Sigma_i^P$ is a generalization of this notion higher in the polynomial hierarchy, where the verifier need not be polynomial time.

With the notion of search problem pinned down, one can begin to talk about reductions between search problems. The norm when reducing between search problems is to restrict oneself to many-to-one reductions. The following definition formalizes the intuitive idea that a reduction from $\mathcal{R}$ to $\mathcal{Q}$ should take as input an instance x of $\mathcal{R}$, transform it to an instance $f(x)$ of $\mathcal{Q}$, get back an answer y to that instance, and transform that to $g(y)$, an answer for instance x of $\mathcal{R}$.

Definition 2.4. Let $\mathcal{R}, \mathcal{Q}$ be two search problems. A many-to-one reduction from $\mathcal{R}$ to $\mathcal{Q}$ is defined as two polynomial time computable functions f, g such that for all $x \in \pi_1(\mathcal{R}), y \in \{0, 1\}^*$, the following holds.

$$(x, g(y)) \in \mathcal{R} \iff (f(x), y) \in \mathcal{Q}$$

We note that [Definition 2.4](#) is slightly different from the notion of a reduction between TFNP problems since we only quantify over all $x \in \pi_1(\mathcal{R})$ rather than all $x \in \{0, 1\}^*$. We do this since we will often work with problems where the inputs to the search problem are restricted and there exists some x such that $x \notin \pi_1(\mathcal{R})$. One should think of these relations as promise problems. Occasionally, it will be helpful to make these problems total in a trivial way, so we define the following.

Definition 2.5. Let $\mathcal{R}$ be a relation and let $A = \{(x, \perp) : x \notin \pi_1(\mathcal{R})\}$. We define the completion of $\mathcal{R}$ as $\overline{\mathcal{R}} := \mathcal{R} \cup A$.

We now define what it means for an (oracle) algorithm to solve a search problem $\mathcal{R}$ in polynomial time.

Definition 2.6. We say $\mathcal{R}$ has a polynomial time algorithm $\mathcal{A}$ if $\mathcal{A}$ runs in polynomial time for all inputs in $\{0, 1\}^*$ and for all $x \in \pi_1(\mathcal{R})$, $(x, \mathcal{A}(x)) \in \mathcal{R}$.

Definition 2.7. Let $\mathcal{Q}$ be a total search problem. We say $\mathcal{R}$ has a polynomial time algorithm $\mathcal{A}^{\mathcal{Q}}$ if $\mathcal{A}^{\mathcal{Q}}$ runs in polynomial time and for all $x \in \pi_1(\mathcal{R})$ and all instantiations of the oracle $\mathcal{Q}$, $(x, \mathcal{A}^{\mathcal{Q}}(x)) \in \mathcal{R}$.

2.2 TFNP Sub-problems

Finally, we will make extensive use of the following TFNP subclasses. For any TFNP subclass $\mathcal{C}$ has a circuit as part of its input, we define $\mathcal{C}^{\Sigma_i^P}$ as the same problem as $\mathcal{C}$ except that the input circuit to $\mathcal{C}$ may include Σ_i^P oracle gates. It is not hard to confirm that $\mathcal{C}^{\Sigma_i^P} \subseteq \text{TF}\Sigma_{i+1}^P$.

Definition 2.8 (PLS and SINK-OF-DAG). PLS is defined as all problems which are many-to-one reducible to SINK-OF-DAG, which is defined as follows. Given $\text{poly}(n)$ size circuits $S : [2^n] \rightarrow [2^n]$ (successor circuit) and $V : [2^n] \rightarrow [2^n]$ (value circuit), find v such that $S(v) \neq v$ and either $S(S(v)) = S(v)$ or $V(S(v)) \leq V(v)$.

One should think of SINK-OF-DAG as a solution to a gradient ascent problem. There are two circuits, a successor circuit and a value circuit. At every point v in the space, we hope that the successor function S leads to a point which has a higher value ($V(S(v)) > V(v)$). A solution to SINK-OF-DAG is a point such that this condition is violated $V(S(v)) \leq V(v)$, or one which acts as (a predecessor of) a sink in the gradient ascent process, where $S(v) \neq v$ but $S(S(v)) = S(v)$.

We will also make use of the following promise problem. SINK-OF-VERIFIABLE-LINE is very similar to SINK-OF-DAG except that we also require a verifier which can tell us if a vertex x is the i^{th} vertex visited by repeatedly applying the successor function. Note that SINK-OF-VERIFIABLE-LINE is a promise problem since we have no way to confirm if the verifier meets this condition.

Definition 2.9 (SINK-OF-VERIFIABLE-LINE). Given a successor circuit $S : \{0, 1\}^n \rightarrow \{0, 1\}^n$, a source $s \in \{0, 1\}^n$, a target index $T \in [2^n]$, and a verifier circuit $W : \{0, 1\}^n \times [T] \rightarrow \{0, 1\}$ with the guarantee that for $(x, i) \in \{0, 1\}^n \times [T]$, $W(x, i) = 1$ if and only if $x = S^{i-1}(s)$, find the string $v \in \{0, 1\}^n$ such that $W(v, T) = 1$.

Finally, although we will not make use of the complete problem for the UEOPL, we present it here for the sake of completeness. See [FGMS20] for an explanation of why the following is a reasonable definition.

Definition 2.10 (UNIQUE-END-OF-POTENTIAL-LINE). UNIQUE-END-OF-POTENTIAL-LINE is defined as follows. Given two $\text{poly}(n)$ size circuits $S, P : \{0, 1\}^n \rightarrow \{0, 1\}^n$, such that $P(0^n) = 0^n \neq S(0^n)$, and a value circuit $V : \{0, 1\}^n \rightarrow \{0, 1, \dots, 2^{m-1}\}$ where $V(0^n) = 0$. Find one of the following:

- A point $x \in \{0, 1\}^n$ such that $P(S(x)) \neq x$.
- A point $x \in \{0, 1\}^n$ such that $x \neq S(x)$, $P(S(x)) = x$ and $V(S(x)) - V(x) \leq 0$.
- A point $x \in \{0, 1\}^n$ such that $S(P(x)) \neq x \neq 0^n$.
- Two points $x, y \in \{0, 1\}^n$, such that $x \neq y$, $x \neq S(x)$, $y \neq S(y)$, and either $V(x) = V(y)$ or $V(x) < V(y) < S(x)$.

UEOPL is defined as all problems which are many-to-one reducible to UNIQUE-END-OF-POTENTIAL-LINE.

Lemma 2.11 ([FGMS20]). SINK-OF-VERIFIABLE-LINE *reduces to* UNIQUE-END-OF-POTENTIAL-LINE *under relativizing reductions*.

3 Extending Downward self-reducibility

3.1 μ -Downward self-reducibility

We begin by defining the types of problems for which we hope to show downward self-reducibility: Efficiently verifiable promise problems.

Definition 3.1. We say that a search problem $\mathcal{R} \subseteq \{0, 1\}^* \times \{0, 1\}^*$ is in the class $\text{PromiseF}\Sigma_i^P$ if it satisfies the following conditions.

1. For all $x \in \pi_1(\mathcal{R})$, and for all y s.t. $(x, y) \in \mathcal{R}$, $|y| \leq \text{poly}(|x|)$.
2. There exists a polynomial time verifier V and polynomials $p_1(n), \dots, p_{i-1}(n)$ such that the following holds: for all $x \in \pi_1(\mathcal{R})$,

$$(x, y) \in \mathcal{R} \iff \forall z_1 \in \{0, 1\}^{p_1(|x|)} \exists z_2 \in \{0, 1\}^{p_2(|x|)} \forall z_3 \in \{0, 1\}^{p_3(|x|)} \dots V(x, y, z_1, \dots, z_{i-1}) \text{ accepts.}$$

Definition 3.2. We say a search problem $\mathcal{R} \subseteq \{0, 1\}^* \times \{0, 1\}^*$ has unique solutions if for all $x \in \{0, 1\}^*$, there exists at most one y such that $(x, y) \in \mathcal{R}$.

Perhaps the most helpful way to think of $\text{PromiseF}\Sigma_i^P$ is as a version of $\text{TF}\Sigma_i^P$ where some input instances are disallowed. In particular, we are only concerned with the instances $\pi_1(\mathcal{R})$ ¹⁰. For a problem $\mathcal{R}$, we will often say that x meets the promise of $\text{TF}\Sigma_i^P$ if $x \in \pi_1(\mathcal{R})$. Some examples of $\text{PromiseF}\Sigma_i^P$ problems to have in mind are PROMISE-FSAT and PROMISE-UNIQUE-FSAT.

Definition 3.3. PROMISE-FSAT: $(\varphi, y) \in \text{PROMISE-FSAT}$ if φ is a satisfiable boolean formula, and y is a satisfying assignment for φ .

Definition 3.4. PROMISE-UNIQUE-FSAT: $(\varphi, y) \in \text{PROMISE-UNIQUE-FSAT}$ if φ is a satisfiable boolean formula with a unique satisfying assignment, and y corresponds to the unique satisfying assignment for φ .

We now define μ -downward self-reducibility. Intuitively, a problem $\mathcal{R}$ is μ -d.s.r if there exists some function μ over instances of $\mathcal{R}$ such that there exists a self-reduction for $\mathcal{R}$ which calls its $\mathcal{R}$ oracle on instances which have strictly smaller values for μ . A traditionally d.s.r problem is μ -d.s.r for $\mu(x) = |x|$. In some sense, $\mathcal{R}$ being μ -d.s.r means that there is a recursive algorithm where μ decreases at each recursive call. In this way, we believe μ -d.s.r more fully captures recursive algorithms over traditional d.s.r.

Definition 3.5. Let $\mu : \{0, 1\}^* \rightarrow \mathbb{N}$ be a function. A $\text{PromiseF}\Sigma_i^P$ problem $\mathcal{R}$ is (randomized) Σ_{i-1}^P - μ -downward-self-reducible (μ -d.s.r.) if there is a (randomized) polynomial-time oracle algorithm $\mathcal{A}^{\Sigma_{i-1}^P, \mathcal{R}}$ for $\mathcal{R}$ that on input $x \in \pi_1(\mathcal{R})$, makes queries to a $\mathcal{R}$ -oracle on instances y such that y satisfies the promise of $\mathcal{R}$, $|y| \leq |x| + \text{poly}(\mu(x))$, and $\mu(y) < \mu(x)$. We simply say that such a problem is μ -downward-self-reducible when Σ_{i-1}^P is clear from context.

Remark 3.6. We make a few comments about the definition above.

¹⁰ $\pi_1(\mathcal{R})$ is the set of instances that have a solution, for example, in the case of FSAT: $\pi_1(\mathcal{R})$ is the set of satisfiable formulas.

- It is implicit in the [Definition 3.5](#) that for any instance x where $\mu(x) = 0$, the reduction algorithm $\mathcal{A}^{\Sigma_{i-1}^P, \overline{\mathcal{R}}}$ straightaway solves x since it cannot make any query of smaller μ .
- To be fully formal, [Definition 3.5](#) specifies that $\mathcal{R}$ is Σ_{i-1}^P - μ -downward-self-reducible rather than just saying it is μ -downward-self-reducible to eliminate trivial cases like the following. PROMISE-FSAT is a $\text{TF}\Sigma_2^P$ problem since it is a FNP problem. Furthermore, PROMISE-FSAT is Σ_1^P - μ -d.s.r since one can use the Σ_1^P oracle to find a solution. So in some sense, PROMISE-FSAT is μ -d.s.r. However, this clearly misses the point that we wish for the d.s.r to have access to less computational power than an oracle which simply solves the problem in question. We therefore parameterize the d.s.r by the oracle it is allowed access to (Σ_{i-1}^P - μ -downward-self-reducible). However, for the rest of this work, it will be clear what Σ_{i-1}^P should be and we therefore omit it and simply write μ -d.s.r to avoid unnecessary notation.
- In the definition of μ -d.s.r, if x meets the promise of $\mathcal{R}$, then all the queries which $\mathcal{A}^{\Sigma_{i-1}^P, \overline{\mathcal{R}}}$ ¹¹ makes to its $\mathcal{R}$ oracle, call them y_i , must satisfy $y_i \in \pi_1(R)$. In other words, if x satisfies the promise of $\mathcal{R}$, the μ -d.s.r for $\mathcal{R}$ can only make queries to its oracle which satisfy the promise of $\mathcal{R}$. Notice also that by our definition of μ -d.s.r, we do not have to worry about the behavior of the oracle for $\mathcal{A}$ on inputs which do not satisfy the promise of $\mathcal{R}$. One should imagine these inputs as causing the oracle to output $\perp$. However we wish to emphasize that this is just for definitional convenience and that $\mathcal{A}$ can never use such information, since by definition, it never queries its oracle on such an input.

For intuition, let us consider what a μ -d.s.r for PROMISE-FSAT would look like. This would be an algorithm which on a satisfiable SAT instance x would query *only* satisfiable SAT instances $y_1, \dots, y_{\text{poly}(n)}$ such that each y_i is smaller than x (under μ), receives back satisfying assignments $a_1, \dots, a_{\text{poly}(n)}$, and then uses those to construct a satisfying assignment for x .

We now prove one of our main theorems relating μ -d.s.r to PLS/UEOPL. The proof uses essentially the same strategy as [\[HMR22\]](#) of creating a SINK-OF-DAG instance where nodes represent possible stack traces of the natural recursive algorithm for solving our μ -d.s.r problem.

Theorem 3.7. *Let $\mathcal{R}$ be a search problem in $\text{PromiseF}\Sigma_i^P$ which has a (randomized) μ -d.s.r. If $\mu(x) \leq p(|x|)$ for a polynomial p , then there is a (randomized) reduction from $\mathcal{R}$ to $\text{PLS}^{\Sigma_{i-1}^P}$. Furthermore, if $\mathcal{R}$ has unique solutions, then there is a (randomized) reduction from $\mathcal{R}$ to $\text{UEOPL}^{\Sigma_{i-1}^P}$.*

Proof. We start by proving the simple case where the μ -d.s.r. algorithm is deterministic and $i = 1$, i.e. $\mathcal{R} \in \text{PromiseF}\Sigma_1^P$. If the following reduction breaks at any point or runs for too long, the reduction simply outputs $\perp$. This will be fine since we will show that the reduction works whenever x satisfies the promise of $\mathcal{R}$.

Let A' be the polynomial-time oracle algorithm for $\mathcal{R}$. Without loss of generality, let us assume that on input $x \in \{0, 1\}^n$, A' makes at most $q(n)$ queries, each of size bounded by $n + \ell(\mu(x))$ for polynomials q and ℓ . Let A be an (exponential-time) depth-first recursive algorithm that simulates A' and recursively calls A' whenever A' makes an oracle query. Due to the strictly decreasing property of $\mu(x)$ in the μ -d.s.r. definition, the maximum depth of recursion is $\mu(x) \leq p(n)$ and all instances have size bounded by $s := n + \mu(x)\ell(\mu(x))$. A' on input x computes depth $d := \mu(x)$ and width $w := q(s)$. Without loss of generality by introducing dummy oracle queries, we enforce A' to

¹¹ $\overline{\mathcal{R}}$ is the extension of $\mathcal{R}$ that makes it total (see [Definition 2.5](#)).

make *exactly* w oracle queries at each simulation of A' and have recursive depth *exactly* d . In other words, the computational graph of A would be a perfect w -ary tree of height d . For simplicity, let us further assume that the solutions have the same size as the instances, which can be done by padding $\mathcal{R}$ in the first place.

On input $x \in \{0, 1\}^n$, we will construct a SINK-OF-DAG instance as follows. Each vertex is represented by a table $t[\cdot, \cdot]$ of $d + 1$ rows and w columns. Each entry $t[i, j] \in \{0, 1\}^s \times \{0, 1\}^s$ consists of an instance-solution pair and may take one of the following forms:

1. $(\xi, \perp)$ of an $\mathcal{R}$ -instance ξ and $\perp$ indicating that its solution is yet to be found.
2. (ξ, γ) of an $\mathcal{R}$ -instance ξ and its purported solution γ .
3. $(\perp, \perp)$ indicating that there is no instance.

Intuitively, $t[\cdot, \cdot]$ is the stack content of the recursive algorithm A . On input x , the source vertex t_0 is defined by setting $t_0[0, 1] = (x, \perp)$ and $t_0[i, j] = (\perp, \perp)$ for all other i, j .

Validity A vertex t is valid if and only if it passes the following validity test: Let i^* be the smallest index such that the row $t[i^*, \cdot]$ consists of only $(\perp, \perp)$. For $i \in [p(n)]$, let j_i^* be the smallest index such that $t[i, j_i^*]$ takes the form of $(\xi, \perp)$.

- All rows $t[i, \cdot]$ for $i \geq i^*$ consist of only $(\perp, \perp)$. For any $i \in [p(n)]$ and $j \geq j_i^*$, $t[i, j] = (\perp, \perp)$.
- For any $t[i, j]$ taking the form (ξ, γ) , γ is a valid solution for ξ . We check this using the verifier for $\mathcal{R}$.
- For any $t[i, j] \neq (\perp, \perp)$, consider simulating A' on the input instance stored in $t[i - 1, j_{i-1}^*]$, and verify that the first $j - 1$ query-solution pairs are in $t[i, 1], \dots, t[i, j - 1]$, and $t[i, j]$ is the j -th query made by A' .

Successor On vertex t , the successor circuit S behaves as follows.

- If t is invalid, then $S(t) = t$. These are isolated vertices.
- If $t[0, 1] = (x, y)$ and $y \neq \perp$ is a valid solution to x , then $S(t) = t$.
- Otherwise, we construct successor vertex t' as follows. Let i be the largest index such that the i -th row $t[i, \cdot]$ consists of an entry of the form $(\xi, \perp)$. Suppose that the $(i + 1)$ -th row $t[i + 1, \cdot]$ has j query-solution pairs. If A' on input ξ and these j query-solution pairs makes a $(j + 1)$ -th query ξ' , then we set $t[i + 1, j + 1] := (\xi', \perp)$. Otherwise A' would have found a solution γ for ξ . In this case, we replace $(\xi, \perp)$ by (ξ, γ) and set $t[i + 1, \cdot]$ to all $(\perp, \perp)$.

Potential For invalid t , $V(t) = 0$. For valid vertex t , the potential circuit V returns the *exact* number of steps taken for the depth-first recursive algorithm A to reach the state t . This can be easily computed as follows: Go through the table t , for each $t[i, j] = (\xi, \gamma)$, add $\sum_{k=i}^d 2w^{d-k}$; for each $t[i, j] = (\xi, \perp)$, add 1. We provide a quick verification that for a non-sink vertex t , $V(S(t)) = V(t) + 1$:

- If $S(t)$ writes an entry from $(\perp, \perp)$ to $(\xi, \perp)$, the potential increases by exactly 1.

- If $S(t)$ writes an entry $t[i, j]$ from $(\xi, \perp)$ to (ξ, γ) for $i = d$ (i.e. at the leaf level), the potential changes by $2 - 1 = 1$.
- If $S(t)$ writes an entry $t[i, j]$ from $(\xi, \perp)$ to (ξ, γ) for $i < d$ and erase the row $t[i + 1, \cdot]$ to $(\perp, \perp)$, by assumption all w entries in $t[t + 1, \cdot]$ were of the form (ξ', γ') . Hence the potential changes by $\sum_{k=i}^d 2w^{d-k} - w \cdot \sum_{k=i+1}^d 2w^{d-k} - 1 = 1$.

Correctness follows from that the only non-isolated vertices without a proper successor correspond to the final configurations of A with the solution to x written in $t[0, 1]$.

Randomized Reduction If the μ -d.s.r. algorithm A' is randomized, without loss of generality assume that A' uses $r(n)$ bits of randomness and succeeds with probability at least $1 - 1/\nu(n)$ for polynomials $r(\cdot)$ and $\nu(\cdot)$.

Since $\mathcal{R} \in \text{PromiseF}\Sigma_1^P$, we could verify the validity of the solution. Hence, we start by amplifying the success probability to $1 - \frac{1}{\nu(n)^{n+s}}$ by repeating A' $(n + s)$ times in parallel with $(n + s)r(n)$ bits of randomness. By union bound over at most 2^s instances of size s , the success probability remains exponentially high.

The randomized reduction to PLS is as follows. Sample $(n + s)r(n)$ bits of randomness and hardcode them into the PLS instance. With the randomness fixed, we could apply the reduction from above to obtain the PLS instance.

Relativization The proof above fully relativizes in the following sense: the *only* property of $\text{PromiseF}\Sigma_1^P$ used is that given an instance-solution pair (ξ, γ) , one could verify if γ is a solution for ξ in polynomial time. Now if $\mathcal{R}$ is a $\text{PromiseF}\Sigma_i^P$ problem, one could use a Σ_{i-1}^P oracle to verify the validity of a purported solution. Moreover, the d.s.r. algorithm A' (with Σ_{i-1}^P oracle) can be simulated by a circuit with Σ_{i-1}^P oracle gates.

Unique Solutions We reduce to SINK-OF-VERIFIABLE-LINE and the appeal to the fact that SINK-OF-VERIFIABLE-LINE reduces to UEOPL (Lemma 2.11). The reduction to SINK-OF-VERIFIABLE-LINE uses the exact same S as previously but creates W as $W(t, i) = 1$ if and only if t is valid and $V(t) = i$, sets $T = \sum_{k=0}^d 2w^{d-k}$ and then calls SINK-OF-VERIFIABLE-LINE to get an answer t' . The reduction then outputs the solution stored in $t'[0, 1]$. We see that W is a valid verifier for SINK-OF-VERIFIABLE-LINE. If $S^{i-1}(s) = t$, then $W(t, i) = 1$ because t must be valid and $V(t) = i$ since V increased by exactly one at each application of S . If $W(t, i) = 1$, then $S^{i-1}(s) = t$. This follows from the fact that since $\mathcal{R}$ has unique solutions, for every i , there exists exactly one valid table t such that $V(t) = i$. Therefore, W satisfies the promise required by SINK-OF-VERIFIABLE-LINE. $S^{T-1}(s) = t'$ is clearly the $(T - 1)^{\text{th}}$ state of the recursive algorithm A and therefore clearly contains the solution for x , which the reduction correctly outputs. $\square$

Corollary 3.8. *If PROMISEFSAT is μ -d.s.r for polynomially bounded μ , then SAT reduces to PLS. If PROMISE-UNIQUE-FSAT is μ -d.s.r for polynomially bounded μ , then SAT reduces to UEOPL.*

Proof. Say PROMISEFSAT were μ -d.s.r for polynomially bounded μ . That would imply a reduction (f, g) from PROMISEFSAT to PLS by Theorem 3.7. We can turn this into a reduction from SAT to PLS as follows. For an input x to SAT, the reduction computes $f(x)$ (notice that by definition,

f must terminate in polynomial time and output something even if x does not meet the promise of PROMISEFSAT) and feeds $f(x)$ to the PLS oracle to get back y . The reduction then computes $g(y)$ (notice that by definition, g must terminate in polynomial time and output something). If $g(y)$ is a satisfying assignment for x , the reduction outputs SAT and otherwise outputs UNSAT. Correctness of the reduction follows from the correctness of the reduction f, g on satisfiable instances. Essentially the same proof holds for PROMISE-UNIQUE-FSAT, but PLS is replaced by UEOP. $\square$

Corollary 3.9. *Let $\mu : \{0, 1\}^* \rightarrow \mathbb{N}$ be a function bounded by some polynomial. Any $\text{TF}\Sigma_i^P$ problem which is μ -d.s.r with a Σ_{i-1}^P oracle is in $\text{PLS}^{\Sigma_{i-1}^P}$. Any $\text{TFU}\Sigma_i^P$ problem which is μ -d.s.r with a Σ_{i-1}^P oracle is in $\text{UEOP}^{\Sigma_{i-1}^P}$.*

Proof. Notice that any μ -d.s.r for a $A \in \text{TF}\Sigma_i^P$ problem is promise-preserving since A is total. This lets us apply [Theorem 3.7](#) to achieve the desired result. $\square$

3.2 μ -Downward self-reducibility with essentially unique solutions

[Corollary 3.9](#) tells us that if a $\text{TF}\Sigma_i^P$ problem has unique solutions and is μ -d.s.r for polynomially bounded μ , then it is in $\text{UEOP}^{\Sigma_{i-1}^P}$. However, [\[KP24\]](#) observed that for $\text{TF}\Sigma_2^P$, unique solutions are sometimes not the correct notion to work with. A good example to keep in mind when thinking about essentially unique solutions is the problem EMPTY.

Definition 3.10. The problem EMPTY is defined as follows. Given a $\text{poly}(n)$ size circuit $C : [2^n - 1] \rightarrow [2^n]$, output y such that $y \notin \text{range}(C)$.

Although EMPTY technically does not have unique solutions (consider $C(x) = 0$), it seems rather close. In particular, it is easy to verify with a NP oracle if we are in the case when EMPTY has a unique solution (C is injective). [\[KP24\]](#) define the notion of essentially unique solutions to capture these types of problems. The following is a generalization of the notion of essentially unique solutions from $\text{TF}\Sigma_2^P$ to $\text{TF}\Sigma_i^P$ for any $i \geq 2$.

Definition 3.11 (Essentially Unique Solutions [\[KP24\]](#)). We say that a total search problem $\mathcal{R} \in \text{TF}\Sigma_i^P$ ($i \geq 2$) has essentially unique solutions if there exist two verifiers V_1, V_2 such that the following hold.

1. V_1 is testable in polynomial time with an Σ_{i-2}^P oracle and V_2 is testable in polynomial time with an Σ_{i-1}^P oracle.
2. For all x , either there exists a y such that $V_1(x, y) = 1$ and $(x, y) \in \mathcal{R}$, or else there exists a unique y such that $V_2(x, y) = 1$ and $(x, y) \in \mathcal{R}$.

We will show that if a $\text{TF}\Sigma_i^P$ problem has essentially unique solutions and is μ -d.s.r for polynomially bounded μ , then it has a (randomized) reduction to $\text{UEOP}^{\Sigma_{i-1}^P}$. To do so, for any search problem $\mathcal{R}$ that has essentially unique solutions, we define $\mathcal{R}_u$, a version of $\mathcal{R}$ that has unique solutions.

Definition 3.12. For any relation $\mathcal{R}$ with essentially unique solutions and verifiers V_1, V_2 , we define $\mathcal{R}_u$ as follows.

1. For any $x \in \pi_1(\mathcal{R})$ such that there exists y where $V_1(x, y) = 1$, $(x, y) \in \mathcal{R}_u$ if and only if y is the lexicographically smallest y where $V_1(x, y) = 1$.
2. For any $x \in \pi_1(\mathcal{R})$ such that $V_1(x, y) = 0$ for all y , $(x, y) \in \mathcal{R}_u$ if and only if $(x, y) \in \mathcal{R}$.

Note that if $\mathcal{R} \in \text{TF}\Sigma_i^P$, then $\mathcal{R}_u \in \text{TF}\Sigma_i^P$. We now show that $\mathcal{R}$ reduces to $\mathcal{R}_u$, and that if $\mathcal{R}$ is μ -d.s.r, then $\mathcal{R}_u$ is also μ -d.s.r. The combination of these two facts will be sufficient to show $\text{UEOPL}^{\Sigma_{i-1}^P}$ membership of μ -d.s.r $\text{TF}\Sigma_i^P$ problems with essentially unique solutions (assuming μ is polynomially bounded of course).

Lemma 3.13. *Let $\mathcal{R} \in \text{TF}\Sigma_i^P$ be any relation with essentially unique solutions. Then, there is a reduction from $\mathcal{R}$ to $\mathcal{R}_u$ in polynomial time.*

Proof. The reduction simply outputs the solution on $\mathcal{R}_u$. To see that this is a valid solution, simply note that $\mathcal{R}_u \subseteq \mathcal{R}$. $\square$

Lemma 3.14. *If $\mathcal{R} \in \text{TF}\Sigma_i^P$ has essentially unique solutions and is μ -d.s.r, then $\mathcal{R}_u \in \text{TF}\Sigma_i^P$ is μ -d.s.r with access to a Σ_{i-1}^P oracle.*

Proof. Let V_1, V_2 be the verifiers for $\mathcal{R}$. Let $\mathcal{A}^{\mathcal{R}, \Sigma_{i-1}^P}$ be the μ -d.s.r for $\mathcal{R}_u$. We will use this to construct $\mathcal{B}^{\mathcal{R}_u, \Sigma_{i-1}^P}$, a μ -d.s.r for $\mathcal{R}$. $\mathcal{B}^{\mathcal{R}_u, \Sigma_{i-1}^P}$ works as follows.

On input x , $\mathcal{B}^{\mathcal{R}_u, \Sigma_{i-1}^P}$ simulates $\mathcal{A}^{\mathcal{R}, \Sigma_{i-1}^P}$ and obtains a solution y such that $(x, y) \in \mathcal{R}$. This is feasible since an $\mathcal{R}_u$ oracle is also an $\mathcal{R}$ oracle by [Lemma 3.13](#).

If $V_1(x, y) = 0$, $\mathcal{B}^{\mathcal{R}_u, \Sigma_{i-1}^P}$ returns y . Otherwise, it uses its Σ_{i-1}^P oracle to find the lexicographically smallest y' such that $V_1(x, y') = 1$, and returns y' .

Notice that the reduction proceeds in polynomial time since V_1 can be verified in Σ_{i-2}^P , we can use a Σ_{i-1}^P oracle to find the lexicographically smallest y' such that $V_1(x, y') = 1$ if such a y' exists (which it must whenever this case is encountered in our simulation due to the definition of $\mathcal{R}_u$). Correctness of $\mathcal{B}^{\mathcal{R}_u, \Sigma_{i-1}^P}$ is inherited from the correctness $\mathcal{A}^{\mathcal{R}, \Sigma_{i-1}^P}$. $\square$

Theorem 3.15. *If $\mathcal{R} \in \text{TF}\Sigma_i^P$ has essentially unique solutions and is (randomized) μ -d.s.r with access to a Σ_{i-1}^P oracle, then $\mathcal{R}$ has a (randomized) reduction to $\text{UEOPL}^{\Sigma_{i-1}^P}$.*

Proof. This follows from [Lemma 3.13](#) and [Theorem 3.7](#). $\square$

4 New Upper Bounds for Avoid and the Linear Ordering Principle

In this section we will use the μ -d.s.r framework to show that both AVOID and LINEAR ORDERING PRINCIPLE are in UEOPL^{NP} ([Theorem 2](#) and [Theorem 3](#)).

We start by recalling the definition of AVOID that was introduced by [\[KKMP21, Kor22b\]](#).

Problem: Avoid

Input: A polynomial sized circuit $C : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$

Output: Find an element in $\{0, 1\}^{n+1}$ not in the range of the circuit C .

It is clear that AVOID describes a total problem since a solution is always guaranteed via the dual pigeonhole principle. Moreover any solution y to AVOID can be verified by an NP oracle since checking if y is a valid solution corresponds to checking that $\forall x: C(x) \neq y$ which is just a coNP statement. As a result we can safely put AVOID in $\text{TF}\Sigma_2^P$.

To get an upper bound on AVOID we follow the framework in [KP24] by proving an upper bound on an adjacent problem LINEAR ORDERING PRINCIPLE (LOP) which AVOID can be reduced to. Moreover since LOP has a unique solution, we end up being able to isolate a canonical solution for any instance of AVOID.

We now define the problem LINEAR ORDERING PRINCIPLE introduced in [KP24]

Problem: Linear Ordering Principle (LOP)

Input: A polynomial sized circuit $C : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$, encoding a total order $\prec$

Output:

1. Find a witness that $\prec$ does not define a total order i.e. $x, y, z \in \{0, 1\}^n$ such that one of the following holds: (i) $x \prec x$, (ii) $x \neq y$, $x \not\prec y$ and $y \not\prec x$ or (iii) $x \prec y \prec z$ and $x \not\prec z$.
2. Find the minimal element of the total order defined by $\prec$.

By definition, LOP is a total problem. Verifying a witness of the violation of a total order can be done in polynomial time. To verify that a solution y is actually the minimal element, we can use a NP oracle since it corresponds to checking the following coNP predicate $\forall x : \prec(y, x) = 1$. As a consequence we have that LOP is in $\text{TF}\Sigma_2^P$, and LOP has essentially unique solutions (Definition 3.11).

We will now show LOP is μ -d.s.r for polynomially bounded μ . The main idea is to reduce the problem of finding a minimal element in the total order to finding minimal elements a_0, a_1 in the first and second halves of the total order respectively, and then comparing a_0 and a_1 . This idea bears a resemblance to the proof in [HMR22] that the PLS-complete problem ITER-WITH-SOURCE is d.s.r.

Theorem 4.1. *LOP is μ -d.s.r with access to a NP oracle for polynomially bounded μ .*

Proof. Let $\prec: \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$ be our LOP instance. We define $\mu(\prec) = n$. Clearly, $\mu(\prec) \leq |\prec|$. If $\prec$ is not a total order, we can find the lexicographically smallest witness violating the total order using an NP oracle. We can ensure that the three types of violations have some lexicographic preference over each other.

Otherwise, $\prec$ must be a total order. For $i \in \{0, 1\}$, define $\prec_i : \{0, 1\}^{n-1} \times \{0, 1\}^{n-1} \rightarrow \{0, 1\}$ such that $\prec_i(x, y) = \prec(i||x, i||y)$. We query the oracle on $\prec_0$ and $\prec_1$, each has a unique solution a_0 and a_1 respectively. We output $\min(a_0, a_1)$ with respect to $\prec$.

$\prec_0$ and $\prec_1$ are total since $\prec$ is total. Therefore, $a_i \prec x$ for all $x \in i||\{0, 1\}^{n-1}$. Therefore $\min(a_0, a_1) \prec x$ for all $x \in \{0, 1\}^n$, as required. $\square$

Corollary 4.2. $\text{LOP} \in \text{UEOPL}^{\text{NP}}$, $\text{AVOID} \in \text{UEOPL}^{\text{NP}}$, and L_2^P reduces to UEOPL^{NP} .

Proof. The fact that $\text{LOP} \in \text{UEOPL}^{\text{NP}}$ follows from [Theorem 4.1](#) and [Theorem 3.15](#). This combined with the fact that AVOID reduces to LOP [\[KP24\]](#) lets us conclude $\text{AVOID} \in \text{UEOPL}^{\text{NP}}$. Finally, L_2^{P} is simply the class of all problems which are polynomial time many-one reducible to LOP [\[KP24\]](#). $\square$

Finally, as a consequence of the connections between solving AVOID and explicit construction of combinatorial objects we get the following corollary ([Corollary 2](#)):

Corollary 4.3. *There is an explicit construction of Ramsey Graphs, Two Source Extractors, Rigid Matrices, Linear Codes, Hard Truth Tables for any Fixed Polynomial Sized Circuit, K^{poly} -random strings in UEOPL^{NP}*

5 The Complexity of King

In this section we use ideas from the μ -d.s.r framework to show that the problem KING lies in $(\text{PLS}^{\Sigma_2^{\text{P}}} \cap \text{ZPP}^{\Sigma_2^{\text{P}}})$. In [Lemma 5.5](#) we identify a certain weak d.s.r property of KING that can be exploited, to design both $\text{PLS}^{\Sigma_2^{\text{P}}}$ and $\text{ZPP}^{\Sigma_2^{\text{P}}}$ algorithms. Surprisingly, the two approaches seem to use downward self-reducibility in very different ways.

Unlike the other total function problems we deal with, we will not be able to apply the μ -d.s.r framework directly since we will find that μ is not polynomially bounded (and we therefore cannot apply [Theorem 3.7](#)). But the ideas from [Theorem 3.7](#) will carry over to KING naturally and allow us to show KING is in $\text{PLS}^{\Sigma_2^{\text{P}}}$ and design non-trivial algorithms for KING .

We begin by reviewing the definitions from [\[KKMP21\]](#) on what it means for a player in a tournament to be a king.

Definition 5.1. A vertex v in a digraph G is called a king if every vertex in G can be reached from v by a path of length at most 2.

Definition 5.2. A digraph G is called a tournament if for every pair of distinct vertices $u, v \in G$, exactly one of the directed edges (u, v) or (v, u) is present in G .

In fact, there is a very simple lemma due to [\[Lan53\]](#) which shows:

Lemma 5.3. *Every tournament has a king.*

Proof. Let G be a digraph representing a tournament, and $\mathcal{N}(u)$ represent all vertices who have an edge incident from u in G . We now argue that any node v with maximum out-degree in G is a king. Towards contradiction, suppose that v is not a king. Then there must exist a vertex u , such that u is not incident to v , and u is not incident to any vertex in $\mathcal{N}(v)$. So the out-degree of $u \geq |\mathcal{N}(v)| + 1$, which is greater than out-degree of v , a contradiction. $\square$

[\[KKMP21\]](#) define the search problem KING which sits in Σ_3^{P} and whose totality follows from [Lemma 5.3](#). Consider as input a circuit $C : [2^n] \times [2^n] \rightarrow \{0, 1\}$ where one can imagine C implicitly encoding the digraph G corresponding to a tournament, i.e. $C(x, y) = 1$ implies that there is a directed edge from x to y in G . If C is not a tournament then there exists $x \neq y$ such that $C(x, y) = C(y, x)$ which can easily be checked by an NP oracle. If C is a tournament we are guaranteed to have a king.

Problem: KING**Input:** A circuit $C : [2^n] \times [2^n] \rightarrow \{0, 1\}$ encoding a digraph**Output:**

1. Find a distinct $x_1, x_2 \in [2^n]$ such that $C(x_1, x_2) = C(x_2, x_1)$. [A no witness, showing C does not encode a tournament]
2. Find an element $k \in [2^n]$ such that for every $x \in [2^n] \setminus \{k\}$, either $C(k, x) = 1$, or there exists an element $j \in [2^n] \setminus \{k, x\}$, such that $C(k, j) = 1$ and $C(j, x) = 1$. [A yes witness, a solution to KING].

We note that deciding whether there is a king for a digraph can be written as a Σ_3^P predicate: $\exists k, \forall x, \exists j$ s.t. $[C(k, x) = 1] \vee [C(k, j) = 1 \wedge C(j, x) = 1]$. By definition KING is total, and it is in $\text{TF}\Sigma_3^P$ since it's solutions can be verified by a Σ_2^P verifier:

1. We can verify the no witness (x_1, x_2) using an NP oracle. There exists $(x_1, x_2) \in [2^n]$ such that $C(x_1, x_2) = C(x_2, x_1)$.
2. We can verify the yes witness (k) using a Σ_2^P oracle. For all $x \neq k \in [2^n]$, there exists j such that either $C(k, x) = 1$ or $(C(k, j) = 1 \wedge C(j, x) = 1)$.

If we want to extract a solution for KING via the existence argument in [Lemma 5.3](#), it appears that we need to solve some kind of generic counting problem (outside of the polynomial hierarchy) to compute the neighborhood of a given node. Interestingly, we show that we need much less. Our starting point is the following crucial lemma which could be thought of as a weak d.s.r property about the structure of tournaments.

We start by defining what we call a *weak king* of a subset of vertices.

Definition 5.4 (Weak King). Let G be a digraph on a set of vertices V and let $U \subseteq V$ be a subset of vertices. A vertex $v \in U$ is called a weak king of U if every vertex in U can be reached from v by a path of length at most 2 in G .

We note that a weak king v for $U \subseteq V$ may not be a king in the induced subgraph $G[U]$ since it is allowed to use edges from $G \setminus G[U]$. On the other hand, if $U = V$, then a weak king is also a king.

Lemma 5.5. *Let G be a tournament on a set of vertices V where $|V| \geq 1$ and let $U \subseteq V$ be any subset of vertices. For any $v \in V \setminus U$, if u is a weak king for U , then either u or v is a weak king for $U \cup \{v\}$.*

Proof. Consider any such $G, U \subseteq V$ and $v \in V \setminus U$. Let u be a weak king for U . There can be two cases:

1. Suppose there exists a path of length at most 2 from u to v in G . Since u is a weak king for U , this implies u is also a king for $U \cup \{v\}$.
2. Suppose there is no path of length at most 2 from u to v in G . We will show that for any $x \in U$, there is a path of length at most 2 from v to x . We can have three cases:

- $x = u$: We know that $(u, v) \notin G$. Hence $(v, u) \in G$.
- $(u, x) \in G$: We know that $(x, v) \notin G$ for otherwise we found a path of length 2 from u to v . Hence $(v, x) \in G$.
- $\exists w \in V$, such that $(u, w), (w, x) \in G$: We know that $(w, v) \notin G$ for otherwise there is a path of length 2 from u to v . Hence $(v, w) \in G$ and we have the path $(v, w), (w, x)$.

Hence v is a weak king for $U \cup \{v\}$.

□

Lemma 5.5 now suggests a natural d.s.r strategy. Given a tournament G on a set of vertices V , finding a king is equivalent to finding a weak king for the whole vertex set V . Moreover, we can reduce the problem of finding a weak king for vertices $[1, t]$ to that of finding a weak king for vertices $[1, t-1]$, get back a weak king k' , and then output either k' or t , whichever is a weak king for $[1, t]$. Notice that by **Lemma 5.5**, either k' or t will be a weak king for $[1, t]$. This d.s.r does not show $\text{PLS}^{\Sigma_2^P}$ membership of KING because for a circuit C , $\mu(C)$ is the number of vertices in the tournament implicitly defined by C . μ is therefore not polynomially bounded. Fortunately, our proposed d.s.r is “memoryless” in that we do not need to keep a stack trace. Therefore, we do not need all the machinery of **Theorem 3.7**. We will use that insight to turn our proposed d.s.r into a reduction from KING to $\text{SINK-OF-DAG}^{\Sigma_2^P}$.

Theorem 5.6. KING is in $\text{PLS}^{\Sigma_2^P}$.

Proof. As previously discussed, a Σ_2^P oracle is sufficient to check the answer to a KING instance efficiently. We reduce KING to SINK-OF-DAG with a Σ_2^P oracle as follows. We are given a KING instance $C : [2^n] \times [2^n] \rightarrow \{0, 1\}$. We first use our Σ_2^P oracle to check if there exists distinct $x_1, x_2 \in [2^n]$ such that $C(x_1, x_2) = C(x_2, x_1)$. If there are, we can use the NP oracle to find x_1, x_2 and output them as a type 1 solution to KING. Notice that in this case the reduction runs in $\text{poly}(n)$ time and is correct. We will therefore assume for the remainder of the proof that C defines a proper tournament G .

We now show the reduction from KING to SINK-OF-DAG. We first specify the successor circuit $S : [2^n] \times [2^n] \rightarrow [2^n] \times [2^n]$. We should think of the first input i as an integer and the second input x as a vertex. S is defined as follows:

1. If x is not a weak king for $[1, i]$ (which one can check using a Σ_2^P oracle), $S(i, x) = (i, x)$.
2. For all x , $S(2^n, x) = (2^n, x)$.
3. If x is a weak king for $[1, i+1]$ (which one can check using a Σ_2^P oracle), $S(i, x) = (i+1, x)$.
4. If $i+1$ is a weak king for $[1, i+1]$ (which one can check using a Σ_2^P oracle), $S(i, x) = (i+1, i+1)$.

Notice that S covers all possible cases by **Lemma 5.5**.

We now specify the value circuit $V : [2^n] \times [2^n] \rightarrow [2^n]$, $V(i, k) = i$. The reduction calls the SINK-OF-DAG oracle on S, V and receives back (j, k) as output. Let $(j', k') = S(j, k)$. The reduction outputs k' . It should be clear that the reduction runs in time $\text{poly}(n)$. We now show correctness. By the definition of SINK-OF-DAG, $S(j, k) \neq (j, k)$ but one of the following two conditions hold.

1. $S(S(j, k)) = S(j, k)$: In other words $S(j', k') = (j', k')$. Notice that by the definition of S , since $S(j, k) \neq (j, k)$, k must be a weak king for $[1, j]$. Again by the definition of S , k' is a weak king for $[1, j' = j + 1]$. Since $S(j', k') = (j', k')$, but k' is a weak king for $[1, j' = j + 1]$, it must be the case that $j' = 2^n$ (otherwise $S(j', k')$ would equal $(j' + 1, k'')$ for some k''). Therefore, k' is a weak king for $[1, 2^n]$, and hence a king for G as desired.
2. $V(S(j, k)) \leq V(j, k)$: Notice that since $S(j, k) \neq (j, k)$, by the definition of S , either $S(j, k) = (j + 1, k)$ or $S(j, k) = (j + 1, j + 1)$. In either case $V(S(j, k)) = j + 1$. So it cannot be the case that $V(S(j, k)) \leq V(j, k)$.

□

To our surprise, we also find that KING is in $\text{TFZPP}^{\Sigma_2^P}$. This proof uses different ideas from downward self-reducibility, but interestingly still relies on [Lemma 5.5](#).

Theorem 5.7. KING is in $\text{TFZPP}^{\Sigma_2^P}$

We will need a uniform sampler for Σ_2^P relations, which follows from observing that the sampler for NP relations in [\[BGP00\]](#) directly applies for higher classes in the polynomial hierarchy.

Lemma 5.8 ([\[BGP00\]](#)). For $i \geq 1$, let R be a Σ_i^P relation. Then there is a uniform generator for R which is implementable in probabilistic, polynomial time with a Σ_i^P oracle.

Proof of Theorem 5.7. We are given a KING instance $C : [2^n] \times [2^n] \rightarrow \{0, 1\}$. We first use our Σ_2^P oracle to check if there exists distinct $x_1, x_2 \in [2^n]$ such that $C(x_1, x_2) = C(x_2, x_1)$. If there are, we can use the NP oracle to find x_1, x_2 and output them as a type 1 solution to KING. Notice that in this case the reduction runs in $\text{poly}(n)$ time and is correct. We will therefore assume for the remainder of the proof that C defines a proper tournament G .

For any vertex $u \in V$, we use W_u to denote the set of all vertices witnessing that u is not a king. Formally,

$$W_u := \{v \in V \setminus \{u\} : (u, v) \notin G, \forall w \in V \setminus \{u, v\}, (u, w) \notin G \vee (w, v) \notin G\}.$$

In particular, if u is a king, then $W_u = \emptyset$. For any given $u \in V$, we could sample uniformly from W_u in probabilistic polynomial time with a Σ_2^P oracle by [Lemma 5.8](#).

Next, we describe the algorithm for finding a king. Start with a vertex $u \in V$, if u is a king, we find a solution and terminate. Otherwise we sample v uniformly from W_u and repeat the process with v .

To see that the algorithm above terminates in (expected) polynomial-time, we have the following two claims:

Claim 5.9. For any $u \in V$ and $v \in W_u$, $W_v \subsetneq W_u$.

Proof. Let $U = V \setminus W_u$. By definition of W_u , u is a weak king for U .

Now by [Lemma 5.5](#), v is a weak king of $U \cup \{v\}$ since $v \in W_u$.

It follows that $W_v \subseteq V \setminus (U \cup \{v\}) = W_u \setminus \{v\} \subsetneq W_u$. □

Claim 5.10. For any $u \in V$,

$$\Pr_{v \sim W_u} \left[|W_v| \leq \frac{2|W_u|}{3} \right] \geq \frac{1}{4}.$$

Proof. Let $s = |W_u|$. Consider the induced tournament $G[W_u]$ on the vertex set W_u . The total outdegree is exactly $s(s-1)/2$. By a simple counting argument, at least $1/4$ of the vertices in W_u have outdegree at least $(s-1)/3$. Moreover, any vertex w that is an outgoing neighbor of v cannot be in W_v . Combining these two facts yields

$$\Pr_{v \sim W_u} \left[|W_v| \leq (s-1) - \frac{s-1}{3} \right] \geq \frac{1}{4}.$$

□

Now we can measure the progress of our algorithm by $|W_u|$, which in expectation decreases by a constant factor in constant number of iterations. It terminates when $|W_u| = 0$ which takes in expectation $O(\text{polylog}(|V|)) = O(\text{poly}(n))$ iterations. □

Theorem 5.7 stands in striking contrast to the following theorem. This indicates that the Σ_2^P oracle is in some sense making KING much easier.

Theorem 5.11 ([MPS23]). *In the black-box query model, any randomized algorithm for KING requires at least $\Omega(2^n)$ time.*

Finally, as a side note, we show that there exists a faster than trivial algorithm to solve KING if we are given access to an NP oracle.

Theorem 5.12. *There exists a $\text{poly}(n)2^n$ time algorithm using an NP oracle algorithm for KING.*

Proof. The algorithm $\mathcal{A}$ is as follows. $\mathcal{A}$ uses the NP oracle to first find distinct $x, y \in [2^n]$ such that $C(x, y) = C(y, x)$ (if any) and if it finds them outputs x, y as a type 1 solution to KING. This can be achieved in polynomial time with access to an NP oracle. We therefore assume that $C(x, y) \neq C(y, x)$ for all $x \neq y$ for the remainder of the proof.

The algorithm runs for 2^n steps. At step i , we aim to find a weak king v_i for $[1, i]$. Clearly, at step 1, $v_1 = 1$ is a weak king of $[1, 1]$. When we are at step i , **Lemma 5.5** tells us that a weak king of $[1, i]$ is either v_{i-1} or i . Furthermore, it provides an easy check to determine if v_{i-1} or i is a weak king. If there is a length 2 path in from v_{i-1} to i , then $v_i = v_{i-1}$ is a weak king for $[1, i]$, otherwise $v_i = i$ is a weak king for $[1, i]$. We can check this using the NP oracle which tells us if there exists a $z \in V$ such that $C(v_{i-1}, z) = 1$ and $C(z, i) = 1$. Finally, the algorithm outputs v_{2^n} , the weak king of $[1, 2^n]$ which is by definition a king.

Since each step of the algorithm takes at most $\text{poly}(n)$ time with access to the NP oracle, we also get our desired runtime. Correctness follows from **Lemma 5.5**. □

Our NP oracle algorithm for KING compares favorably to lower bounds for finding a king in a tournament without access to an oracle.

Theorem 5.13 ([SSW03]). *In the black-box query model, any deterministic algorithm for KING requires at least $\Omega(2^{4n/3})$ time.*

Theorem 5.13 and **Theorem 5.11** were originally stated for the problem of finding a king in a tournament on t vertices. They showed a $\Omega(t^{4/3})$ query lower bound for deterministic algorithms and a $\Omega(2^n)$ query lower bound for randomized algorithms respectively. Setting $t = 2^n$ yields **Theorem 5.13** and **Theorem 5.11**. We find it somewhat interesting that access to a NP provides a provable speedup for deterministic algorithms (in the query model) from $\Omega(2^{4n/3})$ to $\text{poly}(n)2^n$ and that access to a Σ_2^P oracle provides a provable speedup for randomized algorithms (in the query model) from $\Omega(2^n)$ to $\text{poly}(n)$.

6 Downward self-reducibility in TFNP

We now demonstrate the utility of the μ -d.s.r framework by applying it to several TFNP problems. For P-LCP and MEMDET, we are able to dramatically simplify proofs of UEOPL and PLS membership respectively. Interestingly, unlike most proofs of membership in a TFNP subclass, our proofs of PLS/UEOPL membership do not mirror (or even resemble) the proof of totality for the base problem. For the TARSKI problem, we give a proof of PLS membership which mirrors the divide-and-conquer algorithm for finding Tarski fixed points. Although this proof of PLS membership is not syntactically simpler than the original [EPRY20], we view it as conceptually simpler since it only requires knowledge of the well-known divide-and-conquer algorithm for TARSKI. Finally, we show UEOPL membership for the S-ARRIVAL problem. This proof is admittedly significantly more complicated than the original proof [GHH⁺18]¹². However, we have chosen to include it because we hope it will provide an alternative perspective which may be useful in future analysis of the S-ARRIVAL problem.

6.1 P-LCP

We begin by defining a P-matrix, the linear complementarity problem, and the PROMISE-P-LCP problem.

Definition 6.1. A matrix $M \in \mathbb{R}^{n \times n}$ is a P-matrix if every principal minor is positive.

Definition 6.2. For any $M \in \mathbb{R}^{n \times n}$ and $q \in \mathbb{R}^n$, we say $z \in \mathbb{R}^n$ is a solution to the linear complementarity problem (LCP) if all the following conditions hold.

1. $z \geq 0$,
2. $y = q + Mz \geq 0$,
3. $z^T y = 0$.

Problem: Promise-P-LCP

Input: A P-matrix $M \in \mathbb{R}^{n \times n}$ and $q \in \mathbb{R}^n$.

Output: A solution z to the linear complementarity problem with inputs M and q .

The following lemma will be useful in showing a μ -d.s.r for PROMISE-P-LCP and immediately implies uniqueness of solutions for PROMISE-P-LCP.

Lemma 6.3 ([STW58]). *$M \in \mathbb{R}^{n \times n}$ is a P-matrix if and only if for any $q \in \mathbb{R}^n$, LCP with input M, q has exactly one solution.*

Lemma 6.4. PROMISE-P-LCP μ -d.s.r for polynomially bounded μ .

¹²technically this work only showed CLS membership, but it was observed in [FGMS20] that the proof also implies UEOPL membership

Proof. For an input matrix $M \in \mathbb{R}^{n \times n}$ and $q \in \mathbb{R}^n$, we define the measure $\mu(M) = n$. We now show a downward self-reduction for PROMISE-P-LCP. If $-M^{-1}q \geq 0$, output $-M^{-1}q$. Otherwise, let $M_i \in \mathbb{R}^{(n-1) \times (n-1)}$ denote M with row i and column i removed. Similarly, let $q_i \in \mathbb{R}^{n-1}$ denote q with entry i removed. The reduction queries its oracle on $(M_1, q_1), \dots, (M_n, q_n)$ to get back the answers $z_1, \dots, z_n \in \mathbb{R}^{n-1}$ respectively. We find i such that z_i is a solution to LCP(M, q) after inserting a 0 in the i -th coordinate, and output the solution.

This reduction clearly runs in $\text{poly}(n)$ time. It is also a downward reduction since $\mu(M_j) = n-1$ for all $j \in [1, n]$. We see that it is also promise-preserving since if M is a P-matrix, then M_j is also a P-matrix for all $j \in [1, n]$.

If the reduction outputs $z = M^{-1}q$, then clearly $z \geq 0$, $y = q + Mz = 0$, and $z^T y = 0$. Otherwise, consider z^* , the solution to LCP on input M, q (and note that z^* is unique by [Lemma 6.3](#)). If z^* contains no zero coordinates, then $y = (0, \dots, 0)$ since $z^{*T} y = 0$. But this implies $z^* = -M^{-1}q$, which cannot be true since that case has already been checked. So let i be any coordinate of z^* that is zero and consider $z_{-i}^* \in \mathbb{R}^{n-1}$, z^* with the i^{th} coordinate removed. z_{-i}^* must be a solution to the LCP on input M_i, q_i since z^* was a solution to LCP on input M, q . Notice that by [Lemma 6.3](#), z_{-i}^* is the only solution to (M_i, q_i) . Therefore, by the correctness of the d.s.r., $z_i = z_{-i}^*$, and the reduction outputs z^* as desired. $\square$

Corollary 6.5. PROMISE-P-LCP reduces to UEPOL.

Proof. PROMISE-P-LCP is clearly a $\text{Promise}\Sigma_1^P$ problem. [Lemma 6.3](#) shows that it has unique solutions and [Lemma 6.4](#) shows that it is μ -d.s.r for polynomially bounded μ . We can therefore apply [Theorem 3.7](#) to show that PROMISE-P-LCP reduces to UEPOL. $\square$

6.2 Graph Games

We begin by defining graph games [\[BM08\]](#).

Definition 6.6 (Graph Games, [\[BM08\]](#)). $G = (V_0, V_1, E)$ is a graph game of size n if

1. V_i are the positions of player P_i , for $i = 0, 1$. They have to satisfy $V_0 \cap V_1 = \emptyset$, and $V_0 \cup V_1 \subseteq \{1, \dots, n\}$. $V := V_0 \cup V_1$ is the set of all positions.
2. $E \subseteq V \times V$ is the set of possible moves.
3. In graph-theoretic terms, V is the set of nodes, and E the set of edges of graph G . They have to satisfy in addition that at least one edge is leaving each node.

If $v \in V_i$, we say that player P_i owns v .

As an example, in [Figure 2](#), $V_0 = \{1, 3, 5\}$, $V_1 = \{2, 4\}$, and $E = \{(1, 2), (2, 1), (1, 3), (5, 1), (2, 4), (3, 4), (3, 5), (4, 5)\}$.

Definition 6.7 (Playing and Winning, [\[BM08\]](#)). A play from a node $v \in V$ is an infinite path $v = v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \dots$ in G with each edge $v_i \rightarrow v_{i+1} \in E$ chosen by the player owning v_i . The winner of a play is the player owning the least node which is visited infinitely often in the play.

As an example, in [Figure 2](#), a play may be $1 \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \dots$. In this example, the player 1 (the blue player), made moves 1, 3, 4 and player 2 (the red player), made moves 2 and 5.

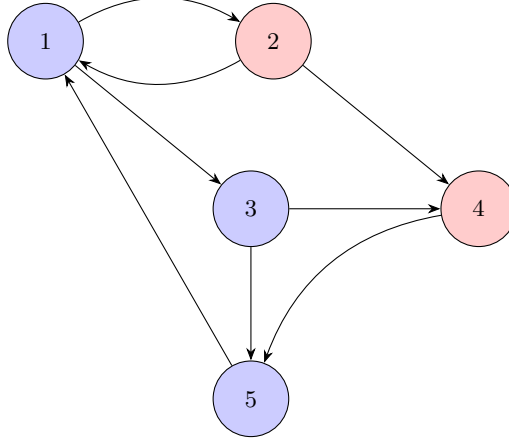


Figure 2: An example graph game

Definition 6.8. Let $G = (V_0, V_1, E)$ and $V = V_0 \cup V_1$. We say G has a memoryless deterministic strategy for player i if there exists a function $\sigma : V_i \rightarrow V$ such that for all $v \in V_i$, $(v, \sigma(v)) \in E$, and σ is a dominant strategy for player i . In particular, even if the other player uses a non-memoryless strategy, player i still wins by playing σ .

Lemma 6.9 ([BM08]). *For every simple graph game G , a memoryless deterministic strategy exists, and can be verified in polynomial time.*

We are now ready to define the TFNP problem corresponding to simple graph games, and thus to parity games.

Problem: Memdet

Input: A canonical representation of a graph game $G = (V_0, V_1, E)$

Output: A winner $i \in \{0, 1\}$ and a memoryless deterministic strategy $\sigma : V_i \rightarrow V$ for player i .

Lemma 6.9 shows that MEMDET is in TFNP. Now, we will use downward self-reducibility to show that MEMDET is in PLS.

Theorem 6.10. *MEMDET is μ -downward self-reducible for a polynomially bounded μ .*

Proof. For a MEMDET instance $G = (V_0, V_1, E)$, we define $\mu(G) = |E| - |V_0| - |V_1|$. Notice that when $\mu(G) = 0$, each vertex has exactly one edge coming out of it. Therefore, each player has exactly one possible memoryless deterministic strategy. We can therefore determine which strategy wins in polynomial time. In this case, MEMDET is solvable in polynomial time. It should also be clear that $\mu(G) \leq |G|$.

We now show MEMDET is μ -d.s.r. Assume $\mu(G) > 0$. We say that player i has a forced strategy if each vertex in V_i has exactly one edge coming out of it. The reduction first checks if there is a forced strategy which also a winning strategy for either player. If so, it outputs the solution it finds. Otherwise, let G_i denote G with edge i removed. Notice however that G_i is not necessarily

a graph game as defined in [Definition 6.6](#) since it may be the case that G_i does not have at least one edge leaving each vertex. So define $\mathcal{G} = \{G_i : i \in E, G_i \text{ is a graph game}\}$. The self reduction constructs $\mathcal{G}$ and calls the MEMDET oracle on every graph in $\mathcal{G}$ to get back solutions of the form (p, σ_i) where σ_i is a memoryless deterministic strategy for player p to win in G_i . The reduction then checks for every (p, σ_i) it receives from the oracle if σ_i is a memoryless, deterministic strategy for G , and outputs the first (p, σ_i) that is. If no such strategy is found (which we will show never happens), the reduction outputs 0.

We first show that the reduction is valid. The construction of $\mathcal{G}$ clearly runs in polynomial time. Checking whether σ_i is a winning strategy for G can be done in polynomial time and therefore so is checking all at most $|G|$ of them. Therefore, the reduction runs in polynomial time. Furthermore, we see that $\mu(G_i) = \mu(G) - 1$ for every $i \in E$, so the reduction is indeed a *downward* self-reduction.

We now show correctness. We only show the reduction succeeds when $\mu(G) \geq 1$, since otherwise, the MEMDET is solvable in polynomial time. We must show that one of the (p, σ_i) is a solution to MEMDET on G . Let σ^* be a memoryless, deterministic strategy for G . Notice that since $\mu(G) \geq 1$, there must exist an edge i^* we can eliminate to create G_{i^*} such that (p, σ^*) is a solution to MEMDET on G_{i^*} . Here we consider two cases.

1. Say p has a forced strategy in for G . The reduction clearly succeeds.
2. Say player p does not have a forced strategy for G . Let us consider what happens when i^* is any edge belonging to p which is not chosen in σ^* . Then player p wins in G_{i^*} since σ^* is clearly a winning strategy for G_{i^*} . Therefore, the solution to G_{i^*} is (p, σ') for some σ' . Notice that (p, σ') is a solution for G since the player opposing p has the same options in G_{i^*} as they do in G . Therefore, any solution to G_{i^*} is a solution to G .

□

Corollary 6.11. MEMDET is in PLS.

We note that the problem of finding memoryless deterministic strategies for parity games reduces to MEMDET [\[BM08\]](#). Therefore, the problem of finding memoryless deterministic strategies for parity games is also in PLS.

6.3 Tarski

Consider any integers $L, d \geq 1$. For an integer lattice $[L]^d$, we define a partial order $\preceq$ such that $x \preceq y$ if for all $i \in [d]$, $x_i \leq y_i$. We now define the TFNP problem of finding a Tarski fixed point [\[Tar55, EPRY20\]](#).

Problem: Tarski

Input: $N = 2^n, n, d \geq 1$. Given $C : [N]^d \rightarrow [N]^d$ with $\text{size}(C) \leq \text{poly}(d, n)$.

Output:

1. $x \in [N]^d$ such that $C(x) = x$.

2. distinct $x, y \in [N]^d$ such that $x \preceq y$ but $C(x) \not\preceq C(y)$.

Although it is unclear how this problem is itself downward self-reducible, we can define a new more general problem, which we call TARSKI+, which is indeed downward self-reducible. For $\ell, h \in [N]^d$, let $L(\ell, h) = \{x \in [N]^d : \ell \preceq x \preceq h\}$.

Problem: Tarski+

Input: Let $N = 2^n, n, d \geq 1, t \in [2, d+1]$. Given $C : [N]^d \rightarrow [N]^d$ with $\text{size}(C) \leq \text{poly}(d, n)$ $m_t, \dots, m_d \in [N]$, $\ell, h \in [N]^{t-1}$.

Define $f : L(\ell, h) \rightarrow [N]^{t-1}$ as $f(m_1, \dots, m_{t-1}) = C(m_1, \dots, m_{t-1}, m_t, \dots, m_d)_{[1, t-1]}$.

Output:

1. $x \in [N]^d$ such that $C(x) = x$.
2. distinct $x, y \in [N]^d$ such that $x \preceq y$ but $C(x) \not\preceq C(y)$.
3. x such that $f(x) \notin L(\ell, h)$.

In the above definition, we imagine that m_i is some n -bit string encoding $\perp$ if $i \in [t, d]$. TARSKI+ simply lets consider TARSKI problem on a sublattice $L(\ell, h)$ where $\ell, h \in [N]^{t-1}$ and fixed $m_t, \dots, m_d$. Our downward self-reduction for TARSKI+ is now simply based on the classic divide-and-conquer algorithm for finding Tarski fixed points. See [DQY11] for a description of this algorithm, or [EPY20] for a compressed description.

Lemma 6.12. TARSKI+ is μ -d.s.r for polynomially bounded μ .

Proof. Consider a TARSKI+ instance $T = (C, m_t, \dots, m_d, \ell, h)$. Let $\mu(T) = t + \lceil \log_2(|L(\ell, h)|) \rceil$. Note that μ is clearly polynomially bounded in the instance size. We now show a μ -d.s.r for T . By definition, $f : L(\ell, h) \rightarrow [N]^{t-1}$ is such that $f(m_1, \dots, m_{t-1}) = C(m_1, \dots, m_{t-1}, m_t, \dots, m_d)_{[1, t-1]}$. If $t = 2$ or $|L(\ell, h)| \leq 100$, the reduction just solves the instance (in the case $t = 2$ via a binary search, and in the case $|L(\ell, h)| \leq 100$ by brute force).

Otherwise, let $t' = t - 1$ and $m = m_{t'} = \lceil (\ell_{t'} + h_{t'})/2 \rceil$. The reduction calls its oracle on $C, m_{t'}, m_t, \dots, m_d, \ell_{[1, t'-1]}, h_{[1, t'-1]}$ and receives an answer back. Depending on the answer, we can have a number of cases. First, we will show how to find a solution for the current instance in each case and argue its correctness.

First, let's handle the simpler case where the oracle returns a solution of type 2 or 3:

1. If it receives a type 2 solution $x, y \in [N]^{t'-1}$, it outputs a type 2 solution $(x, m), (y, m)$. Let $f'(x) = f(x, m)_{[1, t'-1]}$ be the implicit function in the oracle call. Clearly if $x \preceq y$ but $f'(x) \not\preceq f'(y)$, we also have that $(x, m) \preceq (y, m)$ but $f(x, m) = (f'(x), a) \not\preceq (f'(y), b) = f(y, m)$, for some $a, b \in [N]$. This is because $\not\preceq$ is preserved under concatenating any element on either side.
2. If it receives a type 3 solution $x \in [N]^{t'-1}$ such that $f'(x) \notin L(\ell_{[1, t'-1]}, h_{[1, t'-1]})$. The reduction outputs a type 3 solution (x, m) . Note $f(x, m)_{[1, t'-1]} \notin L(\ell_{[1, t'-1]}, h_{[1, t'-1]})$. Notice that this property is persevered by adding coordinates, so $f(x, m) \notin L(\ell, h)$, and (x, m) forms a type 3 solution to our instance.

Otherwise, the reduction receives a type 1 solution $x^* \in [N]^{t'-1}$ such that $f'(x^*) = x^*$. Here the output x^* of the first oracle call has the property $f(x^*, m)_{[1, t']} = x^*$ and for all $i \in [1, t' - 1]$, $\ell_i \leq x_i^* \leq h_i$. We can have four cases:

1. If $f(x^*, m)_{t'} \notin [\ell_{t'}, h_{t'}]$, then output (x^*, m) as a type 3 solution. Note $f(x^*, m)_{t'} \notin [\ell_{t'}, h_{t'}]$, then $f(x^*, m) \notin L(\ell, h)$. But clearly $(x^*, m) \in L(\ell, h)$ since for all $i \in [1, t' - 1]$, $\ell_i \leq x_i^* \leq h_i$ and $\ell_{t'} \leq m \leq h_{t'}$. Therefore (x^*, m) is a type 3 solution.
2. If $f(x^*, m)_{t'} = m$, the reduction outputs (x^*, m) as a type 1 solution since $f(x^*, m) = (x^*, m)$.
3. If $f(x^*, m)_{t'} > m$, the the reduction calls its oracle on the input $C, m_t, \dots, m_d, f(x^*, m), h$. If the oracle returns a type 1 solution, z , the reduction outputs z . If the reduction receives a type 2 solution, z_1, z_2 , the reduction outputs z_1, z_2 . These are both valid since $L(f(x^*, m), h) \subseteq L(\ell, h)$.

If the reduction receives a type 3 solution $z \in [N]^{t'}$, this implies $z \in L(f(x^*, m), h)$ but $f(z) \notin L(f(x^*, m), h)$. We can have two cases:

- If $f(x^*, m) \not\preceq f(z)$, output $(x^*, m), z$ as a type 2 solution. Note that $(x^*, m) \preceq f(x^*, m) \preceq z$ where the second inequality follows from the fact that $z \in L(f(x^*, m), h)$ and the first inequality follows from the fact that $f'(x^*) = x^*$ and $f(x^*, m)_{t'} > m$. Therefore, $(x^*, m) \preceq z$ but $f(x^*, m) \not\preceq f(z)$, so $(x^*, m), z$ forms a type 2 solution to our instance.
 - Otherwise $f(z) \not\preceq h$, then output z as a type 3 solution. Since $f(z) \notin L(\ell, h)$, z is a type 3 solution to the instance.
4. If $f(x^*, m)_{t'} < m$, the reduction calls its oracle on the input $C, m_t, \dots, m_d, \ell, f(x^*, m)$. If the oracle returns a type 1 solution, z , the reduction outputs z . If the reduction receives a type 2 solution, z_1, z_2 , the reduction outputs z_1, z_2 . If the reduction receives at type 3 solution z and $f(z) \not\preceq f(x^*, m)$, output $z, f(x^*, m)$ as a type 2 solution. Otherwise, output z as a type 3 solution. Correctness follows by a similar argument as in the previous case.

We have proved the correctness of our reduction. Finally, we show that the potential μ decreases by at least one at each step. Clearly in the first oracle call in the μ -d.s.r, the number of points in $L(\ell, h)$ does not go up, but t goes down by at least 1, so μ goes down by at least one. In the other oracle call, t stays the same, but by our choice of m , $|L(\ell, h)|$ goes down by at least a factor 2. In particular, the lattice the oracle is queried on is a sub-lattice of $L(\ell, h)$ where the last coordinate is only allowed to range over half the values it was originally allowed to range over. Therefore, μ decreases by at least 1. $\square$

Corollary 6.13. Tarski is in PLS.

Proof. Tarski reduces to Tarski+ by setting $t = d + 1$ and $\ell = (0, \dots, 0), h = (N, \dots, N)$. Tarski+ is in PLS by [Lemma 6.12](#) and [Corollary 3.9](#). Therefore, Tarski is in PLS. $\square$

6.4 S-Arrival

ARRIVAL is a zero-player variant of graph games ([Definition 6.6](#)). Here, instead of players making the decision of where to move, we imagine a train which is making the decision all by itself. We imagine a graph on n vertices where each vertex i has exactly two directed edges coming out of it

e_i^0, e_i^1 . The j^{th} time the train arrives at vertex i , it takes the edge $e_i^{(j \bmod 2)}$. The ARRIVAL problem asks us to prove that the train will or will not eventually make it to its destination d . We now define the ARRIVAL problem more formally.

Definition 6.14 (Switch Graph, [GHH⁺18]). A switch graph is a tuple $G = (V, E, s_0, s_1)$ where $s_0, s_1 : V \rightarrow V$ and $E = \{(v, s_i(v)) \mid \forall v \in V, i \in \{0, 1\}\}$.

Algorithm 1 RUN [GHH⁺18]

Require: A switch graph $G = (V, E, s_0, s_1)$ and two vertices $o, d \in V$.

Ensure: For each edge $e \in E$, the number of times the train traversed e .

```

 $v \leftarrow o$  ▷ position of the train
 $\forall u \in V$ , set  $s_{\text{curr}}[u] \leftarrow s_0(u)$  and  $s_{\text{next}}[u] \leftarrow s_1(u)$ 
 $\forall e \in E$ , set  $r[e] \leftarrow 0$  ▷ initialize the run profile
 $\text{step} \leftarrow 0$ 
while  $v \neq d$  do
     $w \leftarrow s_{\text{curr}}[v]$  ▷ compute the next vertex
     $r[s_{\text{curr}}[v]] + 1$  ▷ update the run profile
     $\text{swap}(s_{\text{curr}}[v], s_{\text{next}}[v])$ 
     $v \leftarrow w$  ▷ move the train
     $\text{step} \leftarrow \text{step} + 1$ 
end while
return  $r$ 

```

Definition 6.15 (ARRIVAL, [DGK⁺17]). Given a switch graph $G = (V, E, s_0, s_1)$ and two vertices $o, d \in V$, the ARRIVAL problem is to decide whether the algorithm RUN (Section 6.4) terminates, i.e., whether the train reaches the destination d starting from the origin o .

For an instance G of ARRIVAL, we first define some notation. A run is simply an ordered tuple of edges $(e_1, \dots, e_t)$. When starting at o on a graph G , if the train traverses $e_1, \dots, e_t$ in order for some consecutive portion, then $(e_1, \dots, e_t)$ is called a run for G, o . The run profile of a run $m = (e_1, \dots, e_t)$ is a vector v indexed by the edges in E such that v_e is the number of times e appears in m . We say that a run profile r for G, o is valid if there exists a run $m = (e_1, \dots, e_t)$ such that m is a run for G and r is the run profile of m . Given two runs m and m' such that (m, m') is also a valid run, it is not hard to see that $r_m + r_{m'}$ is a valid run profile for (m, m') . We will often use this correspondence between concatenating two runs and easily being able to compute their combined run profile from their individual run profiles. Finally, we refer to the set of edges specified by the map s_{curr} as the active edges.

The above problem is a decision problem, but to define the search problem S-ARRIVAL, we need the validity of run profiles to be efficiently verifiable. For a destination d , let $V_{\text{good}}(d)$ denote the set of all vertices v for which there exists a directed path in (V, E) from v to d and let $V_{\text{bad}}(d)$ denote $V \setminus V_{\text{good}}$.

Lemma 6.16 ([DGK⁺17]). *For an arrival instance $G = (V, E, s_0, s_1)$, o, d , the train eventually reaches d from o or it eventually reaches a vertex in $V_{\text{bad}}(d)$ from o . Furthermore, it always reaches one of these nodes in at most $n2^n$ steps.*

Notice that if the train reaches d from o , the run ends. If the train reached $V_{\text{bad}}(d)$ from o , then it has no hope of reaching d , since there is no path possible from any vertex in $V_{\text{bad}}(d)$ to d . Since there are at most $n2^n$ steps in either case, the entries in the run profile sum up to $n2^n$. This means the run profile can be represented by $O(n)$ bits.

Lemma 6.17 ([GHH⁺18]). *For any run profile of length $O(n)$ bits, there exists an $\text{poly}(n)$ time algorithm to check if the run profile is valid. Furthermore, for a valid run profile, the final vertex of the run can be determined in polynomial time.*

Lemma 6.16 and **Lemma 6.17** together tell us that there exists a witness that train reaches d or not. If the train reaches d , the run profile that ends at d is a valid and efficiently checkable witness of this fact. If the train does not reach d , the run profile that ends at V_{bad} (without going through an outgoing edge of d) is an efficiently checkable witness of this fact.

We are now ready to define the S-ARRIVAL problem. We use the original definition from [Kar17]:

Problem: S-Arrival

Input: A switch graph $G = (V, E, s_0, s_1)$ and two vertices $o, d \in V$. First, we define a graph G' as follows:

- Add a new vertex $\bar{d}$.
- For each vertex $v \in V_{\text{bad}}$, set $s_0(v) = \bar{d}$ and $s_1(v) = \bar{d}$.
- Edges $s_0(d), s_1(d), s_0(\bar{d}), s_1(\bar{d})$ are self-loops.

Output: The task is to find a run profile r starting at o and ending at either d or $\bar{d}$ (such that either is visited exactly once)

We now confirm for ourselves that S-ARRIVAL is in TFUP. In polynomial time, we can find the vertices in V_{bad} . This will allow us to construct the graph G' . By **Lemma 6.16**, we know that the run profile can be represented by at most $O(n)$ bits. By **Lemma 6.17**, we can verify if the run profile is valid in polynomial time. From the run profile, we can also verify that the final vertex in the run is either d or $\bar{d}$ and that they only appear once. This proves that S-ARRIVAL is in TFUP. We now show that S-ARRIVAL is d.s.r.

Theorem 6.18. *S-ARRIVAL is μ -d.s.r. for some polynomially bounded μ .*

Proof Sketch. We define our measure $\mu(G) = |V|$, the number of vertices in G . In this proof, we will use run profiles interchangeably with any runs they might represent. The only modification to runs we will do is concatenating two runs together, thereby also ensuring that the new run profile can also be easily computed.

We will downward self-reduce an S-ARRIVAL instance I where $\mu(I) = n$. I consists of $G = (V, E, s_0, s_1)$ and $o, d \in V$. Let V_{good} be the set of vertices which have an outgoing edge to d and E_{good} be the corresponding edges. If V_{good} was empty, we could output the run $(o, \bar{d})$. Now, consider any $v \in V_{\text{good}}$ such that $(v, d) \in E_{\text{good}}$. Modify the graph G to produce $G^{(1)} = (V^{(1)}, E^{(1)}, s_0^{(1)}, s_1^{(1)})$ such that each edge that leads to d ($(u, d) \in E$) leads to v instead ($(u, v) \in E^{(1)}$). First, observe that $V_{\text{bad}} = V_{\text{bad}}^{(1)}$.

Let $m^{(1)}$ be a run that is a solution to S-ARRIVAL on $G^{(1)}$, $o^{(1)} = o$, $d^{(1)} = v$. Now, we can have a few cases:

1. $m^{(1)}$ ends at node $\bar{d}$: we return $m^{(1)}$ since it is also a valid run in G' that ends at $\bar{d}$ since no such run can pass through a node in V_{good} .
2. $m^{(1)}$ ends at node v via an edge $(u, d) \in E_{good}$: we can easily modify the run to include (u, d) instead of (u, v) to produce a valid run m in G' that ends at d .
3. $m^{(1)}$ ends at node v via an edge $(u, v) \notin E_{good}$ and $(v, d) \in s_0$: we can modify $m^{(1)}$ by adding an edge (v, d) to produce a valid run m in G' ending at d .
4. If $m^{(1)}$ ends at node v via an edge $(u, d) \notin E_{good}$ and $(v, d) \in s_1$: In this case, since we have already reached v once, we must reach v again in order to use (v, d) to reach d . Let $o^{(2)} = s_0(v)$. Note $(v, o^{(2)})$ is the next edge visited by a run in G' . Modify $m^{(1)}$ to concatenate the edge $(v, s_0(v))$. Next, we modify $G^{(1)}$ to reflect the fact that we have already traversed the run $m^{(1)}$. For every edge $e \in s_i^{(1)}$, $e \in s_i^{(2)}$ iff e is visited an even number of times in the run $m^{(1)}$. Note that this can be computed by the run profile of $m^{(1)}$. Now, we use the oracle to receive a solution $m^{(2)}$ to S-ARRIVAL on $G^{(2)}$, $o^{(2)}$, v . Now, we can have three cases reflecting the previous three cases:
 - (a) $m^{(2)}$ ends at node $\bar{d}$: we return $m^{(1)}$ concatenated with $m^{(2)}$ since it is also a valid run in G' that ends at $\bar{d}$.
 - (b) $m^{(2)}$ ends at node v via an edge $(u, d) \in E_{good}$: we can easily modify the run, changing (u, v) to (u, d) instead. Now, $m^{(1)}$ concatenated with $m^{(2)}$ is a valid run in G' ending at d .
 - (c) $m^{(2)}$ ends at node v via an edge $(u, d) \notin E_{good}$: We return the run m which is $m^{(1)}$ concatenated with $m^{(2)}$ and (v, d) . We know that the run $m^{(1)}$ concatenated with $m^{(2)}$ ends at v and visits v exactly twice. Furthermore, $(v, d) \in s_1$, therefore the next edge taken by the train will be (v, d) which makes m a valid run in G' ending in d .

We have showed the correctness of our reduction. This reduction clearly runs in polynomial time as each graph modification and run (profile) modification can be carried out in polynomial time as noted above. Finally, the reduction has the downward property. It only calls the oracle on smaller instances as $\mu(G^{(1)}) = \mu(G^{(2)}) = \mu(G) - 1$. □

Corollary 6.19. S-ARRIVAL is in UEOPL.

Proof. The fact that S-ARRIVAL is in TFUP and d.s.r (Theorem 6.18) implies that it is in UEOPL by Corollary 3.9. □

7 Limitations of self-reducibility as a framework in TFNP

7.1 Not all PLS-complete problems are traditionally d.s.r

[HMR22] asked if (traditional) downward self-reducibility is a complete framework for PLS-complete problems. Recall that traditional downward self-reducibility means that the oracle can only answer

queries of input length strictly less than n , rather than smaller in some size function μ . This would be a rather surprising characterization of PLS-completeness. Here, we give a negative answer to this question under the assumption that $\text{PLS} \neq \text{FP}$. To do so, we will exhibit a problem which is PLS-complete but not traditionally d.s.r.

Problem: RestrictedSizeSoD

Input: A bitstring x of length $2^{100 \cdot 2^k}$ for some integer k .

Output: If $|x| \neq 2^{100 \cdot 2^k}$ for some integer k , 0 is the only valid answer. Otherwise, let $x = 0^t \circ 1 \circ x'$ for some integer t and string x' . y is a solution if and only if y is a solution to the SINK-OF-DAG on input x' .

In RESTRICTEDSIZESoD, one should think of only certain input sizes being valid. In particular, input sizes of length 2^{2^k} for some integer k (where we allow padding) are valid. To ensure totality though, in cases when the input is not of length 2^{2^k} for some integer k , we say exactly the trivial solution 0 is a solution.

Lemma 7.1. RESTRICTEDSIZESoD is in PLS.

Proof. Given an input $x \in \{0, 1\}^n$ to RESTRICTEDSIZESoD, we can solve it using a PLS oracle as follows. If $n \neq 2^{2^k}$ for any integer k , we output 0. Otherwise, we let $x = 0^t \circ 1 \circ x'$ for some integer t and string x' , use our PLS oracle to solve SINK-OF-DAG on x' , and output that solution. The fact that the reduction runs in polynomial time and is correct is immediate. $\square$

Lemma 7.2. RESTRICTEDSIZESoD is PLS-hard.

Proof. Say $x' \in \{0, 1\}^n$ is some input to SINK-OF-DAG. Let $k = \lceil \log_2(\log_2 n) \rceil + 1$ and $n' = 2^{2^k}$. We can reduce SINK-OF-DAG on x' to RESTRICTEDSIZESoD as follows. Let $x = 0^{n'-n-1} \circ 1 \circ x'$. We now feed x to our RESTRICTEDSIZESoD oracle to get a solution y , which we then output.

The reduction clearly runs in polynomial time since $n' \leq 2^{2^{\log_2(\log_2 n) + 2}} = 2^{4 \cdot \log_2(n)} = n^4$. Correctness follows from the definition of RESTRICTEDSIZESoD. $\square$

Lemma 7.3. If RESTRICTEDSIZESoD is traditionally d.s.r, then RESTRICTEDSIZESoD has a polynomial time algorithm.

Proof. Say that RESTRICTEDSIZESoD has a traditional d.s.r. We assume without loss of generality that this d.s.r only queries its oracle on inputs of size 2^{2^k} for some integer k since otherwise, we could simulate the answers to those queries as being 0 ourselves. Furthermore, we assume that the number of queries made by the d.s.r on an input of size 2^{2^k} is exactly $(2^{2^k})^c$. Furthermore, all other operations in the d.s.r (including the ones that move the tape-head and prepare it for the d.s.r call) require time exactly $(2^{2^k})^{c'}$ for some constant c .

Let us now consider running the recursive algorithm defined by the recursive algorithm for RESTRICTEDSIZESoD. Let $T(k)$ denote the time the recursive algorithm takes on an input of size 2^{2^k} . Notice that $T(0) = O(1)$.

$$T(k) = (2^{2^k})^c \cdot T(k-1) + (2^{2^k})^{c'}$$

Since $T(k-1) \geq 1$ and $c' < c$, we can say the following.

$$T(k) = 2 \cdot (2^{2^k})^c \cdot T(k-1)$$

Let $d = \max(2c, c') + 1$. We will show by induction that $T(k) \leq (2^{2^k})^d$. Notice that this clearly holds for $k = 0$.

Now for the inductive case.

$$\begin{aligned} T(k) &= (2^{2^k})^c \cdot T(k-1) + (2^{2^k})^{c'} \\ &\leq (2^{2^k})^c \cdot (2^{2^{k-1}})^d + (2^{2^k})^{c'} \\ &= (2^{2^k})^c \cdot (2^{2^k})^{\frac{d}{2}} + (2^{2^k})^{c'} \\ &= (2^{2^k})^{c+\frac{d}{2}} + (2^{2^k})^{c'} \end{aligned}$$

Note that by our choice of d , $c + d/2 \leq (d-1)/2 + d/2 = d - 1/2$ and $c' \leq d - 1$. Therefore, the following holds.

$$\begin{aligned} T(k) &\leq (2^{2^k})^{c+\frac{d}{2}} + (2^{2^k})^{c'} \\ &\leq (2^{2^k})^{d-\frac{1}{2}} + (2^{2^k})^{d-1} \\ &= (2^{2^k})^{d-1} \left[2^{2^{k-1}} + 1 \right] \\ &\leq (2^{2^k})^{d-1} \left[2^{2^k} \right] \\ &= (2^{2^k})^d \end{aligned}$$

The second to last inequality holds since $k \geq 1$. Therefore, the runtime of our recursive algorithm for RESTRICTEDSIZESOD on an input of length 2^{2^k} is at most $(2^{2^k})^d$ for some constant d . Therefore, our algorithm's running time is polynomial in its input length, as desired. $\square$

Theorem 7.4. *If every PLS-complete problem is traditionally d.s.r, then $\text{PLS} = \text{FP}$.*

Proof. If every PLS-complete problem is traditionally d.s.r, then RESTRICTEDSIZESOD is traditionally d.s.r (since RESTRICTEDSIZESOD is PLS-complete by [Lemma 7.1](#) and [Lemma 7.2](#)). This implies RESTRICTEDSIZESOD can be solved in polynomial time by [Lemma 7.3](#). However, since RESTRICTEDSIZESOD is PLS-complete, this implies $\text{PLS} = \text{FP}$. $\square$

We have therefore shown that it is unlikely that every PLS-complete problem is traditionally d.s.r. We leave open the question of whether every PLS-complete problem is μ -d.s.r.

Observation 7.5. *The exact same technique of defining a padded problem shows that not every NP-complete problem is traditionally d.s.r unless $\text{P} = \text{NP}$ and not every PSPACE-complete problem is traditionally d.s.r unless $\text{P} = \text{PSPACE}$.*

7.2 A note on random self-reducibility

We now turn our attention to a different question asked in [\[HMR22\]](#). We say a problem is random self-reducible (r.s.r) if it has a worst-case to average-case reduction. The notion of average-case

here is with respect to some efficiently samplable distribution. [HMR22] ask if every r.s.r TFNP problem is in some syntactic TFNP subclass (e.g. PPP).

An explicit construction problem is one where the input is a string 1^n and we are asked to generate some combinatorial object of length $\text{poly}(n)$ satisfying some property. A classic example is the problem of generating an n -bit prime larger than 2^n given as input 1^n .

We simply note here that all TFNP explicit construction problems can be reduced to an r.s.r problem. Consider a TFNP explicit construction A with verifier $V(1^n, \cdot)$. Consider the problem B where as input we are given a string from $\{0, 1\}^n$ and are asked to output a string y such that $V(1^n, y) = 1$. Note that A trivially reduces to B . Furthermore, B is trivially uniformly self-reducible since only its input length matters.

Therefore, if all TFNP r.s.r problems belonged to some TFNP subclass C , then all TFNP explicit construction problems would belong to C . This would be quite a surprising theorem and we view this as a weak barrier to a [Theorem 3.7](#) type theorem for random self-reducibility.

8 Acknowledgements

The authors would like to thank Alexander Golovnev, and Noah Stephens-Davidowitz for many helpful discussions and feedback on an earlier draft of this manuscript. The authors would also like to thank the anonymous referees for useful comments.

References

- [Ald83] David Aldous. Minimization algorithms and random walk on the d -cube. *The Annals of Probability*, 11(2):403–413, 1983. [11](#)
- [All10] Eric Allender. New surprises from self-reducibility. In *Programs, Proofs, Processes, Conference on Computability in Europe*, pages 1–5. Citeseer, 2010. [9](#)
- [BCH⁺22] Nir Bitansky, Arka Rai Choudhuri, Justin Holmgren, Chethan Kamath, Alex Lombardi, Omer Paneth, and Ron D Rothblum. Ppad is as hard as lwe and iterated squaring. In *Theory of Cryptography Conference*, pages 593–622. Springer, 2022. [10](#)
- [BGP00] Mihir Bellare, Oded Goldreich, and Erez Petrank. Uniform generation of np-witnesses using an np-oracle. *Inf. Comput.*, 163(2):510–526, December 2000. [8](#), [24](#)
- [BM08] Arnold Beckmann and Faron Moller. On the complexity of parity games. In *Visions of Computer Science-BCS International Academic Conference*. BCS Learning & Development, 2008. [9](#), [27](#), [28](#), [29](#)
- [Cai07] Jin-Yi Cai. $S2P \subseteq ZPP^{NP}$. *Journal of Computer and System Sciences*, 73(1):25–35, 2007. [6](#)
- [Can96] Ran Canetti. More on bpp and the polynomial-time hierarchy. *Information Processing Letters*, 57(5):237–241, 1996. [7](#)
- [CHLR23] Yeyuan Chen, Yizhi Huang, Jiayu Li, and Hanlin Ren. Range avoidance, remote point, and hard partial truth table via satisfying-pairs algorithms. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1058–1066, 2023. [5](#)

- [CHR24] Lijie Chen, Shuichi Hirahara, and Hanlin Ren. Symmetric exponential time requires near-maximum circuit size. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1990–1999, 2024. 2, 6, 7, 10
- [CL24] Yilei Chen and Jiatu Li. Hardness of range avoidance and remote point for restricted circuits via cryptography. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 620–629, 2024. 10
- [DGK⁺17] Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek, and Emo Welzl. AR-RIVAL: A zero-player graph game in $\text{NP} \cap \text{coNP}$. *A Journey Through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 367–374, 2017. 32
- [DQY11] Chuangyin Dang, Qi Qi, and Yinyu Ye. Computational models and complexities of tarski’s fixed points. Technical report, Technical report, 2011. 30
- [EPRY20] Kousha Etessami, Christos Papadimitriou, Aviad Rubinfeld, and Mihalis Yannakakis. Tarski’s theorem, supermodular games, and the complexity of equilibria. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, pages 18–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. 9, 26, 29, 30
- [FGMS20] John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *Journal of Computer and System Sciences*, 114:1–35, 2020. 9, 11, 13, 26
- [GGNS23] Karthik Gajulapalli, Alexander Golovnev, Satyajeet Nagargoje, and Sidhant Saraogi. Range avoidance for constant depth circuits: Hardness and algorithms. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2023. 5
- [GHH⁺18] Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubáček, Karel Král, Hagar Mosaad, and Veronika Slívová. Arrival: Next stop in cls. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018. 11, 26, 32, 33
- [GLV24] Karthik Gajulapalli, Zeyong Li, and Ilya Volkovich. Oblivious complexity classes revisited: Lower bounds and hierarchies. In *44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, page 25, 2024. 5, 10
- [GLW22] Venkatesan Guruswami, Xin Lyu, and Xiuhua Wang. Range avoidance for low-depth circuits and connections to pseudorandomness. *ACM Transactions on Computation Theory*, 2022. 5
- [HMR22] Prahladh Harsha, Daniel Mitropolsky, and Alon Rosen. Downward self-reducibility in tfnp. *arXiv preprint arXiv:2209.10509*, 2022. 1, 2, 3, 9, 10, 15, 20, 34, 36, 37
- [HV25] Edward A Hirsch and Ilya Volkovich. Upper and lower bounds for the linear ordering principle. *arXiv preprint arXiv:2503.19188*, 2025. 10
- [ILW23] Rahul Ilango, Jiatu Li, and R Ryan Williams. Indistinguishability obfuscation, range avoidance, and bounded arithmetic. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1076–1089, 2023. 10

- [JW24] Ce Jin and Hongxun Wu. A faster algorithm for pigeonhole equal sums. *arXiv preprint arXiv:2403.19117*, 2024. [10](#)
- [Kar17] Karthik C. S. Did the train reach its destination: The complexity of finding a witness. *Information Processing Letters*, 121:17–21, 2017. [33](#)
- [KKMP21] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos Papadimitriou. Total functions in the polynomial hierarchy. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021. [2](#), [4](#), [7](#), [8](#), [10](#), [11](#), [12](#), [19](#), [21](#)
- [Kor22a] Oliver Korten. Derandomization from time-space tradeoffs. In *37th Computational Complexity Conference (CCC 2022)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2022. [5](#)
- [Kor22b] Oliver Korten. The hardest explicit construction. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 433–444. IEEE, 2022. [2](#), [5](#), [10](#), [19](#)
- [KP24] Oliver Korten and Toniann Pitassi. Strong vs. weak range avoidance and the linear ordering principle. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1388–1407, 2024. [2](#), [4](#), [5](#), [6](#), [7](#), [18](#), [20](#), [21](#)
- [KVM99] Adam R Klivans and Dieter Van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 659–667, 1999. [7](#)
- [Lan53] Hyman Garshin Landau. On dominance relations and the structure of animal societies: Iii the condition for a score structure. *The bulletin of mathematical biophysics*, 15:143–148, 1953. [7](#), [8](#), [21](#)
- [Li24] Zeyong Li. Symmetric exponential time requires near-maximum circuit size: Simplified, truly uniform. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 2000–2007, 2024. [2](#), [6](#), [7](#), [10](#)
- [LY22] Jiayu Li and Tianqi Yang. $3.1n - o(n)$ circuit lower bounds for explicit functions. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1180–1193, 2022. [2](#)
- [MPS23] Nikhil S Mande, Manaswi Paraashar, and Nitin Saurabh. Randomized and quantum query complexities of finding a king in a tournament. *arXiv preprint arXiv:2308.02472*, 2023. [25](#)
- [PPY23] Amol Pasarkar, Christos Papadimitriou, and Mihalis Yannakakis. Extremal Combinatorics, Iterated Pigeonhole Arguments and Generalizations of PPP. In Yael Tsafran Kalai, editor, *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, volume 251 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 88:1–88:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. [11](#)

- [RS98] Alexander Russell and Ravi Sundaram. Symmetric alternation captures bpp. *computational complexity*, 7(2):152–162, 1998. 7
- [RSW22] Hanlin Ren, Rahul Santhanam, and Zhikun Wang. On the range avoidance problem for circuits. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 640–650. IEEE, 2022. 5
- [SSW03] Jian Shen, Li Sheng, and Jie Wu. Searching for sorted sequences of kings in tournaments. *SIAM Journal on Computing*, 32(5):1201–1209, 2003. 25
- [STW58] Hans Samelson, Robert M Thrall, and Oscar Wesler. A partition theorem for euclidean n-space. *Proceedings of the American Mathematical Society*, 9(5):805–807, 1958. 26
- [Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. 1955. 29
- [Zha24] Stan Zhang. Pigeonhole equal subset sum in $O^*(2^{n/3})$. Master’s thesis, Massachusetts Institute of Technology, 2024. 10