End of Potential Line

John Fearnley[1], Spencer Gordon[2], Ruta Mehta[3], and Rahul Savani[1]

[1]University of Liverpool, {john.fearnley, rahul.savani}@liverpool.ac.uk
[2]California Institute of Technology, slgordon@caltech.edu
[3]University of Illinois at Urbana-Champaign, rutamehta@cs.illinois.edu

April 19, 2018

Abstract

We introduce the problem EndOfPotentialLine and the corresponding complexity class EOPL of all problems that can be reduced to it in polynomial time. This class captures problems that admit a single combinatorial proof of their joint membership in the complexity classes PPAD of fixpoint problems and PLS of local search problems. EOPL is a combinatorially-defined alternative to the class CLS (for Continuous Local Search), which was introduced in [16] with the goal of capturing the complexity of some well-known problems in PPAD $\cap$ PLS that have resisted, in some cases for decades, attempts to put them in polynomial time. Two of these are Contraction, the problem of finding a fixpoint of a contraction map, and P-LCP, the problem of solving a P-matrix Linear Complementarity Problem.

We show that EndOfPotentialLine is in CLS via a two-way reduction to EndOfMeteredLine. The latter was defined in [36] to show query and cryptographic lower bounds for CLS. Our two main results are to show that both PL-Contraction (Piecewise-Linear Contraction) and P-LCP are in EOPL. Our reductions imply that the promise versions of PL-Contraction and P-LCP are in the promise class UniqueEOPL, which corresponds to the case of a single potential line. This also shows that simple-stochastic, discounted, mean-payoff, and parity games are in EOPL.

Using the insights from our reduction for PL-Contraction, we obtain the first polynomial-time algorithms for finding fixed points of contraction maps in fixed dimension for any ℓ_p norm, where previously such algorithms were only known for the ℓ_2 and ℓ_∞ norms. Our reduction from P-LCP to EndOfPotentialLine allows a technique of Aldous to be applied, which in turn gives the fastest-known randomized algorithm for P-LCP.

Contents

1 Introduction

Total function problems in NP. The complexity class TFNP contains search problems that are guaranteed to have a solution, and whose solutions can be verified in polynomial time [47]. While it is a semantically defined complexity class and thus unlikely to contain complete problems, a number of syntactically defined subclasses of TFNP have proven very successful at capturing the complexity of total search problems. In this paper, we focus on two in particular, PPAD and PLS. The class PPAD was introduced in [53] to capture the difficulty of problems that are guaranteed total by a parity argument. It has attracted intense attention in the past decade, culminating in a series of papers showing that the problem of computing a Nash-equilibrium in two-player games is PPAD-complete [10,15], and more recently a conditional lower bound that rules out a PTAS for the problem [55]. No polynomial-time algorithms for PPAD-complete problems are known, and recent work suggests that no such algorithms are likely to exist [3,29]. PLS is the class of problems that can be solved by local search algorithms (in perhaps exponentially-many steps). It has also attracted much interest since it was introduced in [37], and looks similarly unlikely to have polynomial-time algorithms. Examples of problems that are complete for PLS include the problem of computing a pure Nash equilibrium in a congestion game [22], a locally optimal max cut [56], or a stable outcome in a hedonic game [27].

Continuous Local Search. If a problem lies in both PPAD and PLS then it is unlikely to be complete for either class, since this would imply an extremely surprising containment of one class in the other. In their 2011 paper [16], Daskalakis and Papadimitriou observed that there are several prominent total function problems in $\mathsf{PPAD} \cap \mathsf{PLS}$ for which researchers have not been able to design polynomial-time algorithms. Motivated by this they introduced the class CLS, a syntactically defined subclass of $\mathsf{PPAD} \cap \mathsf{PLS}$. CLS is intended to capture the class of optimization problems over a continuous domain in which a continuous potential function is being minimized and the optimization algorithm has access to a polynomial-time continuous improvement function. They showed that many classical problems of unknown complexity are in CLS, including the problem of solving a simple stochastic game, the more general problems of solving a Linear Complementarity Problem with a P-matrix, and finding an approximate fixpoint to a contraction map. Moreover, CLS is the smallest known subclass of TFNP not known to be in P, and hardness results for it imply hardness results for PPAD and PLS simultaneously.

Recent work by Hubáček and Yogev [36] proved lower bounds for CLS. They introduced a problem known as EndOfMeteredLine which they showed was in CLS, and for which they proved a query complexity lower bound of $\Omega(2^{n/2}/\sqrt{n})$ and hardness under the assumption that there were one-way permutations and indistinguishability obfuscators for problems in $\mathsf{P}_{/\mathsf{poly}}$. Another recent result showed that the search version of the Colorful Carathéodory Theorem is in $\mathsf{PPAD} \cap \mathsf{PLS}$, and left open whether the problem is also in CLS [49].

Until recently, it was not known whether there was a natural CLS-complete problem . In their original paper, Daskalakis and Papadimitriou suggested two natural candidates for CLS-complete problems, ContractionMap and P-LCP, which we study in this paper. Recently, two variants of ContractionMap have been shown to be CLS-complete. Whereas in the original definition of ContractionMap it is assumed that an ℓ_p or ℓ_∞ norm is fixed, and the contraction property is measured with respect to the metric induced by this fixed norm, in these two new complete variants, a metric [17] and meta-metric [23] are given as input to the problem[1].

Our contribution. Our main conceptual contribution is to define the problem EndOfPoten-

[1] The result for meta-metrics appeared in an unpublished earlier version of this paper [23]. We include it Section E with a comparison to [17]. This current version of our paper has new results and a corresponding different title.

tialLine, which unifies in an extremely natural way the circuit-based views of PPAD and of PLS. The canonical PPAD-complete problem is EndOfLine, a problem that provides us with an exponentially large graph consisting of lines and cycles, and asks us to find the end of one of the lines. The canonical PLS-complete problem provides us with an exponentially large DAG, whose acyclicity is guaranteed by the existence of a *potential function* that increases along each edge. The problem EndOfPotentialLine is an instance of EndOfLine that *also* has a potential function that increases along each edge.

We define the corresponding complexity class EOPL, consisting of all problems that can be reduced to EndOfPotentialLine. This class captures problems that admit a *single* combinatorial proof of their joint membership in the classes PPAD of fixpoint problems and PLS of local search problems, and is a combinatorially-defined alternative to the class CLS.

The problem EndOfMeteredLine was introduced in [36] to show query and cryptographic lower bounds for CLS. We show that EndOfPotentialLine is contained in CLS via a two-way reduction to EndOfMeteredLine, and in doing so, we show that EOPL is the closest class to P among the syntactic sub-classes of TFNP. EndOfMeteredLine captures problems that have a PPAD directed path structure where it is possible to keep count of *exactly* how far each vertex is from the start of the path. In a sense, this may seem rather unnatural, as many common problems do not seem to have this property. However, our reduction from EndOfPotentialLine to EndOfMeteredLine shows that the latter is more general than it may at first seem. The reduction from EndOfMeteredLine to EOPL implies that the query and cryptographic lower bounds for CLS also hold for EOPL.

For our two main technical results, we show that two problems with long-standing unresolved complexity belong to EOPL. These problems are:

- The P-matrix Linear Complementarity Problem (P-LCP). Designing a polynomial-time algorithm for this problem has been open for decades, at least since the 1978 paper of Murty [51] that provided exponential-time examples for *Lemke's algorithm* [44] for P-matrix LCPs.

- Finding a fixpoint of a Piecewise-Linear Contraction Map (PL-Contraction). The problem of finding a fixpoint of a LinearFIXP circuit with purported Lipschitz constant c is complete for PPAD. In PL-Contraction we are asked to solve this problem when c is strictly less than 1. A contraction map on a continuous bounded domain has a *unique* fixpoint and captures many important problems including discounted games and simple stochastic games.

We summarize our complexity-theoretic results in Figure 1. Our reductions imply that the promise versions of PL-Contraction and P-LCP are in the promise class UniqueEOPL, which corresponds to instances of EndOfPotentialLine that have a single line. Other well-known problems also lie in UniqueEOPL: Our two reductions that put P-LCP and PL-Contraction in EOPL both independently also show that simple stochastic, discounted, mean-payoff, and parity games are all in UniqueEOPL (unique because these games are *guaranteed* to produce contraction maps and P-matrix LCPs). Recently several papers have studied the problem ARRIVAL, which is a reachability problem for a switching system. ARRIVAL was introduced in [19] and shown to be in $\mathsf{NP} \cap \mathsf{coNP}$, then in PLS [41], and then most recently in CLS [30]. In fact, the latter proof also shows that ARRIVAL belongs to UniqueEOPL. In addition, we reduce the non-promise version of P-LCP, the problem of solving the LCP or returning a violation of the P-matrix property, to (non-unique) EOPL.

Our final contributions are algorithmic and arise from the structural insights provided by our two main reductions. Using the ideas from our reduction for PL-Contraction, we obtain the first polynomial-time algorithms for finding fixpoints of contraction maps in fixed dimension for any ℓ_p

norm, where previously such algorithms were only known for ℓ_2 and ℓ_∞ norms. We also show that these results can be extended to the case where the contraction map is given by a general arithmetic circuit. As noted in [30], our reduction from P-LCP to EndOfPotentialLine allows a technique of Aldous to be applied, which in turn gives the fastest known randomized algorithm for P-LCP.

Related work. Papadimitriou showed that P-LCP, the problem of solving the LCP or returning a violation of the P-matrix property, is in PPAD [53] using Lemke's algorithm. The relationship between Lemke's algorithm and PPAD has been studied by Adler and Verma [1]. Later, Daskalakis and Papadimitrou showed that P-LCP is in CLS [16], using the potential reduction method in [43]. Many algorithms for P-LCP have been studied, e.g., [42, 50, 51]. However, no polynomial-time algorithms are known for P-LCP, or for the promise version where one can assume that the input matrix is a P-matrix. This promise version is motivated by the reduction from Simple Stochastic Games to P-matrix LCPs.

The best known algorithms for P-LCP are based on a reduction to Unique Sink Orientations (USOs) of cubes [60]. For an P-matrix LCP of size n, the USO algorithms of [61] apply, and give a deterministic algorithm that runs in time $O(1.61^n)$ and a randomized algorithm with expected running time $O(1.43^n)$. The application of Aldous' algorithm to the EndOfPotentialLine instance that we produce from a P-matrix LCP takes expected time $2^{n/2} \cdot \text{poly}(n) = O(1.4143^n)$ in the worst case.

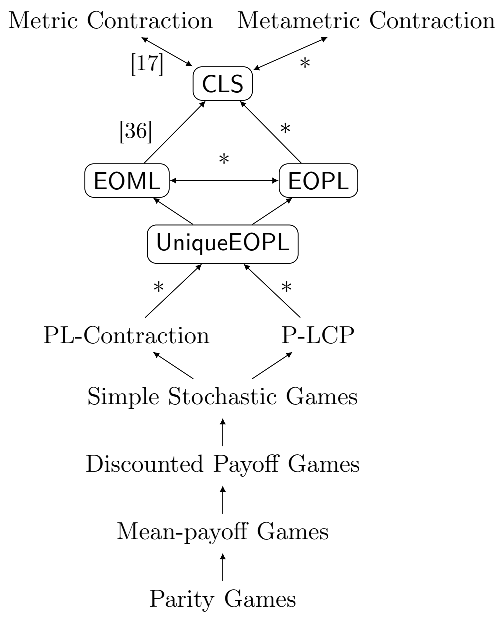


Figure 1: The relationship of EOPL to other classes and problems. Results from this paper are shown as stars.

Simple Stochastic Games are related to Parity games, which are an extensively studied class of two-player zero-sum infinite games that capture important problems in formal verification and logic [20]. There is a sequence of polynomial-time reductions from parity games to mean-payoff games to discounted games to simple stochastic games [31, 32, 40, 54, 63]. The complexity of solving these problems is unresolved and has received much attention over many years (see, for example, [4, 12, 25, 26, 38, 63]). In a recent breakthrough [6], a quasi-polynomial time algorithm for parity games have been devised, and there are now several algorithms with this running time [6, 24, 39]. For mean-payoff, discounted, and simple stochastic games, the best-known algorithms run in randomized subexponential time [45]. The existence of a polynomial time algorithm for solving any of these games would be a major breakthrough. Simple stochastic games can also be reduced in polynomial time to PL-Contraction with the ℓ_∞ norm [21].

The problem of computing a fixpoint of a continuous map $f : \mathcal{D} \mapsto \mathcal{D}$ with Lipschitz constant c has been extensively studied, in both continuous and discrete variants [8, 9, 18]. For arbitrary maps with $c > 1$, exponential bounds on the query complexity of computing fixpoints are known [7, 33]. In [5, 35, 59], algorithms for computing fixpoints for specialized maps such as weakly ($c = 1$) or strictly ($c < 1$) contracting maps are studied. For both cases, algorithms are known for the case of ℓ_2 and ℓ_∞ norms, both for absolute approximation ($||x - x^*|| \leq \epsilon$ where x^* is an exact fixpoint) and relative approximation ($||x - f(x)|| \leq \epsilon$). A number of algorithms are known for the ℓ_2 norm handling both types of approximation [34, 52, 58]. There is an exponential lower bound for absolute approximation with $c = 1$ [58]. For relative approximation and a domain of dimension d, an

$O(d \cdot \log 1/\epsilon)$ time algorithm is known [34]. For absolute approximation with $c < 1$, an ellipsoid-based algorithm with time complexity $O(d \cdot [\log(1/\epsilon) + \log(1/(1-c))])$ is known [34]. For the ℓ_∞ norm, [57] gave an algorithm to find an ϵ-relative approximation in time $O(\log(1/\epsilon)^d)$ which is polynomial for constant d. In summary, for the ℓ_2 norm polynomial-time algorithms are known for strictly contracting maps; for the ℓ_∞ norm algorithms that are polynomial time for constant dimension are known. For arbitrary ℓ_p norms, to the best of our knowledge, no polynomial-time algorithms for constant dimension were known before this paper.

2 EndOfPotentialLine

In this section, we define a new problem EndOfPotentialLine, and we show that this problem is polynomial-time equivalent to EndOfMeteredLine. First we recall the definition of EndOfMeteredLine, which was first defined in [36]. It is close in spirit to the problem EndOfLine that is used to define PPAD [53].

Definition 1 (EndOfMeteredLine [36])**.** *Given circuits $S, P : \{0,1\}^n \to \{0,1\}^n$, and $V : \{0,1\}^n \to \{0,\dots,2^n\}$ such that $P(0^n) = 0^n \neq S(0^n)$ and $V(0^n) = 1$, find a string $\boldsymbol{x} \in \{0,1\}^n$ satisfying one of the following*

(T1) either $S(P(\boldsymbol{x})) \neq \boldsymbol{x} \neq 0^n$ or $P(S(\boldsymbol{x})) \neq \boldsymbol{x}$,

(T2) $\boldsymbol{x} \neq 0^n, V(\boldsymbol{x}) = 1$,

(T3) either $V(\boldsymbol{x}) > 0$ and $V(S(\boldsymbol{x})) - V(\boldsymbol{x}) \neq 1$, or $V(\boldsymbol{x}) > 1$ and $V(\boldsymbol{x}) - V(P(\boldsymbol{x})) \neq 1$.

Intuitively, an EndOfMeteredLine is an EndOfLine instance that is also equipped with an "odometer" function. The circuits P and S implicitly define an exponentially large graph in which each vertex has degree at most 2, just as in EndOfLine, and condition T1 says that the end of every line (other than 0^n) is a solution. In particular, the string 0^n is guaranteed to be the end of a line, and so a solution can be found by following the line that starts at 0^n. The function V is intended to help with this, by giving the number of steps that a given string is from the start of the line. We have that $V(0^n) = 1$, and that V increases by exactly 1 for each step we make along the line. Conditions T2 and T3 enforce this by saying that any violation of the property is also a solution to the problem.

In EndOfMeteredLine, the requirement of incrementing V by exactly one as we walk along the line is quite restrictive. We define a new problem, EndOfPotentialLine, which is similar in spirit to EndOfMeteredLine. The key difference is that while V is still required to be strictly monotonically increasing along the line, the amount that it increases in each step can be any amount.

Definition 2 (EndOfPotentialLine)**.** *Given Boolean circuits $S, P : \{0,1\}^n \to \{0,1\}^n$ such that $P(0^n) = 0^n \neq S(0^n)$ and a Boolean circuit $V : \{0,1\}^n \to \{0,1,\dots,2^m-1\}$ such that $V(0^n) = 0$ find one of the following:*

(R1) A point $x \in \{0,1\}^n$ such that $S(P(x)) \neq x \neq 0^n$ or $P(S(x)) \neq x$.

(R2) A point $x \in \{0,1\}^n$ such that $x \neq S(x)$, $P(S(x)) = x$, and $V(S(x)) - V(x) \leq 0$.

We define the complexity class EOPL as all problems that can be reduced to EndOfPotentialLine in polynomial time. We also define a natural variant of EOPL, called UniqueEOPL which has a *single* potential line. Since there is no efficient way to verify this, UniqueEOPL is a promise class. We will show that the promise versions of PL-Contraction and P-LCP are in UniqueEOPL.

Aldous' algorithm for problems in EOPL. In [30], the problem ARRIVAL of deciding reachability for a directed switching system was reduced to EndOfMeteredLine, and it was noted that a randomized algorithm by Aldous [2] provides the best-known algorithm for ARRIVAL. Aldous' algorithm is actually very simple: it randomly samples a large number of candidate solutions and then performs a local search from the best sampled solution. Aldous' algorithm can actually solve any problem in EOPL.

EndOfMeteredLine to EndOfPotentialLine and back. As one might expect, the reduction to EndOfPotentialLine is relatively easy, but we need to be careful about handling the solutions that correspond to violations. Full details of the reduction with proofs are in Appendix A.1.

The reduction to EndOfMeteredLine is involved. The basic idea is that, when the potential value jumps by more than 1, we introduce a sequence of new vertices. We embed the potential into the vertex space, so that if there is an edge from $\boldsymbol{u}$ to $\boldsymbol{u}'$ in the EndOfPotentialLine instance whose respective potentials are p and p' such that $p < p'$ then we create edges $(u, p) \to (u, p+1) \to \ldots \to (u, p'-1) \to (u, p')$. By increasing the vertex space in this way, we create many vertices that are not on any line. To ensure that these vertices are not solutions we turn them into self-loops. The remaining details, including how we deal with solutions that correspond to violations, are given in Appendix A.2. We obtain the following theorem.

Theorem 3. EndOfMeteredLine *and* EndOfPotentialLine *are polynomial-time equivalent.*

We note that our reduction implies that UniqueEOML = UniqueEOPL.

3 The P-matrix Linear Complementarity Problem

In this section, we reduce P-LCP to EndOfPotentialLine. Our reduction relies heavily on the application of Lemke's algorithm to P-matrix LCPs. Let $[n]$ denote the set $\{1, \ldots, n\}$.

Definition 4 (LCP $(M, \boldsymbol{q})$). *Given a matrix $M \in \mathbb{R}^{d \times d}$ and vector $\boldsymbol{q} \in \mathbb{R}^{d \times 1}$, find a $\mathbf{y} \in \mathbb{R}^{d \times 1}$ s.t.:*

$$M\mathbf{y} \leq \boldsymbol{q}; \quad \mathbf{y} \geq 0; \quad y_i(\boldsymbol{q} - M\mathbf{y})_i = 0, \ \forall i \in [n]. \tag{1}$$

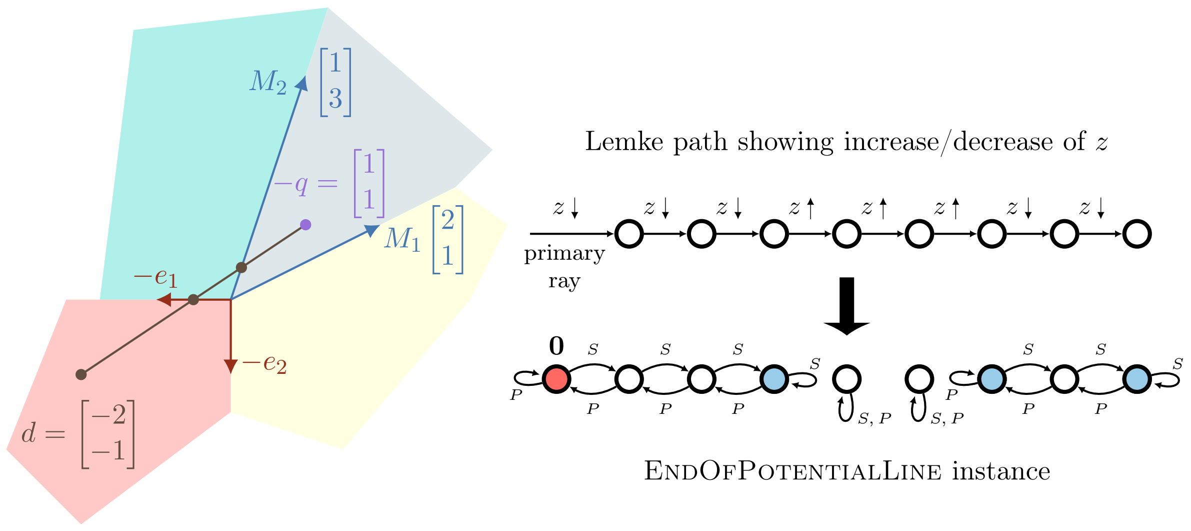
In general, deciding whether an LCP has a solution is NP-complete [11], but if M is a P-matrix, as defined next, then the LCP $(M, \boldsymbol{q})$ has a *unique* solution for all $\boldsymbol{q} \in \mathbb{R}^{d \times 1}$.

Definition 5 (P-matrix). *$M \in \mathbb{R}^{d \times d}$ is called a P-matrix if every principle minor of M is positive.*

Checking if a matrix is a P-matrix is coNP-complete [14]. Thus, Megiddo [46, 47] defined the problem P-LCP, which avoids the need for a promise about M.

Definition 6 (P-LCP and Promise-P-LCP). *Given an LCP $(M, \boldsymbol{q})$, find either: (Q1) $\mathbf{y} \in \mathbb{R}^{n \times 1}$ that satisfies (1); or (Q2) a non-positive principal minor of M. For* Promise-P-LCP*, we are promised that M is a P-matrix and thus only seek a solution of type Q1.*

Overview of P-LCP to EndOfPotentialLine. Our reduction is based on Lemke's algorithm, explained in detail in Section B.1. Lemke's algorithm introduces an extra variable z and an extra column d, called a covering vector, and follows a path along edges of the new LCP polyhedron based on a complementary pivot rule that maintains an almost-complementary solution. Geometrically, solving an LCP is equivalent to finding a *complementary cone*, corresponding to a subset of columns of M and the complementary unit vectors, that contains $-q$. This is depicted on the left in Figure 2, which also shows Lemke's algorithm as inverting a piecewise linear map



M_2 $\begin{bmatrix}1\\3\end{bmatrix}$
$-q = \begin{bmatrix}1\\1\end{bmatrix}$
M_1 $\begin{bmatrix}2\\1\end{bmatrix}$
$-e_1$
$-e_2$
$d = \begin{bmatrix}-2\\-1\end{bmatrix}$
Lemke path showing increase/decrease of z
$z\downarrow$ $z\downarrow$ $z\downarrow$ $z\uparrow$ $z\uparrow$ $z\uparrow$ $z\downarrow$ $z\downarrow$
primary ray
0
S
P
S, P
EndOfPotentialLine instance

Figure 2: Left: geometric view of Lemke's algorithm as inverting a piecewise linear map. Right: construction of S and P for EndOfPotentialLine instance from the Lemke path, where the arrows on its edges indicate whether z increases or decreases along the edge.

along the line from d to $-q$[2]. The algorithm pivots between the brown vertices at the intersections of complementary cones and terminates at $-q$. The extra variable z can be normalized and then describes how far along the line from d to $-q$ we are. P-matrix LCPs are exactly those where the complementary cones cover the whole space with no overlap, and then z decreases monotonically as Lemke's algorithm proceeds.

The vertices along the Lemke path correspond to a subset of $[n]$ of basic variables. These subsets correspond to the bit strings describing vertices in our EndOfPotentialLine instance. We use the following key properties of Lemke's algorithm as applied to a P-matrix:

1. If Lemke's algorithm does not terminate with a LCP solution Q1, it provides a Q2 solution.
2. For a P-matrix, the extra variable z strictly decreases in each step of the algorithm.
3. Given a subset of $[n]$, we can efficiently decide if it corresponds to a vertex on a Lemke path.
4. By a result of Todd [62, Section 5], the Lemke path can be locally oriented.

The starting point of the EndOfPotentialLine instance corresponds to d. We will then follow the line given to us by Lemke's algorithm, and we will use z as the potential function of this line. If we start with a P-LCP instance where M is actually a P-matrix then this reduction will produce a single line from d to the solution of the P-LCP, and z will monotonically decrease along this line. The main difficulty of the reduction is dealing with the case where M is not a P-matrix. This may cause Lemke's algorithm to terminate without an LCP solution. Another issue is that, even when Lemke's algorithm does find a solution, z may not decrease monotonically along the line.

In the former case, the first property above gives us a Q2 solution for the P-LCP problem. In the latter case, we define any point on the line where z increases to be a self-loop, breaking the line at these points. Figure 2 shows an example, where the two vertices at which z increases are turned into self loops, thereby introducing two new solutions before and after the break. Both of these solutions give us a Q2 solution for the P-LCP instance. The full details of the reduction

[2]Our figures should ideally be viewed in color.

are involved and appear in Appendix B.2. It is worth noting that, in the case where the input is actually a P-matrix, the resulting EndOfPotentialLine instance has a unique line. So, we have the following.

Theorem 7. P-LCP *is in* EOPL. Promise-P-LCP *is in* UniqueEOPL*.*

Our reduction produces, for an LCP defined by an $n \times n$ matrix M, an EOPL instance with $O(2^n)$ vertices. Thus, Aldous' algorithm can be applied to give the following corollary.

Corollary 8. *There is a randomized algorithm for* P-LCP *that runs in expected time* $O(1.4143^n)$*.*

4 Piecewise Linear Contraction Maps

We study contraction maps where f is given as a LinearFIXP *circuit*, which is an arithmetic circuit comprised of $\max, \min, +, -$, and $\times\zeta$ (multiplication by a constant) gates [21]. Hence, a LinearFIXP circuit defines a *piecewise linear* function.

Definition 9 (PL-Contraction)**.** *The input is a* LinearFIXP *circuit computing* $f : [0,1]^d \to [0,1]^d$*, a constant* $c \in (0,1)$*, and a* $p \in \mathbb{N} \cup \{\infty\}$*. It is promised that* f *is a contraction map in the* ℓ_p *norm with contraction factor* c*. We are required to find the* $x \in [0,1]^d$ *such that* $f(x) = x$*.*

PL-Contraction is the *promise* version of contraction. We reduce it to EndOfPotentialLine. In the case where the promise is not satisfied (i.e., f is not actually contracting), our reduction will detect this fact. However, this does not allow us to reduce from the non-promise version, since in the case where the EndOfPotentialLine solution is not a fixpoint, we do not have an easy way to produce two points that violate contraction, even though two such points must exist.

UFEOPL. The first step in our reduction will be to reduce to a problem that we call UniqueForwardEOPL. This is a version of EndOfPotentialLine with two changes. Firstly, it is guaranteed (via a promise) that there is a unique line. Secondly, the vertices on this line are only equipped with circuits that give the next point on the line, and unlike EOPL, there is no circuit giving the predecessor. Otherwise, the problem is unchanged. So the goal is still to find a vertex on the line where the potential increases, or the end of the line. Formally, we define the problem as follows.

Definition 10 (UniqueForwardEOPL)**.** *The input is a tuple* (C, S, V) *of Boolean circuits* $C : \{0,1\}^n \to \{0,1\}$*,* $S : \{0,1\}^n \to \{0,1\}^n$*, and* $V : \{0,1\}^n \to \{0,1,\ldots,2^m - 1\}$*, with the property that* $C(0^n) = 1$ *and* $V(0^n) = 0$*. It is promised that, for every vertex* $x \in \{0,1\}^n$ *such that* $C(x) = 1$*, there exists a* k *such that* $x = S^k(0^n)$*. We must output one of the following.*

(U1) A vertex $x \in \{0,1\}^n$ *such that* $C(x) = 1$ *and* $V(S(x)) - V(x) \leq 0$*.*

(U2) A vertex $x \in \{0,1\}^n$ *such that* $C(x) = 1$ *and* $C(S(x)) = 0$*.*

The circuits S and V give the successor and potential for each vertex on the line, as usual. The extra circuit C is used to determine whether a vertex is on the line or not. This was not necessary for EOPL, because the successor and predecessor circuits could be used to check this. Observe that the promise guarantees that every vertex on the line can be reached from the start point of the line, so there is indeed only one line whenever the promise holds.

Discretizing the problem. We discretize the space $[0,1]^d$ by defining a grid of points. This grid

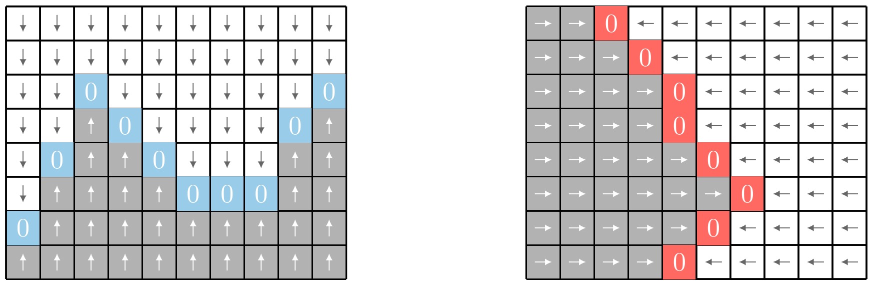


Figure 3: Left: A direction function for the up/down dimension. Right: A direction function for the left/right dimension.

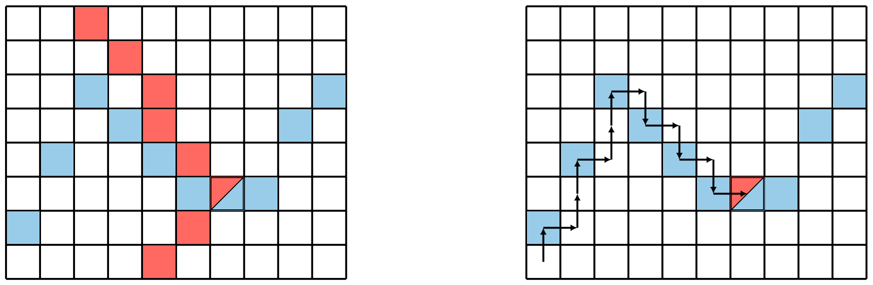


Figure 4: Left: The red and blue surfaces. Right: the path that we follow.

will be represented by the set P and each element of P will be a tuple $p = (p_1, p_2, \dots, p_d)$, where each p_i is a rational whose denominator is polynomially sized.

We also discretize the contraction map f itself in the following way. For each dimension d we define a *direction* function $D_i : P \to \{\mathsf{up}, \mathsf{down}, \mathsf{zero}\}$, where for each point $p \in P$, we have: if $f(p)_i > p$ then $D_i(p) = \mathsf{up}$, if $f(p)_i < p$ then $D_i(p) = \mathsf{down}$, and if $f(p)_i = p$ then $D_i(p) = \mathsf{zero}$. In other words, the function D_i simply outputs whether $f(p)$ moves up, down, or not at all in dimension i. So a fixpoint of f is a point $p \in P$ such that $D_i(p) = \mathsf{zero}$ for all i.

We define the Discrete-Contraction problem to be the problem of finding such a point, and we show that PL-Contraction can be reduced in polynomial time to Discrete-Contraction. The key part of this proof is that, since the input to PL-Contraction is defined by a LinearFIXP circuit, we are able to find a suitably small grid such that there will actually exist a point $p \in P$ with $D_i(p) = \mathsf{zero}$ for all i. The details of this are deferred to Appendix C.2.

A two-dimensional example. To illustrate our construction, we first give an example in two dimensions. Figure 3 gives two direction functions for a two-dimensional problem. The figure on the left shows a direction function for the up-down dimension, which we will call dimension 1 and illustrate using the color blue. The figure on the right shows a direction function for the left-right dimension, which we will call dimension 2 and illustrate using the color red. Each square in the figures represents a point in the discretized space, and the value of the direction function is shown inside the box.

On the left side in Figure 4, we have overlaid the *surfaces* of these two functions. The surface of a direction function D_i is exactly the set of points $p \in P$ such that $D_i(p) = \mathsf{zero}$. The fixpoint p that we seek has $D_i(p) = \mathsf{zero}$ for all dimensions i, and so it lies at the intersection of these surfaces.

To reach this intersection, we will walk along the blue surface until we reach the point that is on the red and blue surface. The path starts at the bottom left-hand corner of the diagram, which corresponds to the point $(0, 0)$. It initially finds the blue surface by following the blue direction function D_1, which creates a path that walks up until the blue surface is found.

Once we have found the point on the surface we then move one step in dimension 2. To do this, we follow the direction given by the red direction function D_2, and so in our example, we move one step to the right. Once we have done so, we then have to find the blue surface again, which again involves following the direction of D_1. By repeatedly finding the blue surface, and the following the direction given by D_2, we eventually arrive at the point that is on the red and blue surfaces, which is the fixpoint that we seek. The line that we follow is shown in the right-hand side of Figure 4.

We need two properties to make this approach work. Firstly, we must ensure that every *slice* of dimension 2 has a unique point in the blue surface, that is, if we fix a coordinate y in dimension 2, then there is exactly one point (x, y) such that $D_1(x, y) = \mathsf{zero}$. Secondly, it must be the case that at any point p on the blue surface, the function $D_2(p)$ tells us the correct direction to walk to find the fixpoint. We ensure that our DISCRETE-CONTRACTION instance satisfies these properties, and the details are given in Appendix C.2.

The potential. How do we define a potential for this line? Observe that the dimension-two coordinates of the points on the line are weakly monotone, meaning that the line never moves to the left. Furthermore, for any dimension-two slice (meaning any slice in which the left/right coordinate is fixed), the line is either monotonically increasing, or monotonically decreasing, depending on the direction specified by D_1. So, if k denotes the maximum coordinate in either dimension, then the function

$$V(p_1, p_2) = \begin{cases} k \cdot p_2 + p_1 & \text{if } D_1(p_1, p_2) = \mathsf{up}, \\ k \cdot p_2 + (k - p_1) & \text{if } D_1(p_1, p_2) = \mathsf{down}. \end{cases}$$

is a function that monotonically increases along the line.

Uniqueness. Finally, we must argue that the line is unique. Here we must carefully define what exactly a vertex on the line is, so that the circuit C can correctly identify the points that are on the line. In particular, when we are walking towards the blue surface, we must make sure that we are either coming from the point $(0, 0)$ at the start of the line, or that we have just visited a point on the blue surface. For example, consider the third column of points from the left in Figure 4. The first three points from the bottom of the diagram should be identified as not being on the line, while the second three points should be identified as being on the line. To do this, we make the points on our line tuples (q, p), where p is the current point on the line, and q is either

- the symbol $-$, indicating that we are the start of the line and p_2 should be equal to 0, or
- a point q that is on the blue surface, and satisfies $q_2 = p_2 - 1$.

In the latter case, the point q is the last point on the blue surface that was visited by the line, and it can be used to determine whether p is a valid point. In our example in the third column, q would be the point on the blue surface in column 2, and so the three points below q in the third column can be determined to not lie on the line, since their direction function points upwards, and thus no line that visited q can have arrived at these points. Meanwhile, the points above q do lie on the line, for exactly the same reason.

Once the line arrives at the next point on the blue surface, the point q is then overwritten with this new point. This step is the reason why only a successor circuit can be given for the line, since the value that is overwritten cannot easily be computed by a predecessor circuit.

The full reduction. Our reduction from DISCRETE-CONTRACTION to UNIQUEFORWARDEOPL generalizes the approach given above to d dimensions. We say that a point $p \in P$ is on the *i-surface* if $D_j(p) = \mathsf{zero}$ for all $j \leq i$. In our two-dimensional example we followed a line of points on the one-surface, in order to find a point on the two-surface. In between any two points on the one-surface, we followed a line of points on the zero-surface (every point is trivially on the zero-surface).

Our line will visit a sequence of points on the $(d-1)$-surface in order to find the point on the d-surface, which is the fixpoint. Between any two points on the $(d-1)$-surface the line visits a sequence of points on the $(d-2)$-surface, between any two points on the $(d-2)$-surface the line visits a sequence of points on the $(d-3)$-surface, and so on.

These points will satisfy a *recursive monotonicity* property, which is a generalization of the property that we saw in two-dimensions. The sequence of points on the $(d-1)$-surface are monotonically increasing in the dth coordinate. Between every pair of points on the $(d-1)$-surface, the sequence of points on the $(d-2)$-surface will be either monotonically increasing or monotonically decreasing in the $(d-1)$th coordinate, and so on. This will allow us to define a potential function that generalizes the one that we saw in the two-dimensional case.

Full details of this construction are given in Appendix C.3, where the following lemma is proved.

Lemma 11. Discrete-Contraction *can be reduced in polynomial time to* UniqueForwardEOPL.

From UniqueForwardEOPL to EndOfPotentialLine. The final step of the proof is to reduce UniqueForwardEOPL to EndOfPotentialLine. Fortunately, we are able to utilize prior work to perform this step of the reduction. We will utilize the following problem that was introduced by Bitansky et al [3].

Definition 12 (SinkOfVerifiableLine [3])**.** *The input to the problem consists of a starting vertex $x_s \in \{0,1\}^n$, a target integer $T \leq 2^n$, and two boolean circuits $S : \{0,1\}^n \to \{0,1\}^n$, $W : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. It is promised that, for every vertex $x \in \{0,1\}^n$, and every integer $i \leq T$, we have $W(x,t) = 1$ if and only if $x = S^{i-1}(x_s)$. The goal is to find the vertex $x_f \in \{0,1\}^n$ such that $W(x_f, T) = 1$.*

It was shown by Hubáček and Yogev [36] that SinkOfVerifiableLine can be reduced in polynomial time to EndOfMeteredLine, and hence also to EndOfPotentialLine. So, to complete our proof, we need to reduce UniqueForwardEOPL to SinkOfVerifiableLine. We are able to do this because we are guaranteed that the line in our instance is unique, and so we can use the potential of that line to determine how far we are from the start of the line. However, this does not directly give us the exact number of steps between the start of the line, which we need to construct the circuit W. So, we first reduce to a unique forward version of EndOfMeteredLine, using the same technique as we used in Theorem 3, and then reduce that to SinkOfVerifiableLine. The full details of the proof can be found in Appendix C.4.

Lemma 13. UniqueForwardEOPL *can be reduced in polynomial time to* UniqueEOPL.

To summarise, we obtain the following theorem.

Theorem 14. PL-Contraction *is in* UniqueEOPL.

5 Algorithms for Contraction Maps

An algorithm for PL-Contraction. The properties that we observed in our reduction from PL-Contraction to EndOfPotentialLine can also be used to give polynomial time algorithms for the case where the number of dimensions is constant. In our two-dimensional example, we relied on the fact that each dimension-two slice has a unique point on the blue surface, and that the direction function at this point tells us the direction of the overall fixpoint.

This suggests that a nested binary search approach can be used to find the fixpoint. The outer binary search will work on dimension-two coordinates, and the inner binary search will work on dimension-one coordinates. For each fixed dimension-two coordinate y, we can apply the inner binary search to find the unique point (x, y) that is on the blue surface. Once we have done so, $D_2(x, y)$ tells us how to update the outer binary search to find a new candidate coordinate y'.

This can be generalized to d-dimensional instances, by running d nested instances of binary search. Doing so yields the following theorem, whose proof appears in Appendix D.3.

Theorem 15. *Given a* LinearFIXP *circuit C encoding a contraction map $f : [0, 1]^d \to [0, 1]^d$ with respect to any ℓ_p norm, there is an algorithm to find a fixpoint of f in time that is polynomial in size(C) and exponential in d.*

An algorithm for Contraction. We are also able to generalize this to the more general Contraction problem, where the input is given as an arbitrary (non-linear) arithmetic circuit. Here the key issue is that the fixpoint may not be rational, and so we must find a suitably accurate approximate fixpoint. Our nested binary search approach can be adapted to do this.

Since we now deal with approximate fixpoints, we must cut off each of our nested binary search instances at an appropriate accuracy. Specifically, we must ensure that the solution is accurate enough so that we can correctly update the outer binary search. Choosing these cutoff points turns out to be quite involved, as we must choose different cutoff points depending on both the norm and the level of recursion, and moreover the ℓ_1 case requires a separate proof. The details of this are deferred to Appendix D.4, where the following theorem is shown.

Theorem 16. *For a contraction map $f : [0, 1]^d \to [0, 1]^d$ under $\|\cdot\|_p$ for $2 \leq p < \infty$, there is an algorithm to compute a point $v \in [0, 1]^d$ such that $\|f(v) - v\|_p < \varepsilon$ in time $O(p^{d^2} \log^d(1/\varepsilon) \log^d(p))$.*

Actually, our algorithm treats the function as a black-box, and so it can be applied to any contraction map, with Theorem 16 giving the number of queries that need to be made.

6 Open Problems

Many interesting open questions arise from our work. In the case of finding a Nash equilibrium of a two-player game, which we now know is PPAD-complete [10,15], the definition of PPAD was inspired by the path structure of the Lemke-Howson algorithm. Our definition of EndOfPotentialLine is directly inspired by the path structure of Lemke paths for P-matrix LCPs, as well as combining the canonical definitions of PPAD and PLS. Thus, the first conjecture we make is:

Conjecture 1. P-LCP *is hard for* EOPL *and promise* P-LCP *is hard for* UniqueEOPL.

This is actually two conjectures, but many natural approaches for proving the first result would almost automatically prove the second. Similar to those conjectures:

Conjecture 2. PL-Contraction *is hard for* UniqueEOPL.

We defined PL-Contraction as a promise problem in this paper. The reason was that if we start with a non-contracting instance of PL-Contraction then the EndOfPotentialLine instance we produce will either give us a fixpoint or a short proof that the instance was not contracting, but we do not know how to turn this proof into an actual explicit violation of contraction. Solving this issue is one interesting direction.

Another interesting question is whether the more general Contraction is in EndOfPotentialLine. Contraction is defined by a general arithmetic circuit, and may have an irrational fixpoint, so the problem is to find an approximate fixpoint. A natural approach is to approximate general circuits with LinearFIXP circuits, e.g., with interpolation, where the additional requirement would be to maintain the contraction property.

Finally, we have shown that EOPL is contained in CLS. We conjecture that:

Conjecture 3. CLS = EOPL.

It is not at all clear that this is true, and both possible outcomes would be interesting. Given the recent result of Daskalakis, Tzamos, and Zampetakis [17], one approach to proving this conjecture would be to show that Contraction with a metric as input is in EOPL. This seems challenging but plausible. If this conjecture is actually false, then it is natural to ask which further problems known to be in CLS also lie in EOPL.

References

[1] Ilan Adler and Sushil Verma. The linear complementarity problem, Lemke algorithm, perturbation, and the complexity class PPAD. Technical report, Manuscript, Depatrment of IEOR, University of California, Berkeley, CA 94720, 2011.

[2] David Aldous. Minimization algorithms and random walk on the d-cube. *The Annals of Probability*, pages 403–413, 1983.

[3] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In *Proc. of FOCS*, pages 1480–1498, 2015.

[4] Henrik Björklund, Sven Sandberg, and Sergei Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. In *Proc. MFCS*, pages 673–685, 2004.

[5] Ch. Boonyasiriwat, Kris Sikorski, and Ch. Xiong. A note on two fixed point problems. *J. Complexity*, 23(4-6):952–961, 2007.

[6] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proc. STOC*, pages 252–263, 2017.

[7] Xi Chen and Xiaotie Deng. On algorithms for discrete and approximate brouwer fixed points. In *Proc. of STOC*, pages 323–330, 2005.

[8] Xi Chen and Xiaotie Deng. Matching algorithmic bounds for finding a brouwer fixed point. *J. ACM*, 55(3):13:1–13:26, 2008.

[9] Xi Chen and Xiaotie Deng. On the complexity of 2d discrete fixed point problem. *Theor. Comput. Sci.*, 410(44):4448–4456, 2009.

[10] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *J. ACM*, 56(3):14, 2009.

[11] Sung-Jin Chung. NP-completeness of the linear complementarity problem. *Journal of Optimization Theory and Applications*, 60(3):393–399, 1989.

[12] Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

[13] Richard W Cottle, Jong-Shi Pang, and Richard E Stone. *The linear complementarity problem.* SIAM, 2009.

[14] Gregory E Coxson. The P-matrix problem is co-NP-complete. *Mathematical Programming*, 64(1):173–178, 1994.

[15] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[16] Constantinos Daskalakis and Christos Papadimitriou. Continuous local search. In *Proc. of SODA*, pages 790–804. Society for Industrial and Applied Mathematics, 2011.

[17] Constantinos Daskalakis, Christos Tzamos, and Manolis Zampetakis. A Converse to Banach's Fixed Point Theorem and its CLS Completeness. In *Proc. of STOC*, 2018.

[18] Xiaotie Deng, Qi Qi, Amin Saberi, and Jie Zhang. Discrete fixed points: Models, complexities, and applications. *Math. Oper. Res.*, 36(4):636–652, 2011.

[19] Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek, and Emo Welzl. ARRIVAL: a zero-player graph game in $\mathsf{NP} \cap \mathsf{coNP}$. In *A journey through discrete mathematics*, pages 367–374. Springer, Cham, 2017.

[20] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. of FOCS*, pages 368–377, 1991.

[21] Kousha Etessami and Mihalis Yannakakis. On the complexity of nash equilibria and other fixed points. *SIAM J. Comput.*, 39(6):2531–2597, 2010.

[22] Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. The complexity of pure Nash equilibria. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing (STOC)*, pages 604–612. ACM, 2004.

[23] John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. CLS: new problems and completeness. *CoRR*, abs/1702.06017, 2017. URL: `http://arxiv.org/abs/1702.06017`, `arXiv:1702.06017`.

[24] John Fearnley, Sanjay Jain, Sven Schewe, Frank Stephan, and Dominik Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In *Proc. of SPIN*, pages 112–121, 2017.

[25] John Fearnley, Marcin Jurdziński, and Rahul Savani. Linear complementarity algorithms for infinite games. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 382–393. Springer, 2010.

[26] John Fearnley and Rahul Savani. The complexity of all-switches strategy improvement. In *Proc. of SODA*, pages 130–139, 2016.

[27] Martin Gairing and Rahul Savani. Computing stable outcomes in hedonic games. In *Proc. of SAGT*, pages 174–185, 2010.

[28] Jugal Garg, Ruta Mehta, Milind Sohoni, and Vijay V. Vazirani. A complementary pivot algorithm for market equilibrium under separable piecewise-linear concave utilities. *SIAM J. Comput.*, 44(6):1820–1847, 2015.

[29] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a Nash equilibrium. In *Annual Cryptology Conference*, pages 579–604. Springer, 2016.

[30] Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubácek, Karel Král, Hagar Mosaad, and Veronika Slívová. ARRIVAL: next stop in CLS. *CoRR*, abs/1802.07702, 2018. `arXiv:1802.07702`.

[31] Bernd Gärtner and Leo Rüst. Simple stochastic games and P-matrix generalized linear complementarity problems. In *International Symposium on Fundamentals of Computation Theory*, pages 209–220. Springer, 2005.

[32] Thomas Dueholm Hansen and Rasmus Ibsen-Jensen. The complexity of interior point methods for solving discounted turn-based stochastic games. In *Conference on Computability in Europe*, pages 252–262, 2013.

[33] Michael D. Hirsch, Christos H. Papadimitriou, and Stephen A. Vavasis. Exponential lower bounds for finding brouwer fix points. *J. Complexity*, 5(4):379–416, 1989.

[34] Z. Huang, Leonid Khachiyan, and Krzysztof Sikorski. Approximating fixed points of weakly contracting mappings. *J. Complexity*, 15:200–213, 1999.

[35] Z. Huang, Leonid G. Khachiyan, and Christopher (Krzysztof) Sikorski. Approximating fixed points of weakly contracting mappings. *J. Complexity*, 15(2):200–213, 1999.

[36] Pavel Hubáček and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. In *Proc. of SODA*, pages 1352–1371. SIAM, 2017.

[37] David S Johnson, Christos H Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988.

[38] Marcin Jurdziński. Deciding the winner in parity games is in UP $\cap$ co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

[39] Marcin Jurdzinski and Ranko Lazic. Succinct progress measures for solving parity games. In *Proc. of LICS*, pages 1–9, 2017.

[40] Marcin Jurdziński and Rahul Savani. A simple P-matrix linear complementarity problem for discounted games. In *Conference on Computability in Europe*, pages 283–293. Springer, 2008.

[41] Karthik C. S. Did the train reach its destination: The complexity of finding a witness. *Inf. Process. Lett.*, 121:17–21, 2017.

[42] Masakazu Kojima, Nimrod Megiddo, Toshihito Noma, and Akiko Yoshise. *A unified approach to interior point algorithms for linear complementarity problems*, volume 538. Springer Science & Business Media, 1991.

[43] Masakazu Kojima, Nimrod Megiddo, and Yinyu Ye. An interior point potential reduction algorithm for the linear complementarity problem. *Mathematical Programming*, 54(1-3):267–279, 1992.

[44] Carlton E Lemke. Bimatrix equilibrium points and mathematical programming. *Management science*, 11(7):681–689, 1965.

[45] Walter Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and computation*, 117(1):151–155, 1995.

[46] Nimrod Megiddo. *A note on the complexity of P-matrix LCP and computing an equilibrium.* IBM Thomas J. Watson Research Division, 1988.

[47] Nimrod Megiddo and Christos H Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991.

[48] Ruta Mehta. Constant rank bimatrix games are ppad-hard. In *Proc. of STOC*, pages 545–554, 2014.

[49] Frédéric Meunier, Wolfgang Mulzer, Pauline Sarrabezolles, and Yannik Stein. The rainbow at the end of the line: A PPAD formulation of the colorful Carathéodory theorem with applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1342–1351, 2017.

[50] Walter D Morris Jr. Randomized pivot algorithms for P-matrix linear complementarity problems. *Mathematical programming*, 92(2):285–296, 2002.

[51] Katta G Murty. Computational complexity of complementary pivot methods. In *Complementarity and fixed point problems*, pages 61–73. Springer, 1978.

[52] A. Nemirovsky and D. B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley, New York, 1983.

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

[54] Anuj Puri. Theory of hybrid systems and discrete event systems. 1996.

[55] Aviad Rubinstein. Settling the complexity of computing approximate two-player nash equilibria. In *Proc. of FOCS*, pages 258–265, 2016.

[56] Alejandro A Schäffer and Mihalis Yannakakis. Simple local search problems that are hard to solve. *SIAM journal on Computing*, 20(1):56–87, 1991.

[57] Spencer Shellman and Krzysztof Sikorski. A recursive algorithm for the infinity-norm fixed point problem. *Journal of Complexity*, 19(6):799 – 834, 2003.

[58] Krzysztof Sikorski. *Optimal solution of Nonlinear Equations*. Oxford Press, New York, 200.

[59] Krzysztof Sikorski. Computational complexity of fixed points. *Journal of Fixed Point Theory and Applications*, 6(2):249–283, 2009.

[60] Alan Stickney and Layne Watson. Digraph models of bard-type algorithms for the linear complementarity problem. *Mathematics of Operations Research*, 3(4):322–333, 1978.

[61] Tibor Szabó and Emo Welzl. Unique sink orientations of cubes. In *Proc. of FOCS*, pages 547–555, 2001.

[62] Michael J Todd. Orientation in complementary pivot algorithms. *Mathematics of Operations Research*, 1(1):54–66, 1976.

[63] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.

A The Full Reductions and Proofs for Section 2

A.1 EndOfMeteredLine to EndOfPotentialLine

Given an instance $\mathcal{I}$of EndOfMeteredLine defined by circuits S, P and V on vertex set $\{0,1\}^n$ we are going to create an instance $\mathcal{I}'$ of EndOfPotentialLine with circuits S', P', and V' on vertex set $\{0,1\}^{(n+1)}$, i.e., we introduce one extra bit. This extra bit is essentially to take care of the difference in the value of potential at the starting point in EndOfMeteredLine and EndOfPotentialLine, namely 1 and 0 respectively.

Let $k = n+1$, then we create a potential function $V' : \{0,1\}^k \rightarrow \{0,\dots,2^k-1\}$. The idea is to make 0^k the starting point with potential zero as required, and to make all other vertices with first bit 0 be dummy vertices with self loops. The real graph will be embedded in vertices with first bit 1, i.e., of type $(1,\boldsymbol{u})$. Here by $(b,\boldsymbol{u}) \in \{0,1\}^k$, where $b \in \{0,1\}$ and $\boldsymbol{u} \in \{0,1\}^n$, we mean a k length bit string with first bit set to b and for each $i \in [2:k]$ bit i set to bit u_i.

Procedure $V'(b,\boldsymbol{u})$: If $b=0$ then Return 0, otherwise Return $V(\boldsymbol{u})$.

Procedure $S'(b,\boldsymbol{u})$:

1. If $(b,\boldsymbol{u}) = 0^k$ then Return $(1, 0^n)$
2. If $b = 0$ and $\boldsymbol{u} \neq 0^n$ then Return $(b,\boldsymbol{u})$ (creating self loop for dummy vertices)
3. If $b = 1$ and $V(\boldsymbol{u}) = 0$ then Return $(b,\boldsymbol{u})$ (vertices with zero potentials have self loops)
4. If $b = 1$ and $V(\boldsymbol{u}) > 0$ then Return $(b, S(\boldsymbol{u}))$ (the rest follows S)

Procedure $P'(b,\boldsymbol{u})$:

1. If $(b,\boldsymbol{u}) = 0^k$ then Return $(b,\boldsymbol{u})$ (initial vertex points to itself in P').
2. If $b = 0$ and $\boldsymbol{u} \neq 0^n$ then Return $(b,\boldsymbol{u})$ (creating self loop for dummy vertices)
3. If $b = 1$ and $\boldsymbol{u} = 0^n$ then Return 0^k (to make $(0,0^n) \rightarrow (1,0^n)$ edge consistent)
4. If $b = 1$ and $V(\boldsymbol{u}) = 0$ then Return $(b,\boldsymbol{u})$ (vertices with zero potentials have self loops)
5. If $b = 1$ and $V(\boldsymbol{u}) > 0$ and $\boldsymbol{u} \neq 0^n$ then Return $(b, P(\boldsymbol{u}))$ (the rest follows P)

Valid solutions of EndOfMeteredLine of type T2 and T3 requires the potential to be strictly greater than zero, while solutions of EndOfPotentialLine may have zero potential. However, a solution of EndOfPotentialLine can not be a self loop, so we've added self-loops around vertices with zero potential in the EndOfPotentialLine instance. By construction, the next lemma follows:

Lemma 17. *S', P', V' are well defined and polynomial in the sizes of S, P, V respectively.*

Our main theorem in this section is a consequence of the following three lemmas.

Lemma 18. *For any $\boldsymbol{x} = (b,\boldsymbol{u}) \in \{0,1\}^k$, $P'(\boldsymbol{x}) = S'(\boldsymbol{x}) = \boldsymbol{x}$ (self loop) iff $\boldsymbol{x} \neq 0^k$, and $b = 0$ or $V(\boldsymbol{u}) = 0$.*

Proof. This follows by the construction of V', the second condition in S' and P', and third and fourth conditions in S' and P' respectively. ☐

Lemma 19. *Let $\boldsymbol{x} = (b, \boldsymbol{u}) \in \{0,1\}^k$ be such that $S'(P'(\boldsymbol{x})) \neq \boldsymbol{x} \neq 0^k$ or $P'(S'(\boldsymbol{x})) \neq \boldsymbol{x}$ (an R1 type solution of* EndOfPotentialLine *instance $\mathcal{I}'$), then $\boldsymbol{u}$ is a solution of* EndOfMeteredLine *instance $\mathcal{I}$.*

Proof. The proof requires a careful case analysis. By the first conditions in the descriptions of S', P' and V', we have $\boldsymbol{x} \neq 0^k$. Further, since $\boldsymbol{x}$ is not a self loop, Lemma 18 implies $b = 1$ and $V'(1, \boldsymbol{u}) = V(\boldsymbol{u}) > 0$.

Case I. If $S'(P'(\boldsymbol{x})) \neq \boldsymbol{x} \neq 0^k$ then we will show that either $\boldsymbol{u}$ is a genuine start of a line other than 0^n giving a T1 type solution of EndOfMeteredLine instance $\mathcal{I}$, or there is some issue with the potential at $\boldsymbol{u}$ giving either a T2 or T3 type solution of $\mathcal{I}$. Since $S'(P'(1, 0^n)) = (1, 0^n)$, $\boldsymbol{u} \neq 0^n$. Thus if $S(P(\boldsymbol{u})) \neq \boldsymbol{u}$ then we get a T1 type solution of $\mathcal{I}$ and proof follows. If $V(\boldsymbol{u}) = 1$ then we get a T2 solution of $\mathcal{I}$ and proof follows.

Otherwise, we have $S(P(\boldsymbol{u})) = \boldsymbol{u}$ and $V(\boldsymbol{u}) > 1$. Now since also $b = 1$ $(1, \boldsymbol{u})$ is not a self loop (Lemma 18). Then it must be the case that $P'(1, \boldsymbol{u}) = (1, P(\boldsymbol{u}))$. However, $S'(1, P(\boldsymbol{u})) \neq (1, \boldsymbol{u})$ even though $S(P(\boldsymbol{u})) = \boldsymbol{u}$. This happens only when $P(\boldsymbol{u})$ is a self loop because of $V(P(\boldsymbol{u})) = 0$ (third condition of P'). Therefore, we have $V(\boldsymbol{u}) - V(P(\boldsymbol{u})) > 1$ implying that $\boldsymbol{u}$ is a T3 type solution of $\mathcal{I}$.

Case II. Similarly, if $P'(S'(\boldsymbol{x})) \neq \boldsymbol{x}$, then either $\boldsymbol{u}$ is a genuine end of a line of $\mathcal{I}$, or there is some issue with the potential at $\boldsymbol{u}$. If $P(S(\boldsymbol{u})) \neq \boldsymbol{u}$ then we get T1 solution of $\mathcal{I}$. Otherwise, $P(S(\boldsymbol{u})) = \boldsymbol{u}$ and $V(\boldsymbol{u}) > 0$. Now as $(b, \boldsymbol{u})$ is not a self loop and $V(\boldsymbol{u}) > 0$, it must be the case that $S'(b, \boldsymbol{u}) = (1, S(\boldsymbol{u}))$. However, $P'(1, S(\boldsymbol{u})) \neq (b, \boldsymbol{u})$ even though $P(S(\boldsymbol{u})) = \boldsymbol{u}$. This happens only when $S(\boldsymbol{u})$ is a self loop because of $V(S(\boldsymbol{u})) = 0$. Therefore, we get $V(S(\boldsymbol{u})) - V(\boldsymbol{u}) < 0$, i.e., $\boldsymbol{u}$ is a type T3 solution of $\mathcal{I}$. ☐

Lemma 20. *Let $\boldsymbol{x} = (b, \boldsymbol{u}) \in \{0,1\}^k$ be an R2 type solution of the constructed* EndOfPotentialLine *instance $\mathcal{I}'$, then $\boldsymbol{u}$ is a type T3 solution of* EndOfMeteredLine *instance $\mathcal{I}$.*

Proof. Clearly, $\boldsymbol{x} \neq 0^k$. Let $\mathbf{y} = (b', \boldsymbol{u}') = S'(\boldsymbol{x}) \neq \boldsymbol{x}$, and observe that $P(\mathbf{y}) = \boldsymbol{x}$. This also implies that $\mathbf{y}$ is not a self loop, and hence $b = b' = 1$ and $V(\boldsymbol{u}) > 0$ (Lemma 18). Further, $\mathbf{y} = S'(1, \boldsymbol{u}) = (1, S(\boldsymbol{u}))$, hence $\boldsymbol{u}' = S(\boldsymbol{u})$. Also, $V'(\boldsymbol{x}) = V'(1, \boldsymbol{u}) = V(\boldsymbol{u})$ and $V'(\mathbf{y}) = V'(1, \boldsymbol{u}') = V(\boldsymbol{u}')$.

Since $V'(\mathbf{y}) - V'(\boldsymbol{x}) \leq 0$ we get $V(\boldsymbol{u}') - V(\boldsymbol{u}) \leq 0 \Rightarrow V(S(\boldsymbol{u})) - V(\boldsymbol{u}) \leq 0 \Rightarrow V(S(\boldsymbol{u})) - V(\boldsymbol{u}) \neq 1$. Given that $V(\boldsymbol{u}) > 0$, $\boldsymbol{u}$ gives a type T3 solution of EndOfMeteredLine. ☐

Theorem 21. *An instance of* EndOfMeteredLine *can be reduced to an instance of* EndOfPotentialLine *in linear time such that a solution of the former can be constructed in a linear time from the solution of the latter.*

A.2 EndOfPotentialLine to EndOfMeteredLine

In this section we give a linear time reduction from an instance $\mathcal{I}$ of EndOfPotentialLine to an instance $\mathcal{I}'$ of EndOfMeteredLine. Let the given EndOfPotentialLine instance $\mathcal{I}$ be defined on vertex set $\{0,1\}^n$ and with procedures S, P and V, where $V : \{0,1\}^n \to \{0, \ldots, 2^m - 1\}$.

Valid Edge. We call an edge $\boldsymbol{u} \to \boldsymbol{v}$ valid if $\boldsymbol{v} = S(\boldsymbol{u})$ and $\boldsymbol{u} = P(\boldsymbol{v})$.

We construct an EndOfMeteredLine instance $\mathcal{I}'$ on $\{0,1\}^k$ vertices where $k = n + m$. Let S', P' and V' denotes the procedures for $\mathcal{I}'$ instance. The idea is to capture value $V(\boldsymbol{x})$ of the potential in the m least significant bits of vertex description itself, so that it can be gradually increased or decreased on valid edges. For vertices with irrelevant values of these least m significant bits we will create self loops. Invalid edges will also become self loops, e.g., if $\mathbf{y} = S(\boldsymbol{x})$ but $P(\mathbf{y}) \neq \boldsymbol{x}$ then set $S'(\boldsymbol{x}, .) = (\boldsymbol{x}, .)$. We will see how these can not introduce new solutions.

In order to ensure $V'(0^k) = 1$, the $V(S(0^n)) = 1$ case needs to be discarded. For this, we first do some initial checks to see if the given instance $\mathcal{I}$ is not trivial. If the input EndOfPotentialLine instance is trivial, in the sense that either 0^n or $S(0^n)$ is a solution, then we can just return it.

Lemma 22. *If 0^n or $S(0^n)$ are not solutions of* EndOfPotentialLine *instance $\mathcal{I}$ then $0^n \to S(0^n) \to S(S(0^n))$ are valid edges, and $V(S(S(0^n)) \geq 2$.*

Proof. Since both 0^n and $S(0^n)$ are not solutions, we have $V(0^n) < V(S(0^n)) < V(S(S(0^n)))$, $P(S(0^n)) = 0^n$, and for $\boldsymbol{u} = S(0^n)$, $S(P(\boldsymbol{u})) = \boldsymbol{u}$ and $P(S(\boldsymbol{u})) = \boldsymbol{u}$. In other words, $0^n \to S(0^n) \to S(S(0^n))$ are valid edges, and since $V(0^n) = 0$, we have $V(S(S(0^n)) \geq 2$. $\square$

Let us assume now on that 0^n and $S(0^n)$ are not solutions of $\mathcal{I}$, and then by Lemma 22, we have $0^n \to S(0^n) \to S(S(0^n))$ are valid edges, and $V(S(S(0^n)) \geq 2$. We can avoid the need to check whether $V(S(0))$ is one all together, by making 0^n point directly to $S(S(0^n))$ and make $S(0^n)$ a dummy vertex.

We first construct S' and P', and then construct V' which will give value zero to all self loops, and use the least significant m bits to give a value to all other vertices. Before describing S' and P' formally, we first describe the underlying principles. Recall that in $\mathcal{I}$ vertex set is $\{0,1\}^n$ and possible potential values are $\{0, \ldots, 2^m - 1\}$, while in $\mathcal{I}'$ vertex set is $\{0,1\}^k$ where $k = m + n$. We will denote a vertex of $\mathcal{I}'$ by a tuple $(\boldsymbol{u}, \pi)$, where $\boldsymbol{u} \in \{0,1\}^n$ and $\pi \in \{0, \ldots, 2^m - 1\}$. Here when we say that we introduce an *edge* $\boldsymbol{x} \to \mathbf{y}$ we mean that we introduce a valid edge from $\boldsymbol{x}$ to $\mathbf{y}$, i.e., $\mathbf{y} = S'(\boldsymbol{x})$ and $\boldsymbol{x} = P(\mathbf{y})$.

- Vertices of the form $(S(0^n), \pi)$ for any $\pi \in \{0,1\}^m$ and the vertex $(0^n, 1)$ are dummies and hence have self loops.
- If $V(S(S(0^n)) = 2$ then we introduce an edge $(0^n, 0) \to (S(S(0^n)), 2)$, otherwise
  - for $p = V(S(S(0^n))$, we introduce the edges $(0^n, 0) \to (0^n, 2) \to (0^n, 3) \ldots (0^n, p-1) \to (S(S(0^n)), p)$.
- If $\boldsymbol{u} \to \boldsymbol{u}'$ valid edge in $\mathcal{I}$ then let $p = V(\boldsymbol{u})$ and $p' = V(\boldsymbol{u}')$
  - If $p = p'$ then we introduce the edge $(\boldsymbol{u}, p) \to (\boldsymbol{u}', p')$.
  - If $p < p'$ then we introduce the edges $(\boldsymbol{u}, p) \to (\boldsymbol{u}, p+1) \to \ldots \to (\boldsymbol{u}, p'-1) \to (\boldsymbol{u}', p')$.
  - If $p > p'$ then we introduce the edges $(\boldsymbol{u}, p) \to (\boldsymbol{u}, p-1) \to \ldots \to (\boldsymbol{u}, p'+1) \to (\boldsymbol{u}', p')$.
- If $\boldsymbol{u} \neq 0^n$ is the start of a path, i.e., $S(P(\boldsymbol{u})) \neq \boldsymbol{u}$, then make $(\boldsymbol{u}, V(\boldsymbol{u}))$ start of a path by ensuring $P'(\boldsymbol{u}, V(\boldsymbol{u})) = (\boldsymbol{u}, V(\boldsymbol{u}))$.
- If $\boldsymbol{u}$ is the end of a path, i.e., $P(S(\boldsymbol{u})) \neq \boldsymbol{u}$, then make $(\boldsymbol{u}, V(\boldsymbol{u}))$ end of a path by ensuring $S'(\boldsymbol{u}, V(\boldsymbol{u})) = (\boldsymbol{u}, V(\boldsymbol{u}))$.

Last two bullets above remove singleton solutions from the system by making them self loops. However, this can not kill all the solutions since there is a path starting at 0^n, which has to end somewhere. Further, note that this entire process ensures that no new start or end of a paths are introduced.

Procedure $S'(\boldsymbol{u}, \pi)$.

```
1. If ($\boldsymbol{u} = 0^n$ and $\pi = 1$) or $\boldsymbol{u} = S(0^n)$ then Return $(\boldsymbol{u}, \pi)$.
2. If $(\boldsymbol{u}, \pi) = 0^k$, then let $\boldsymbol{u}' = S(S(0^n))$ and $p' = V(\boldsymbol{u}')$.
    (a) If $p' = 2$ then Return $(\boldsymbol{u}', 2)$ else Return $(0^n, 2)$.
3. If $\boldsymbol{u} = 0^n$ then
    (a) If $2 \leq \pi < p' - 1$ then Return $(0^n, \pi + 1)$.
    (b) If $\pi = p' - 1$ then Return $(S(S(0^n)), p')$.
    (c) If $\pi \geq p'$ then Return $(\boldsymbol{u}, \pi)$.
4. Let $\boldsymbol{u}' = S(\boldsymbol{u})$, $p' = V(\boldsymbol{u}')$, and $p = V(\boldsymbol{u})$.
5. If $P(\boldsymbol{u}') \neq \boldsymbol{u}$ or $\boldsymbol{u}' = \boldsymbol{u}$ then Return $(\boldsymbol{u}, \pi)$
6. If $\pi = p = p'$ or ($\pi = p$ and $p' = p + 1$) or ($\pi = p$ and $p' = p - 1$) then Return $(\boldsymbol{u}', p')$.
7. If $\pi < p \leq p'$ or $p \leq p' \leq \pi$ or $\pi > p \geq p'$ or $p \geq p' \geq \pi$ then Return $(\boldsymbol{u}, \pi)$
8. If $p < p'$, then if $p \leq \pi < p' - 1$ then Return $(\boldsymbol{u}, \pi + 1)$. If $\pi = p' - 1$ then Return $(\boldsymbol{u}', p')$.
9. If $p > p'$, then if $p \geq \pi > p' + 1$ then Return $(\boldsymbol{u}, \pi - 1)$. If $\pi = p' + 1$ then Return $(\boldsymbol{u}', p')$.
```

Procedure $P'(\boldsymbol{u}, \pi)$**.**

```
1. If ($\boldsymbol{u} = 0^n$ and $\pi = 1$) or $\boldsymbol{u} = S(0^n)$ then Return $(\boldsymbol{u}, \pi)$.
2. If $\boldsymbol{u} = 0^n$, then
    (a) If $\pi = 0$ then Return $0^k$.
    (b) If $\pi < V(S(S(0^n)))$ and $\pi \notin \{1, 2\}$ then Return $(0^n, \pi - 1)$.
    (c) If $\pi < V(S(S(0^n)))$ and $\pi = 2$ then Return $0^k$.
3. If $\boldsymbol{u} = S(S(0^n))$ and $\pi = V(S(S(0^n))$ then
    (a) If $\pi = 2$ then Return $(0^n, 0)$, else Return $(0^n, \pi - 1)$.
4. If $\pi = V(\boldsymbol{u})$ then
    (a) Let $\boldsymbol{u}' = P(\boldsymbol{u})$, $p' = V(\boldsymbol{u}')$, and $p = V(\boldsymbol{u})$.
    (b) If $S(\boldsymbol{u}') \neq \boldsymbol{u}$ or $\boldsymbol{u}' = \boldsymbol{u}$ then Return $(\boldsymbol{u}, \pi)$
    (c) If $p = p'$ then Return $(\boldsymbol{u}', p')$
    (d) If $p' < p$ then Return $(\boldsymbol{u}', p - 1)$ else Return $(\boldsymbol{u}', p + 1)$
5. Else % when $\pi \neq V(\boldsymbol{u})$
    (a) Let $\boldsymbol{u}' = S(\boldsymbol{u})$, $p' = V(\boldsymbol{u}')$, and $p = V(\boldsymbol{u})$
    (b) If $P(\boldsymbol{u}') \neq \boldsymbol{u}$ or $\boldsymbol{u}' = \boldsymbol{u}$ then Return $(\boldsymbol{u}, \pi)$
    (c) If $p' = p$ or $\pi < p < p'$ or $p < p' \leq \pi$ or $\pi > p > p'$ or $p > p' \geq \pi$ then Return $(\boldsymbol{u}, \pi)$
    (d) If $p < p'$, then If $p < \pi \leq p' - 1$ then Return $(\boldsymbol{u}, \pi - 1)$.
```

(e) If $p > p'$, then if $p > \pi \geq p' + 1$ then Return $(\boldsymbol{u}, \pi + 1)$.

As mentioned before, the intuition for the potential function procedure V' is to return zero for self loops, return 1 for 0^k, and return the number specified by the lowest m bits for the rest.

Procedure $V'(\boldsymbol{u}, \pi)$. Let $\boldsymbol{x} = (\boldsymbol{u}, \pi)$ for notational convenience.

1. If $\boldsymbol{x} = 0^k$, then Return 1.
2. If $S'(\boldsymbol{x}) = \boldsymbol{x}$ and $P'(\boldsymbol{x}) = \boldsymbol{x}$ then Return 0.
3. If $S'(\boldsymbol{x}) \neq \boldsymbol{x}$ or $P'(\boldsymbol{x}) \neq \boldsymbol{x}$ then Return π.

The fact that procedures S', P' and V' give a valid EndOfMeteredLine instance follows from construction.

Lemma 23. *Procedures S', P' and V' gives a valid* EndOfMeteredLine *instance on vertex set $\{0,1\}^k$, where $k = m + n$ and $V' : \{0,1\}^k \to \{0, \dots, 2^k - 1\}$.*

The next three lemmas shows how to construct a solution of EndOfPotentialLine instance $\mathcal{I}$ from a type T1, T2, or T3 solution of constructed EndOfMeteredLine instance $\mathcal{I}'$. The basic idea for next lemma, which handles type T1 solutions, is that we never create spurious end or start of a path.

Lemma 24. *Let $\boldsymbol{x} = (\boldsymbol{u}, \pi)$ be a type T1 solution of constructed* EndOfMeteredLine *instance $\mathcal{I}'$. Then $\boldsymbol{u}$ is a type R1 solution of the given* EndOfPotentialLine *instance $\mathcal{I}$.*

Proof. Let $\Delta = 2^m - 1$. In $\mathcal{I}'$, clearly $(0^n, \pi)$ for any $\pi \in 1, \dots, \Delta$ is not a start or end of a path, and $(0^n, 0)$ is not an end of a path. Therefore, $\boldsymbol{u} \neq 0^n$. Since $(S(0^n), \pi), \forall \pi \in \{0, \dots, \Delta\}$ are self loops, $\boldsymbol{u} \neq S(0^n)$.

If to the contrary, $S(P(\boldsymbol{u})) = \boldsymbol{u}$ and $P(S(\boldsymbol{u})) = \boldsymbol{u}$. If $S(\boldsymbol{u}) = \boldsymbol{u} = P(\boldsymbol{u})$ then $(\boldsymbol{u}, \pi)$, $\forall \pi \in \{0, \dots, \Delta\}$ are self loops, a contradiction.

For the remaining cases, let $P'(S'(\boldsymbol{x})) \neq \boldsymbol{x}$, and let $\boldsymbol{u}' = S(\boldsymbol{u})$. . There is a valid edge from $\boldsymbol{u}$ to $\boldsymbol{u}'$ in $\mathcal{I}$. Then we will create valid edges from $(\boldsymbol{u}, V(\boldsymbol{u}))$ to $(S(\boldsymbol{u}), V(S(\boldsymbol{u}))$ with appropriately changing second coordinates. The rest of $(\boldsymbol{u}, .)$ are self loops, a contradiction.

Similar argument follows for the case when $S'(P'(\boldsymbol{x})) \neq \boldsymbol{x}$. ☐

The basic idea behind the next lemma is that a T2 type solution in $\mathcal{I}'$ has potential 1. Therefore, it is surely not a self loop. Then it is either an end of a path or near an end of a path, or else near a potential violation.

Lemma 25. *Let $\boldsymbol{x} = (\boldsymbol{u}, \pi)$ be a type T2 solution of $\mathcal{I}'$. Either $\boldsymbol{u} \neq 0^n$ is start of a path in $\mathcal{I}$ (type R1 solution), or $P(\boldsymbol{u})$ is an R1 or R2 type solution in $\mathcal{I}$, or $P(P(\boldsymbol{u}))$ is an R2 type solution in $\mathcal{I}$.*

Proof. Clearly $\boldsymbol{u} \neq 0^n$, and $\boldsymbol{x}$ is not a self loop, i.e., it is not a dummy vertex with irrelevant value of π. Further, $\pi = 1$. If $\boldsymbol{u}$ is a start or end of a path in $\mathcal{I}$ then done.

Otherwise, if $V(P(\boldsymbol{u})) > \pi$ then we have $V(\boldsymbol{u}) \leq \pi$ and hence $V(\boldsymbol{u}) - V(P(\boldsymbol{u})) \leq 0$ giving $P(\boldsymbol{u})$ as an R2 type solution of $\mathcal{I}$. If $V(P(\boldsymbol{u})) < \pi = 1$ then $V(P(\boldsymbol{u})) = 0$. Since potential can not go below zero, either $P(\boldsymbol{u})$ is an end of a path, or for $\boldsymbol{u}'' = P(P(\boldsymbol{u}))$ and $\boldsymbol{u}' = P(\boldsymbol{u})$ we have $\boldsymbol{u}' = S(\boldsymbol{u}'')$ and $V(\boldsymbol{u}') - V(\boldsymbol{u}'') \leq 0$, giving $\boldsymbol{u}''$ as a type R2 solution of $\mathcal{I}$. ☐

At a type T3 solution of $\mathcal{I}'$ potential is strictly positive, hence these solutions are not self loops. If they correspond to potential violation in $\mathcal{I}$ then we get a type R2 solution. But this may not be the case, if we made S' or P' self pointing due to end or start of a path respectively. In that case, we get a type R1 solution. The next lemma formalizes this intuition.

Lemma 26. *Let $\boldsymbol{x} = (\boldsymbol{u}, \pi)$ be a type T3 solution of $\mathcal{I}'$. If $\boldsymbol{x}$ is a start or end of a path in $\mathcal{I}'$ then $\boldsymbol{u}$ gives a type R1 solution in $\mathcal{I}$. Otherwise $\boldsymbol{u}$ gives a type R2 solution of $\mathcal{I}$.*

Proof. Since $V'(\boldsymbol{x}) > 0$, it is not a self loop and hence is not dummy, and $\boldsymbol{u} \neq 0^n$. If $\boldsymbol{u}$ is start or end of a path then $\boldsymbol{u}$ is a type R1 solution of $\mathcal{I}$. Otherwise, there are valid incoming and outgoing edges at $\boldsymbol{u}$, therefore so at $\boldsymbol{x}$.

If $V((S(\boldsymbol{x})) - V(\boldsymbol{x}) \neq 1$, then since potential either remains the same or increases or decreases exactly by one on edges of $\mathcal{I}'$, it must be the case that $V(S(\boldsymbol{x})) - V(\boldsymbol{x}) \leq 0$. This is possible only when $V(S(\boldsymbol{u})) \leq V(\boldsymbol{u})$. Since $\boldsymbol{u}$ is not an end of a path we do have $S(\boldsymbol{u}) \neq \boldsymbol{u}$ and $P(S(\boldsymbol{u})) = \boldsymbol{u}$. Thus, $\boldsymbol{u}$ is a type T2 solution of $\mathcal{I}$.

If $V((\boldsymbol{x}) - V(P(\boldsymbol{x})) \neq 1$, then by the same argument we get that for $(\boldsymbol{u}'', \pi'') = P(\boldsymbol{u})$, $\boldsymbol{u}''$ is a type R2 solution of $\mathcal{I}$. □

Our main theorem follows using Lemmas 23, 24, 25, and 26.

Theorem 27. *An instance of* EndOfPotentialLine *can be reduced to an instance of* EndOfMeteredLine *in polynomial time such that a solution of the former can be constructed in a linear time from the solution of the latter.*

B The Full Reduction and Proofs for Section 3

B.1 Lemke's algorithm

The explanation of Lemke's algorithm in this section is taken from [28]. The problem is interesting only when $\boldsymbol{q} \not\geq 0$, since otherwise $\mathbf{y} = 0$ is a trivial solution. Let us introduce slack variables $\boldsymbol{s}$ to obtain the following equivalent formulation:

$$M\mathbf{y} + \boldsymbol{s} = \boldsymbol{q}, \quad \mathbf{y} \geq 0, \quad \boldsymbol{s} \geq 0 \quad \text{and} \quad y_i s_i = 0, \ \forall i \in [d]. \tag{2}$$

Let $\mathcal{Q}$ be the polyhedron in $2d$ dimensional space defined by the first three conditions; we will assume that $\mathcal{Q}$ is non-degenerate (just for simplicity of exposition; this will not matter for our reduction). Under this condition, any solution to (2) will be a vertex of $\mathcal{Q}$, since it must satisfy $2d$ equalities. Note that the set of solutions may be disconnected. The ingenious idea of Lemke was to introduce a new variable and consider the system:

$$M\mathbf{y} + \boldsymbol{s} - z\mathbf{1} = \boldsymbol{q}, \quad \mathbf{y} \geq 0, \quad \boldsymbol{s} \geq 0, \quad z \geq 0 \quad \text{and} \quad y_i s_i = 0, \ \forall i \in [d]. \tag{3}$$

The next lemma follows by construction of (3).

Lemma 28. *Given $(M, \boldsymbol{q})$, $(\mathbf{y}, \boldsymbol{s}, z)$ satisfies* (3) *with $z = 0$ iff $\mathbf{y}$ satisfies* (1).

Let $\mathcal{P}$ be the polyhedron in $2d+1$ dimensional space defined by the first four conditions of (3), i.e.,

$$\mathcal{P} = \{(\mathbf{y}, \boldsymbol{s}, z) \mid M\mathbf{y} + \boldsymbol{s} - z\mathbf{1} = \boldsymbol{q}, \quad \mathbf{y} \geq 0, \quad \boldsymbol{s} \geq 0, \quad z \geq 0\}; \tag{4}$$

for now, we will assume that $\mathcal{P}$ is *non-degenerate*.

Since any solution to (3) must still satisfy $2d$ equalities in $\mathcal{P}$, the set of solutions, say S, will be a subset of the one-skeleton of $\mathcal{P}$, i.e., it will consist of edges and vertices of $\mathcal{P}$. Any solution to the original system (2) must satisfy the additional condition $z = 0$ and hence will be a vertex of $\mathcal{P}$.

Now S turns out to have some nice properties. Any point of S is *fully labeled* in the sense that for each i, $y_i = 0$ or $s_i = 0$. We will say that a point of S *has duplicate label* i if $y_i = 0$ and $s_i = 0$ are both satisfied at this point. Clearly, such a point will be a vertex of $\mathcal{P}$ and it will have only one duplicate label. Since there are exactly two ways of relaxing this duplicate label, this vertex must have exactly two edges of S incident at it. Clearly, a solution to the original system (i.e., satisfying $z = 0$) will be a vertex of $\mathcal{P}$ that does not have a duplicate label. On relaxing $z = 0$, we get the unique edge of S incident at this vertex.

As a result of these observations, we can conclude that S consists of paths and cycles. Of these paths, Lemke's algorithm explores a special one. An unbounded edge of S such that the vertex of $\mathcal{P}$ it is incident on has $z > 0$ is called a *ray*. Among the rays, one is special – the one on which $\mathbf{y} = 0$. This is called the *primary ray* and the rest are called *secondary rays*. Now Lemke's algorithm explores, via pivoting, the path starting with the primary ray. This path must end either in a vertex satisfying $z = 0$, i.e., a solution to the original system, or a secondary ray. In the latter case, the algorithm is unsuccessful in finding a solution to the original system; in particular, the original system may not have a solution. We give the full pseudo-code for Lemke's algorithm in Table 1.

Table 1: Lemke's Complementary Pivot Algorithm

```
If $\boldsymbol{q} \geq 0$ then Return $\mathbf{y} \leftarrow \mathbf{0}$
$\mathbf{y} \leftarrow 0, z \leftarrow |\min_{i \in [d]} q_i|, \boldsymbol{s} = \boldsymbol{q} + z\mathbf{1}$
$i \leftarrow$ duplicate label at vertex $(\mathbf{y}, \boldsymbol{s}, z)$ in $\mathcal{P}$. $flag \leftarrow 1$
While $z > 0$ do
 If $flag = 1$ then set $(\mathbf{y}', \boldsymbol{s}', z') \leftarrow$ vertex obtained by relaxing $y_i = 0$ at $(\mathbf{y}, \boldsymbol{s}, z)$ in $\mathcal{P}$
 Else set $(\mathbf{y}', \boldsymbol{s}', z') \leftarrow$ vertex obtained by relaxing $s_i = 0$ at $(\mathbf{y}, \boldsymbol{s}, z)$ in $\mathcal{P}$
 If $z > 0$ then
  $i \leftarrow$ duplicate label at $(\mathbf{y}', \boldsymbol{s}', z')$
  If $v_i > 0$ and $v'_i = 0$ then $flag \leftarrow 1$. Else $flag \leftarrow 0$
  $(\mathbf{y}, \boldsymbol{s}, z) \leftarrow (\mathbf{y}', \boldsymbol{s}', z')$
End While
Return $\mathbf{y}$
```

B.2 The reduction from P-LCP to EndOfPotentialLine

In this section, we give a polynomial-time reduction from P-LCP to EndOfPotentialLine.

It is well known that if matrix M is a P-matrix (P-LCP), then z strictly decreases on the path traced by Lemke's algorithm [13]. Furthermore, by a result of Todd [62, Section 5], paths traced by complementary pivot rule can be locally oriented. Based on these two facts, we now derive a polynomial-time reduction from P-LCP to EndOfPotentialLine.

Let $\mathcal{I} = (M, \boldsymbol{q})$ be a given P-LCP instance, and let $\mathcal{L}$ be the length of the bit representation of M and $\boldsymbol{q}$. We will reduce $\mathcal{I}$ to an EndOfPotentialLine instance $\mathcal{E}$ in time poly($\mathcal{L}$). According to Definition 2, the instance $\mathcal{E}$ is defined by its vertex set vert, and procedures S (successor), P (predecessor) and V (potential). Next we define each of these.

As discussed in Section B.1 the linear constraints of (3) on which Lemke's algorithm operates forms a polyhedron $\mathcal{P}$ given in (4). We assume that $\mathcal{P}$ is non-degenerate. This is without loss of generality since, a typical way to ensure this is by perturbing $\boldsymbol{q}$ so that configurations of solution vertices remain unchanged [13], and since M is unchanged the LCP is still P-LCP.

Lemke's algorithm traces a path on feasible points of (3) which is on 1-skeleton of $\mathcal{P}$ starting at $(\mathbf{y}^0, \boldsymbol{s}^0, z^0)$, where:

$$\mathbf{y}^0 = 0, \qquad z^0 = |\min_{i \in [d]} q_i|, \qquad \boldsymbol{s}^0 = \boldsymbol{q} + z\mathbf{1} \tag{5}$$

We want to capture vertex solutions of (3) as vertices in EndOfPotentialLine instance $\mathcal{E}$. To differentiate we will sometimes call the latter *configurations*. Vertex solutions of (3) are exactly the vertices of polyhedron $\mathcal{P}$ with either $y_i = 0$ or $s_i = 0$ for each $i \in [d]$. Vertices of (3) with $z = 0$ are our final solutions (Lemma 28). While each of its *non-solution* vertex has a duplicate label. Thus, a vertex of this path can be uniquely identified by which of $y_i = 0$ and $s_i = 0$ hold for each i and its duplicate label. This gives us a representation for vertices in the EndOfPotentialLine instance $\mathcal{E}$.

EndOfPotentialLine Instance $\mathcal{E}$.

- Vertex set $\mathsf{vert} = \{0,1\}^n$ where $n = 2d$.
- Procedures S and P as defined in Tables 4 and 6 respectively
- Potential function $V : \mathsf{vert} \to \{0, 1, \ldots, 2^m - 1\}$ defined in Table 5 for $m = \lceil ln(2\Delta^3) \rceil$, where

$$\Delta = (n! \cdot I_{max}^{2d+1}) + 1$$

  and $I_{max} = \max\{\max_{i,j \in [d]} M(i,j),\ \max_{i \in [d]} |q_i|\}$.

For any vertex $\boldsymbol{u} \in \mathsf{vert}$, the first d bits of $\boldsymbol{u}$ represent which of the two inequalities, namely $y_i \geq 0$ and $s_i \geq 0$, are tight for each $i \in [d]$. A valid setting of the second set of d bits will have at most one non-zero bit – if none is one then $z = 0$, otherwise the location of one bit indicates the duplicate label. Thus, there are many invalid configurations, namely those with more than one non-zero bit in the second set of d bits. These are dummies that we will handle separately, and we define a procedure IsValid to identify non-dummy vertices in Table 2. To go between "valid" vertices of $\mathcal{E}$ and corresponding vertices of the Lemke polytope $\mathcal{P}$ of LCP $\mathcal{I}$, we define procedures EtoI and ItoE in Table 3.

By construction of IsValid, EtoI and ItoE, the next lemma follows.

Lemma 29. *If IsValid($\boldsymbol{u}$) = 1 then $\boldsymbol{u}$ = ItoE(EtoI($\boldsymbol{u}$)), and the corresponding vertex $(\mathbf{y}, \boldsymbol{s}, z) \in$ EtoI($\boldsymbol{u}$) of $\mathcal{P}$ is feasible in (3). If $(\mathbf{y}, \boldsymbol{s}, z)$ is a feasible vertex of (3) then $\boldsymbol{u}$ = ItoE$(\mathbf{y}, \boldsymbol{s}, z)$ is a valid configuration,* i.e., *IsValid($\boldsymbol{u}$) = 1.*

Table 2: Procedure IsValid($\boldsymbol{u}$)

```
If $\boldsymbol{u} = 0^n$ then Return 1
Else let $\tau = (u_{(d+1)} + \cdots + u_{2d})$
  If $\tau > 1$ then Return 0
  Let $S \leftarrow \emptyset$. % set of tight inequalities.
  If $\tau = 0$ then $S = S \cup \{z = 0\}$.
  Else
     Set $l \leftarrow$ index of the non-zero coordinate in vector $(u_{(d+1)}, \ldots, u_{2d})$.
     Set $S = \{y_l = 0, s_l = 0\}$.
  For each $i$ from 1 to $d$ do
     If $u_i = 0$ then $S = S \cup \{y_i = 0\}$, Else $S = S \cup \{s_i = 0\}$
  Let $A$ be a matrix formed by lhs of equalities $M\mathbf{y} + \boldsymbol{s} - \mathbf{1}z = \boldsymbol{q}$ and that of set $S$
  Let $\mathbf{b}$ be the corresponding rhs, namely $\mathbf{b} = [\boldsymbol{q}; \mathbf{0}_{d\times 1}]$.
  Let $(\mathbf{y}', \boldsymbol{s}', z') \leftarrow \mathbf{b} * A^{-1}$
  If $(\mathbf{y}', \boldsymbol{s}', z') \in \mathcal{P}$ then Return 1, Else Return 0
```

Table 3: Procedures ItoE($\boldsymbol{u}$) and EtoI($\mathbf{y}, \boldsymbol{s}, z$)

```
ItoE($\mathbf{y}, \boldsymbol{s}, z$)
  If $\exists i \in [d]$ s.t. $y_i * s_i \neq 0$ then Return $(\mathbf{0}_{(2d-2)\times 1}; 1; 1)$ % Invalid
  Set $\boldsymbol{u} \leftarrow \mathbf{0}_{2d\times 1}$. Let $DL = \{i \in [d] \mid y_i = 0$ and $s_i = 0\}$.
  If $|DL| > 1$ then Return $(\mathbf{0}_{(2d-2)\times 1}; 1; 1)$ %In valid
  If $|DL| = 1$ then for $i \in DL$, set $u_i \leftarrow 1$
  For each $i \in [d]$ If $s_i = 0$ then set $u_{d+i} \leftarrow 1$
  Return $\boldsymbol{u}$

EtoI($\boldsymbol{u}$)
  If $\boldsymbol{u} = 0^n$ then Return $(\mathbf{0}_{d\times 1}, \boldsymbol{q} + z^0 + 1, z^0 + 1)$ % This case will never happen
  If IsValid($\boldsymbol{u}$)=0 then Return $\mathbf{0}_{(2d+1)\times 1}$
  Let $\tau = (u_{(d+1)} + \cdots + u_{2d})$
  Let $S \leftarrow \emptyset$. % set of tight inequalities.
  If $\tau = 0$ then $S = S \cup \{z = 0\}$.
  Else
    Set $l \leftarrow$ index of non-zero coordinate in vector $(u_{(d+1)}, \ldots, u_{2d})$.
    Set $S = \{y_l = 0, s_l = 0\}$.
  For each $i$ from 1 to $d$ do
    If $u_i = 0$ then $S = S \cup \{y_i = 0\}$, Else $S = S \cup \{s_i = 0\}$
  Let $A$ be a matrix formed by lhs of equalities $M\mathbf{y} + \boldsymbol{s} - \mathbf{1}z = \boldsymbol{q}$ and that of set $S$
  Let $\mathbf{b}$ be the corresponding rhs, namely $\mathbf{b} = [\boldsymbol{q}; \mathbf{0}_{d\times 1}]$.
  Return $\mathbf{b} * A^{-1}$
```

Proof. The only thing that can go wrong is that the matrix A generated in IsValid and EtoI procedures are singular, or the set of double labels DL generated in ItoE has more than one elements. Each of these are possible only when more than $2d + 1$ equalities of $\mathcal{P}$ hold at the corresponding point $(\mathbf{y}, \boldsymbol{s}, z)$, violating non-degeneracy assumption. ◻

The main idea behind procedures S and P, given in Tables 4 and 6 respectively, is the following (also see Figure 2): Make dummy configurations in vert to point to themselves with cycles of length

Table 4: Successor Procedure $S(\boldsymbol{u})$

```
If IsValid($\boldsymbol{u}$) $= 0$ then Return $\boldsymbol{u}$
If $\boldsymbol{u} = 0^n$ then Return ItoE($\mathbf{y}^0, \boldsymbol{s}^0, z^0$)
$\boldsymbol{x} = (\mathbf{y}, \boldsymbol{s}, z) \leftarrow$ EtoI($\boldsymbol{u}$)
If $z = 0$ then
  $\boldsymbol{x}^1 \leftarrow$ vertex obtained by relaxing $z = 0$ at $\boldsymbol{x}$ in $\mathcal{P}$.
  If Todd [62] prescribes edge from $\boldsymbol{x}$ to $\boldsymbol{x}^1$
   then set $\boldsymbol{x}' \leftarrow \boldsymbol{x}^1$. Else Return $\boldsymbol{u}$
Else set $l \leftarrow$ duplicate label at $\boldsymbol{x}$
  $\boldsymbol{x}^1 \leftarrow$ vertex obtained by relaxing $y_l = 0$ at $\boldsymbol{x}$ in $\mathcal{P}$
  $\boldsymbol{x}^2 \leftarrow$ vertex obtained by relaxing $s_l = 0$ at $\boldsymbol{x}$ in $\mathcal{P}$
  If Todd [62] prescribes edge from $\boldsymbol{x}$ to $\boldsymbol{x}^1$
   then $\boldsymbol{x}' = \boldsymbol{x}^1$
  Else $\boldsymbol{x}' = \boldsymbol{x}^2$
Let $\boldsymbol{x}'$ be $(\mathbf{y}', \boldsymbol{s}', z')$.
If $z > z'$ then Return ItoE($\boldsymbol{x}'$). Else Return $\boldsymbol{u}$.
```

Table 5: Potential Value $V(\boldsymbol{u})$

```
If IsValid($\boldsymbol{u}$) $= 0$
 then Return 0
If $\boldsymbol{u} = 0^n$
 then Return 0
$(\mathbf{y}, \boldsymbol{s}, z) \leftarrow$ EtoI($\boldsymbol{u}$)
Return $\lfloor \Delta^2 * (\Delta - z) \rfloor$
```

one, so that they can never be solutions. The starting vertex $0^n \in \mathsf{vert}$ points to the configuration that corresponds to the first vertex of the Lemke path, namely $\boldsymbol{u}^0 = \text{ItoE}(\mathbf{y}^0, \boldsymbol{s}^0, z^0)$. Precisely, $S(0^n) = \boldsymbol{u}^0$, $P(\boldsymbol{u}^0) = 0^n$ and $P(0^n) = 0^n$ (start of a path).

For the remaining cases, let $\boldsymbol{u} \in \mathsf{vert}$ have corresponding representation $\boldsymbol{x} = (\mathbf{y}, \boldsymbol{s}, z) \in \mathcal{P}$, and suppose $\boldsymbol{x}$ has a duplicate label. As one traverses a Lemke path for a P-LCPs, the value of z monotonically decreases. So, for $S(\boldsymbol{u})$ we compute the adjacent vertex $\boldsymbol{x}' = (\mathbf{y}', \boldsymbol{s}', z')$ of $\boldsymbol{x}$ on Lemke path such that the edge goes from $\boldsymbol{x}$ to $\boldsymbol{x}'$, and if the $z' < z$, as expected, then we point $S(\boldsymbol{u})$ to configuration of $\boldsymbol{x}'$ namely ItoE($\boldsymbol{x}'$). Otherwise, we let $S(\boldsymbol{u}) = \boldsymbol{u}$. Similarly, for $P(\boldsymbol{u})$, we find $\boldsymbol{x}'$ such that edge is from $\boldsymbol{x}'$ to $\boldsymbol{x}$, and then we let $P(\boldsymbol{u})$ be ItoE($\boldsymbol{x}'$) if $z' > z$ as expected, otherwise $P(\boldsymbol{u}) = \boldsymbol{u}$.

For the case when $\boldsymbol{x}$ does not have a duplicate label, then we have $z = 0$. This is handled separately since such a vertex has exactly one incident edge on the Lemke path, namely the one obtained by relaxing $z = 0$. According to the direction of this edge, we do similar process as before. For example, if the edge goes from $\boldsymbol{x}$ to $\boldsymbol{x}'$, then, if $z' < z$, we set $S(\boldsymbol{u}) = \text{ItoE}(\boldsymbol{x}')$ else $S(\boldsymbol{u}) = \boldsymbol{u}$, and we always set $P(\boldsymbol{u}) = \boldsymbol{u}$. In case the edge goes from $\boldsymbol{x}'$ to $\boldsymbol{x}$, we always set $S(\boldsymbol{u}) = \boldsymbol{u}$, and we set $P(\boldsymbol{u})$ depending on whether or not $z' > z$.

The potential function V, formally defined in Table 5, gives a value of zero to dummy vertices and the starting vertex 0^n. To all other vertices, essentially it is $((z^0 - z) * \Delta^2) + 1$. Since value of z starts at z^0 and keeps decreasing on the Lemke path this value will keep increasing starting from zero at the starting vertex 0^n. Multiplication by Δ^2 will ensure that if $z_1 > z_2$ then the corresponding potential values will differ by at least one. This is because, since z_1 and z_2 are coordinates of two vertices of polytope $\mathcal{P}$, their maximum value is Δ and their denominator is also bounded above by Δ. Hence $z_1 - z_2 \leq 1/\Delta^2$ (Lemma 31).

To show correctness of the reduction we need to show two things: (i) All the procedures are well-defined and polynomial time. (ii) We can construct a solution of $\mathcal{I}$ from a solution of $\mathcal{E}$ in polynomial time.

Lemma 30. *Functions P, S and V of instance $\mathcal{E}$ are well defined, making $\mathcal{E}$ a valid* EndOfPotentialLine *instance.*

Table 6: Predecessor Procedure $P(\boldsymbol{u})$

```
**If** IsValid($\boldsymbol{u}$) $= 0$ **then Return** $\boldsymbol{u}$
**If** $\boldsymbol{u} = 0^n$ **then Return** $\boldsymbol{u}$
$(\mathbf{y}, \boldsymbol{s}, z) \leftarrow$ EtoI($\boldsymbol{u}$)
**If** $(\mathbf{y}, \boldsymbol{s}, z) = (\mathbf{y}^0, \boldsymbol{s}^0, z^0)$ **then Return** $0^n$
**If** $z = 0$ **then**
  $\boldsymbol{x}^1 \leftarrow$ vertex obtained by relaxing $z = 0$ at $\boldsymbol{x}$ in $\mathcal{P}$.
  **If** Todd [62] prescribes edge from $\boldsymbol{x}^1$ to $\boldsymbol{x}$ **then** set $\boldsymbol{x}' \leftarrow \boldsymbol{x}^1$. **Else Return** $\boldsymbol{u}$
**Else**
  $l \leftarrow$ duplicate label at $\boldsymbol{x}$
  $\boldsymbol{x}^1 \leftarrow$ vertex obtained by relaxing $y_l = 0$ at $\boldsymbol{x}$ in $\mathcal{P}$
  $\boldsymbol{x}^2 \leftarrow$ vertex obtained by relaxing $s_l = 0$ at $\boldsymbol{x}$ in $\mathcal{P}$
  **If** Todd [62] prescribes edge from $\boldsymbol{x}^1$ to $\boldsymbol{x}$ **then** $\boldsymbol{x}' = \boldsymbol{x}^1$ **Else** $\boldsymbol{x}' = \boldsymbol{x}^2$
Let $\boldsymbol{x}'$ be $(\mathbf{y}', \boldsymbol{s}', z')$. **If** $z < z'$ **then Return** ItoE($\boldsymbol{x}'$). **Else Return** $\boldsymbol{u}$.
```

Proof. Since all three procedures are polynomial-time in $\mathcal{L}$, they can be defined by poly($\mathcal{L}$)-sized Boolean circuits. Furthermore, for any $\boldsymbol{u} \in \mathsf{vert}$, we have that $S(\boldsymbol{u}), P(\boldsymbol{u}) \in \mathsf{vert}$. For V, since the value of $z \in [0,\ \Delta - 1]$, we have $0 \leq \Delta^2(\Delta - z) \leq \Delta^3$. Therefore, $V(\boldsymbol{u})$ is an integer that is at most $2 \cdot \Delta^3$ and hence is in set $\{0, \ldots, 2^m - 1\}$. ☐

There are two possible types of solutions of an EndOfPotentialLine instance. One indicates the beginning or end of a line, and the other is a vertex with locally optimal potential (that does not point to itself). First we show that the latter case never arise. For this, we need the next lemma, which shows that potential differences in two adjacent configurations adheres to differences in the value of z at corresponding vertices.

Lemma 31. *Let $\boldsymbol{u} \neq \boldsymbol{u}'$ be two valid configurations, i.e., IsValid($\boldsymbol{u}$) $=$ IsValid($\boldsymbol{u}'$) $= 1$, and let $(\mathbf{y}, \boldsymbol{s}, z)$ and $(\mathbf{y}', \boldsymbol{s}', z')$ be the corresponding vertices in $\mathcal{P}$. Then the following holds: (i) $V(\boldsymbol{u}) = V(\boldsymbol{u}')$ iff $z = z'$. (ii) $V(\boldsymbol{u}) > V(\boldsymbol{u}')$ iff $z < z'$.*

Proof. Among the valid configurations all except $\mathbf{0}$ has positive V value. Therefore, wlog let $\boldsymbol{u}, \boldsymbol{u}' \neq \mathbf{0}$. For these we have $V(\boldsymbol{u}) = \lfloor \Delta^2 * (\Delta - z) \rfloor$, and $V(\boldsymbol{u}') = \lfloor \Delta^2 * (\Delta - z') \rfloor$.

Note that since both z and z' are coordinates of vertices of $\mathcal{P}$, whose description has highest coefficient of $\max\{\max_{i,j \in [d]} M(i,j), \max_{i \in [d]} |q_i|\}$, and therefore their numerator and denominator both are bounded above by Δ. Therefore, if $z < z'$ then we have

$$z' - z \geq \frac{1}{\Delta^2} \Rightarrow ((\Delta - z) - (\Delta - z')) * \Delta^2 \geq 1 \Rightarrow V(\boldsymbol{u}) - V(\boldsymbol{u}') \geq 1.$$

For (i), if $z = z'$ then clearly $V(\boldsymbol{u}) = V(\boldsymbol{u}')$, and from the above argument it also follows that if $V(\boldsymbol{u}) = V(\boldsymbol{u}')$ then it can not be the case that $z \neq z'$. Similarly for (ii), if $V(\boldsymbol{u}) > V(\boldsymbol{u}')$ then clearly, $z' > z$, and from the above argument it follows that if $z' > z$ then it can not be the case that $V(\boldsymbol{u}') \geq V(\boldsymbol{u})$. ☐

Using the above lemma, we will next show that instance $\mathcal{E}$ has no local maximizer.

Lemma 32. *Let $\boldsymbol{u}, \boldsymbol{v} \in \mathsf{vert}$ s.t. $\boldsymbol{u} \neq \boldsymbol{v}$, $\boldsymbol{v} = S(\boldsymbol{u})$, and $\boldsymbol{u} = P(\boldsymbol{v})$. Then $V(\boldsymbol{u}) < V(\boldsymbol{v})$.*

Proof. Let $\boldsymbol{x} = (\mathbf{y}, \boldsymbol{s}, z)$ and $\boldsymbol{x}' = (\mathbf{y}', \boldsymbol{s}', z')$ be the vertices in polyhedron $\mathcal{P}$ corresponding to $\boldsymbol{u}$ and $\boldsymbol{v}$ respectively. From the construction of $\boldsymbol{v} = S(\boldsymbol{u})$ implies that $z' < z$. Therefore, using Lemma 31 it follows that $V(\boldsymbol{v}) < V(\boldsymbol{u})$. ☐

Due to Lemma 32 the only type of solutions available in $\mathcal{E}$ is where $S(P(\boldsymbol{u})) \neq \boldsymbol{u}$ and $P(S(\boldsymbol{u})) \neq \boldsymbol{u}$. Next two lemmas shows how to construct solutions of $\mathcal{I}$ from these.

Lemma 33. *Let $\boldsymbol{u} \in \mathsf{vert}$, $\boldsymbol{u} \neq 0^n$. If $P(S(\boldsymbol{u})) \neq \boldsymbol{u}$ or $S(P(\boldsymbol{u})) \neq \boldsymbol{u}$, then IsValid$(\boldsymbol{u}) = 1$, and for $(\mathbf{y}, \boldsymbol{s}, z) =$ EtoI$(\boldsymbol{u})$ if $z = 0$ then $\mathbf{y}$ is a Q1 type solution of* P-LCP *instance $\mathcal{I} = (M, \boldsymbol{q})$.*

Proof. By construction, if IsValid$(\boldsymbol{u}) = 0$, then $S(P(\boldsymbol{u})) = \boldsymbol{u}$ and $P(S(\boldsymbol{u})) = \boldsymbol{u}$, therefore IsValid$(\boldsymbol{u}) = 0$ when $\boldsymbol{u}$ has a predecessor or successor different from $\boldsymbol{u}$. Given this, from Lemma 29 we know that $(\mathbf{y}, \boldsymbol{s}, z)$ is a feasible vertex in (3). Therefore, if $z = 0$ then using Lemma 28 we have a solution of the LCP (1), *i.e.,* a type Q1 solution of our P-LCP instance $\mathcal{I} = (M, \boldsymbol{q})$. ☐

Lemma 34. *Let $\boldsymbol{u} \in \mathsf{vert}$, $\boldsymbol{u} \neq 0^n$ such that $P(S(\boldsymbol{u})) \neq \boldsymbol{u}$ or $S(P(\boldsymbol{u})) \neq \boldsymbol{u}$, and let $\boldsymbol{x} = (\mathbf{y}, \boldsymbol{s}, z) =$ EtoI$(\boldsymbol{u})$. If $z \neq 0$ then $\boldsymbol{x}$ has a duplicate label, say l. And for directions σ_1 and σ_2 obtained by relaxing $y_l = 0$ and $s_l = 0$ respectively at $\boldsymbol{x}$, we have $\sigma_1(z) * \sigma_2(z) \geq 0$, where $\sigma_i(z)$ is the coordinate corresponding to z.*

Proof. From Lemma 33 we know that IsValid$(\boldsymbol{u}) = 1$, and therefore from Lemma 29, $\boldsymbol{x}$ is a feasible vertex in (3). From the last line of Tables 4 and 6 observe that $S(\boldsymbol{u})$ points to the configuration of vertex next to $\boldsymbol{x}$ on Lemke's path only if it has lower z value otherwise it gives back $\boldsymbol{u}$, and similarly $P(\boldsymbol{u})$ points to the previous only if value of z increases.

First consider the case when $P(S(\boldsymbol{u})) \neq \boldsymbol{u}$. Let $\boldsymbol{v} = S(\boldsymbol{u})$ and corresponding vertex in $\mathcal{P}$ be $(\mathbf{y}', \boldsymbol{s}', z') = \text{EtoI}(\boldsymbol{v})$. If $\boldsymbol{v} \neq \boldsymbol{u}$, then from the above observation we know that $z' > z$, and in that case again by construction of P we will have $P(\boldsymbol{v}) = \boldsymbol{u}$, contradicting $P(S(\boldsymbol{u})) \neq \boldsymbol{u}$. Therefore, it must be the case that $\boldsymbol{v} = \boldsymbol{u}$. Since $z \neq 0$ this happens only when the next vertex on Lemke path after $\boldsymbol{x}$ has higher value of z (by above observation). As a consequence of $\boldsymbol{v} = \boldsymbol{u}$, we also have $P(\boldsymbol{u}) \neq \boldsymbol{u}$. By construction of P this implies for $(\mathbf{y}'', \boldsymbol{s}'', z'') = \text{EtoI}(P(\boldsymbol{u}))$, $z'' > z$. Putting both together we get increase in z when we relax $y_l = 0$ as well as when we relax $s_l = 0$ at $\boldsymbol{x}$.

For the second case $S(P(\boldsymbol{u})) \neq \boldsymbol{u}$ similar argument gives that value of z decreases when we relax $y_l = 0$ as well as when we relax $s_l = 0$ at $\boldsymbol{x}$. The proof follows. ☐

Finally, we are ready to prove our main result of this section using Lemmas 32, 33 and 34. Together with Lemma 34, we will use the fact that on Lemke path z monotonically decreases if M is a P-matrix or else we get a witness that M is not a P-matrix [13].

Theorem 35. P-LCP *reduces to* EndOfPotentialLine *in polynomial-time.*

Proof. Given an instance of $\mathcal{I} = (M, \boldsymbol{q})$ of P-LCP, where $M \in \mathbb{R}^{d \times d}$ and $\boldsymbol{q} \in \mathbb{R}^{d \times 1}$ reduce it to an instance $\mathcal{E}$ of EndOfPotentialLine as described above with vertex set $\mathsf{vert} = \{0,1\}^{2d}$ and procedures S, P and V as given in Table 4, 6, and 5 respectively.

Among solutions of EndOfPotentialLine instance $\mathcal{E}$, there is no local potential maximizer, i.e., $\boldsymbol{u} \neq \boldsymbol{v}$ such that $\boldsymbol{v} = S(\boldsymbol{u})$, $\boldsymbol{u} = P(\boldsymbol{v})$ and $V(\boldsymbol{u}) > V(\boldsymbol{v})$ due to Lemma 32. We get a solution $\boldsymbol{u} \neq 0$ such that either $S(P(\boldsymbol{u})) \neq \boldsymbol{u}$ or $P(S(\boldsymbol{u})) \neq \boldsymbol{u}$, then by Lemma 33 it is valid configuration and has a corresponding vertex $\boldsymbol{x} = (\mathbf{y}, \boldsymbol{s}, z)$ in $\mathcal{P}$. Again by Lemma 33 if $z = 0$ then $\mathbf{y}$ is a Q1 type solution of our P-LCP instance $\mathcal{I}$. On the other hand, if $z > 0$ then from Lemma 34 we get that on both the two adjacent edges to $\boldsymbol{x}$ on Lemke path the value of z either increases or deceases. This gives us a minor of M which is non-positive [13], i.e., a Q2 type solution of the P-LCP instance $\mathcal{I}$. ☐

C The Full Reduction and Proofs for Section 4

C.1 Slice Restrictions of Contraction Maps

Before we begin the reduction, we first fix some notation. Our algorithm and reduction for contraction maps will make heavy use of the concept of a *slice restriction* of a contraction map, which we describe here. First, for any $d \in \mathbb{N}$, we define the set of *slices* $\mathrm{Slice}_d = ([0,1] \cup \{*\})^d$ to be vectors of length d each component of which is either a number in $[0,1]$ or the special symbol $*$ which indicates that corresponding component is free to vary. With each slice $\mathbf{s} \in \mathrm{Slice}_d$ we associate a hyperplane $H(\mathbf{s}) = \left\{x \in \mathbb{R}^d \mid x_i = s_i \text{ for } s_i \neq *\right\}$. We define the set of fixed coordinates of a slice $\mathbf{s} \in \mathrm{Slice}_d$, $\mathrm{fixed}(\mathbf{s}) = \{i \in [d] \mid s_i \neq *\}$ and the set of free coordinates, $\mathrm{free}(\mathbf{s}) = [d] \setminus \mathrm{fixed}(s)$. A slice for which $|\mathrm{free}(\mathbf{s})| = i$ will be called an *i-slice*.

We can define the *slice restriction* of a function $f : [0,1]^d \to [0,1]^d$ with respect to a slice $\mathbf{s} \in \mathrm{Slice}_d$, denoted $f_{|\mathbf{s}}$, to be the function obtained by fixing the coordinates $\mathrm{fixed}(\mathbf{s})$ according to $\mathbf{s}$, and keeping the coordinates of $\mathrm{free}(\mathbf{s})$ as arguments. To simplify usage of $f_{|\mathbf{s}}$ we'll formally treat $f_{|\mathbf{s}}$ as a function with d arguments, where the coordinates in $\mathrm{fixed}(\mathbf{s})$ are ignored. Thus, we define $f_{|\mathbf{s}} : [0,1]^d \to [0,1]^d$ by

$$f_{|\mathbf{s}}(x) = f(y) \quad \text{where } y_i = \begin{cases} s_i & \text{if } i \in \mathrm{fixed}(\mathbf{s}) \\ x_i & \text{if } i \in \mathrm{free}(\mathbf{s}). \end{cases}$$

Let $\mathrm{free}(\mathbf{s}) = \{i_1, \ldots, i_k\}$. We'll also introduce a variant of $f_{|\mathbf{s}}$ when we want to consider the slice restriction as a lower-dimensional function, $\tilde{f}_{|\mathbf{s}} : [0,1]^d \to [0,1]^{|\mathrm{free}(\mathbf{s})|}$ defined by

$$\tilde{f}_{|\mathbf{s}}(x) = \left(\left(f_{|\mathbf{s}}(x)\right)_{i_1}, \ldots, \left(f_{|\mathbf{s}}(x)\right)_{i_k} \right).$$

We can also define slice restrictions for vectors in the natural way:

$$\left(x_{|\mathbf{s}}\right)_i = \begin{cases} s_i & \text{if } s_i \neq * \\ x_i & \text{otherwise.} \end{cases}$$

Finally, we'll use $\tilde{x}_{|\mathbf{s}}$ to denote projection of x onto the coordinates in $\mathrm{free}(\mathbf{s})$:

$$\tilde{x}_{|\mathbf{s}} = (x_{i_1}, \ldots, x_{i_k}).$$

We now extend the definition of a contraction map to a slice restriction of a function in the obvious way. We say that $\tilde{f}_{|\mathbf{s}}$ is a contraction map with respect to a norm $\|\cdot\|$ with Lipschitz constant c if for any $x, y \in [0,1]^d$ we have

$$\left\| \tilde{f}_{|\mathbf{s}}(x) - \tilde{f}_{|\mathbf{s}}(y) \right\| \leq c \left\| \tilde{x}_{|\mathbf{s}} - \tilde{y}_{|\mathbf{s}} \right\|.$$

Slice restrictions will prove immensely useful through the following observations:

Lemma 36. *Let $f : [0,1]^d \to [0,1]^d$ be a contraction map with respect to $\|\cdot\|_p$ with Lipschitz constant $c \in (0,1)$. Then for any slice $\mathbf{s} \in \mathrm{Slice}_d$, $f_{|\mathbf{s}}$ is also a contraction map with respect to $\|\cdot\|_p$ with Lipschitz constant c.*

Proof. For any two vectors $x, y \in [0,1]^d$ we have

$$\begin{aligned} \left\| \tilde{f}_{|\mathbf{s}}(x) - \tilde{f}_{|\mathbf{s}}(y) \right\|_p &\leq \left\| f(x_{|\mathbf{s}}) - f(y_{|\mathbf{s}}) \right\|_p \\ &< c \left\| x_{|\mathbf{s}} - y_{|\mathbf{s}} \right\|_p \\ &= c \left\| \tilde{x}_{|\mathbf{s}} - \tilde{y}_{|\mathbf{s}} \right\|_p \end{aligned}$$

□

Since slice restrictions of contraction maps are themselves contraction maps in the sense defined above, they have unique fixpoints, up to the coordinates of the argument which are fixed by the slice and thus ignored. We'll nevertheless refer to the *unique fixpoint of a slice restriction of a contraction map*, which is the unique point $x \in [0,1]^d$ such that

$$\tilde{f}_{|\mathbf{s}}(x) = \tilde{x}_{|\mathbf{s}} \quad \text{and} \quad x = x_{|\mathbf{s}}\,.$$

Lemma 37. *Let $f : [0,1]^d \to [0,1]^d$ be a contraction map with respect to $\|\cdot\|_p$ with Lipschitz constant $c \in (0,1)$. Let $\mathbf{s}, \mathbf{s}' \in \text{Slice}_d$ be such that $\text{fixed}(\mathbf{s}') = \text{fixed}(\mathbf{s}) \cup \{i\}$ and $s_j = s'_j$ for all $j \in \text{fixed}(\mathbf{s})$. Let $x, y \in [0,1]^d$ be the unique fixpoints of $\tilde{f}_{|\mathbf{s}}$ and $\tilde{f}_{|\mathbf{s}'}$, respectively. Then $(x_i - y_i)(f(y)_i - y_i) > 0$.*

Proof. We'll prove this by contradiction. Without loss of generality, assume towards a contradiction that $x_i \le y_i$ and that $f(y)_i > y_i$. Then we have

$$\begin{aligned}
\|f(y) - f(x)\|_p^p &= \|(f(y)_1, \dots, f(y)_d) - (f(x)_1, \dots, f(x)_d)\|_p^p \\
&= \sum_{j \in \text{fixed}(\mathbf{s})} |s_j - s_j|^p + \sum_{j \in \text{free}(\mathbf{s}')} |y_j - x_j|^p + |f(y)_i - x_i| \\
&> \sum_{i \in \text{fixed}(\mathbf{s})} |s_j - s_j|^p + \sum_{j \in \text{free}(\mathbf{s}')} |y_j - x_j|^p + |y_i - x_i| \\
&= \|y - x\|_p^p
\end{aligned}$$

which contradicts the fact that f is a contraction map. The lemma follows. □

C.2 Discretizing the problem.

We will turn PL-Contraction into a discrete problem by overlaying a grid of points on the $[0,1]^d$ cube. For each integer k, let $I_k = \{0, 1/k, 2/k, \dots, k/k\}$ be the discretization of the interval $[0,1]$ into points of the form x/k. Given a tuple $(k_1, k_2, \dots, k_d)$ of values, where k_i specifies the desired grid width for dimension k, we define the set of points $P = I_{k_1} \times I_{k_2} \times \cdots \times I_{k_d}$. Observe that every element of P is a point $p \in [0,1]^d$ where p_i is a rational with denominator k_i.

We will frequently refer to subsets of P in which some of the dimensions have been fixed, and so we specialize the set Slice_d that was defined in Section C.1 for this task. Throughout this section we take $\text{DSlice}_d \subset \text{Slice}_d$ to be the subset of possible slices that align with the grid defined by P. More formally, an element $s \in \text{Slice}_d$ is in DSlice_d if and only if whenever $s_i \ne *$, we have that $s_i = x/k_i$. For every slice $s \in \text{DSlice}_d$, we define the set of points $P_s \subseteq P$, to be the subset of points that lie on the slice s.

Having discretized the space, we now also discretize the function f. A *direction function* for a dimension $i \in \{1, 2, \dots, n\}$ is a function $D : P \to \{\mathsf{up}, \mathsf{down}, \mathsf{zero}\}$ that satisfies the following property. Let p^1 and p^2 be two points in P that differ only in dimension i, and suppose that $p^1_i > p^2_i$. Both of the following conditions must hold.

- If $D(p^1) \in \{\mathsf{up}, \mathsf{zero}\}$ then $D(p^2) = \mathsf{up}$.
- If $D(p^2) \in \{\mathsf{down}, \mathsf{zero}\}$ then $D(p^1) = \mathsf{down}$.

Figure 3 illustrates two direction functions for a two-dimensional problem. The figure on the left shows a direction function for the up-down dimension, while the figure on the right shows a direction function for the left-right dimension. Each square in the figures represent a point in the discretized space, and the value of the direction function is shown inside the box.

The discrete contraction problem. Suppose that we have a set $\mathcal{D}$ of direction functions, such that for each dimension i, there a function $D_i \in \mathcal{D}$ that is a direction function for dimension i. We say that a point $p \in P$ is a *fixpoint* of $\mathcal{D}$ if $D_i(p) = \mathsf{zero}$ for all i. Furthermore, for each slice $s \in \text{DSlice}_d$, we say that a point $p \in P_s$ is a *fixpoint of* s if $D_i(p) = \mathsf{zero}$ for all i for which $s_i = *$.

We say that a slice $s \in \text{DSlice}_d$ is an *i-slice* if, for every $j \leq i$ we have that $s_j = *$, and for every $j > i$ we have $s_j \neq *$.

Definition 38 (Discrete Contraction Map). *We say that $\mathcal{D}$ is a discrete contraction map if, for every i-slice s, the following conditions hold.*

1. *There is a unique fixpoint of s.*
2. *Let $s' \in \text{DSlice}_d$ be a sub-slice of s where some coordinate i for which $s_i = *$ has been fixed to a value, and all other coordinates are unchanged. If q is the unique fixpoint of s, and p is the unique fixpoint of s', then*
   - *if $p_i < q_i$, then $D_i(p) = \mathsf{up}$, and*
   - *if $p_i > q_i$, then $D_i(p) = \mathsf{down}$.*

The first condition ensures that every slice has a unique fixpoint, and is the discrete analogue of the property proved in Lemma 36. Note that taking $s = (*, *, \dots, *)$ in the above implies that $\mathcal{D}$ has a unique fixpoint. The second condition is a technical condition that we will use in our reduction, and is the discrete analogue of the property proved in Lemma 37.

The discrete contraction problem is to find the fixpoint of $\mathcal{D}$.

Definition 39 (Discrete-Contraction). *The input to the problem is a set $\mathcal{D}$ of direction functions, each of which is defined by a boolean circuit. It is promised that each function in $\mathcal{D}$ is a direction function, and that $\mathcal{D}$ satisfies the properties of a discrete contraction map. The task is to find the point $p \in P$ that is the fixpoint of $\mathcal{D}$.*

From PL-Contraction to Discrete-Contraction. We can show that PL-Contraction can be reduced, in polynomial time, to Discrete-Contraction. To do so, we take an instance of contraction, defined by the function f, and produce a set $\mathcal{D}_f$. For each dimension $i \leq n$ we define the function $D_i \in \mathcal{D}_f$ so that, for each point $p \in P$, we have:

- if $f(p)_i > p$ then $D_i(p) = \mathsf{up}$,
- if $f(p)_i < p$ then $D_i(p) = \mathsf{down}$, and
- if $f(p)_i = p$ then $D_i(p) = \mathsf{zero}$.

In other words, the function D_i simply checks whether $f(p)$ moves up, down, or not at all in dimension i. In the rest of this section, we will provide a proof for the following lemma.

Lemma 40. *If f is a contraction map, then $\mathcal{D}_f$ is a discrete contraction map.*

Intuitively, the two required properties are consequences of Lemmas 36 and 37. The main difficulty of the proof is finding suitable values for the tuple $(k_1, k_2, \dots, k_d)$ so that every slice actually has a unique fixpoint in the set P. We prove that such a tuple exists, and we rely on the fact that f is defined by a LinearFIXP circuit, and so has rational fixpoints of polynomial bit-length.

LinearFIXP circuits and LCPs. The goal of this proof is to find a tuple $(k_1, k_2, \dots, k_d)$ such that the lemma holds. We utilize a lemma, which was shown in [48], which proves that that every

LinearFIXP circuit can be transformed, in polynomial time, into an LCP, such that the solution of the LCP captures the fixpoint of the circuit, and gives a bound on the bit-length of the numbers used in the LCP.

Throughout this proof, we will use the function $b(x)$ to denote, for each rational x, the bit-length of the representation of x, which is the bit-length needed to represent the numerator and denominator of x. When we apply b to a matrix or a vector, we take the maximum of the bit-lengths of the elements of that matrix or vector.

Lemma 41 ([48]). *Let C be a* LinearFIXP *circuit, and let n be the number of* max *and* min *gates used in C. We can produce, in polynomial time, an LCP defined by an $n \times n$ matrix M_C and n-dimensional vector $\boldsymbol{q}_C$ every fixpoint of C is a solution of the LCP defined by M and $\mathbf{q}$ and vice versa. Furthermore, if size(C) denotes*

$$(\#inputs\ + \#gates\ +\ number\ of\ bits\ needed\ to\ represent\ the\ constants\ used),$$

then $b(M_C)$ and $b(\boldsymbol{q}_C)$ are both at most $O(n \times size(C))$.

Crucially, if C' denotes a circuit where one of the inputs of C is fixed to be some number x, then $b(M_{C'}) \leq b(M_C)$ and $b(\boldsymbol{q}_{C'}) \leq b(x) + b(M_C) + b(\boldsymbol{q}_C)$. In other words, the bit-length of $M_{C'}$ does not depend on x, and is in fact at most the bit-length of M_C.

Bounding the bit-length of a solution of an LCP. We now prove two technical lemmas about the bit-length of any solution to the LCP. We begin with the following lemma regarding the bit-length of a matrix inverse.

Lemma 42. *Let $A \in \mathbb{Q}^{n\times n}$ be an square matrix of full rank, and let the largest absolute value of an entry of A be denoted by B, i.e., $|A_{ij}| \leq B$ for all i, j. Let p/q for $p, q \in \mathbb{Z}$ denote an arbitrary entry of A^{-1}. Then we have $\max(p, q) \leq B^n n^{n/2}$.*

Proof. We have that $A^{-1} = \frac{1}{\det(A)}(\text{adjoint}(A))$. Entries of adjoint$(A)$ are $\pm$ times the determinant of a square sub-matrix of A of size $n-1$, which this also has a bound of B on the absolute value of any entry. It is a well-known corollary of Hadamard's inequality that $|\det(A)| \leq B^n n^{n/2}$. Applying this bound to entries of adjoint(A) for p and $\det(A)$ for q gives the result. ☐

We now use this to prove the following bound on the bit-length of a fixpoint of an LCP.

Lemma 43. *Let M and $\boldsymbol{q}$ be an LCP, and $\mathbf{y}$ be the solution to the LCP. We have that $b(\mathbf{y}) \leq n \cdot \log n + 3n \cdot b(M) + b(\boldsymbol{q})$.*

Proof. We first note that if an LCP has a solution, then it has a vertex solution. Let $\mathbf{y}$ be a solution of the LCP defined by $M = [M_1, \ldots, M_n]$ and $\boldsymbol{q}$, and let $\mathbf{w} = \boldsymbol{q} - M \cdot \mathbf{y}$ so that $I \cdot \mathbf{w} + M \cdot \mathbf{y} = \boldsymbol{q}$ where I is the identity matrix $[e_1, \ldots, e_n]$. By definition, for each i, we either have that $\mathbf{y}_i = 0$ or that $\mathbf{w}_i = (\boldsymbol{q} - M \cdot \mathbf{y})_i = 0$. Let $\alpha = \{i \mid \mathbf{y}_i = 0\}$. We define the matrix $A = [A_1, \ldots, A_n]$ from unit vectors and the columns of M as follows:

$$A_i = \begin{cases} e_i, & \textbf{if } i \in \alpha, \\ M_i, & \textbf{if } i \notin \alpha. \end{cases}$$

If the vertex is non-degenerate, then whenever $\mathbf{y}_i = 0$ we have $\mathbf{w}_i > 0$, and therefore A is guaranteed to be invertible. In case of a degenerate vertex, note that we take e_i for the ith column in A even when $\mathbf{y}_i$ and $\mathbf{w}_i$ both are zero. This will ensure that A is invertible.

Then, the LCP conditions, i.e., the complementary condition, the equation $I \cdot \mathbf{w} + M \cdot \mathbf{y} = \boldsymbol{q}$, and the non-negativity of $\mathbf{y}$ and $\mathbf{w}$, are equivalent to the existence of a non-negative vector $\boldsymbol{x}$ that satisfies

$$A \cdot \boldsymbol{x} = \boldsymbol{q} \ .$$

Then $\mathbf{y}_i = \boldsymbol{x}_i$ for $i \notin \alpha$ and $\mathbf{y}_i = 0$ for $i \in \alpha$, so we have $b(\mathbf{y}) \leq b(\boldsymbol{x})$. Also note that we have $b(A) \leq b(M)$, since the entries in columns that take the value of e_i have constant bit-length.

We must transform A into an integer matrix in order to apply Lemma 42. Let l denote the least common multiple of the entries in A. Note that $l \leq n^2 \cdot 2^{b(A)}$ and hence $b(l) \leq b(A) + 2\log(n)$. Our matrix equation above can be rewritten as $l \cdot A \cdot \boldsymbol{x} = l \cdot q$, and note that $(l \cdot A)$ is an integer matrix. Hence we have $\boldsymbol{x} = (l \cdot A)^{-1} \cdot l \cdot \boldsymbol{q}$.

If B denotes the maximum entry of $(l \cdot A)$, then $B \leq l \cdot 2^{b(A)}$. So by Lemma 42 the maximum entry of $(l \cdot A)^{-1}$ is

$$(l \cdot 2^{b(A)})^n \cdot n^{n/2} \leq (n^2 \cdot 2^{2b(A)})^n \cdot n^{n/2}$$

Each entry of $\mathbf{y}$ consists of the sum of n entries of $(l \cdot A)^{-1}$ each of which is multiplied by an entry of $\boldsymbol{q}$, and finally the sum is multiplied by l. So we get the following bound on the bit-length of $\boldsymbol{q}$.

$$\begin{aligned} b(\mathbf{y}) &\leq \log\left(n \cdot 2^{b((l \cdot A)^{-1})} \cdot 2^{b(\boldsymbol{q})} \cdot 2^{b(l)}\right) \\ &\leq \log\left(n \cdot \left((n^2 \cdot 2^{2b(A)})^n \cdot n^{n/2}\right) \cdot 2^{b(\boldsymbol{q})} \cdot \left(n^2 \cdot 2^{b(A)}\right)\right) \\ &= \log\left(n^{n/2+5} \cdot 2^{(2n+1)b(A)} \cdot 2^{b(\boldsymbol{q})}.\right) \\ &\leq n \cdot \log n + 3n \cdot b(A) + b(\boldsymbol{q}). \end{aligned}$$

□

Fixing the grid size. We shall fix the grid size iteratively, starting with k_d, and working downwards. Note that there is exactly one d-slice, which is $s = (*, *, \ldots, *)$. We fix $k_d = 2^{n \cdot \log n + 3n \cdot b(M_C) + b(\boldsymbol{q}_C)}$, and recall that the set I_{k_d} contains all rationals with denominators at most k_d. Lemma 43 implies that the fixpoint of s, which is a solution of the LCP defined by M_C and $\boldsymbol{q}_C$, is a rational with denominator at most k_d.

Now let $s = (*, *, \ldots, *, x)$ be a $(d-1)$-slice where x is a member of I_{k_d}, meaning that x is a point with denominator k_d. Recall that if we fix the dth input of C to be x, then we obtain a smaller circuit C', and we have that $b(M_{C'}) \leq b(M_C)$. However, x will be added to some elements of q, and so $b(\boldsymbol{q}_{C'}) \leq b(k_d)$. If $\exp(x) = 2^x$, then we fix

$$\begin{aligned} k_{d-1} &= \exp\left(n \cdot \log n + 3n \cdot b(M_{C'}) + b(\boldsymbol{q}_{C'})\right) \\ &\leq \exp\left(2n \cdot \log n + 6n \cdot b(M_C) + b(\boldsymbol{q}_C)\right). \end{aligned}$$

Again, Lemma 43 implies that the fixpoint of s is a rational with denominator at most k_{d-1}.

Repeating the above argument for all $i \leq d-1$ leads us to set

$$k_i = \exp\left((d-i+1)n \cdot \log n + 3 \cdot (d-i+1) \cdot n \cdot b(M_C) + b(\boldsymbol{q}_C)\right).$$

By the same reasoning, we get that for each i-slice s, the unique fixpoint of s is a rational with denominator at most k_i.

Observe that the maximum bit-length of the numbers k_i is attained by the number k_1, which has bit-length at most

$$d \cdot n \cdot \log n + 3d \cdot n \cdot b(M_C) + b(\boldsymbol{q}_C).$$

By Lemma 41, this is polynomial in the size of C.

Completing the proof of Lemma 40. Now that we have fixed the grid size, we must show that the two properties of a discrete contraction map hold.

The first property requires that each i-slice has a unique fixpoint in the grid. We have specifically chosen the grid to ensure that each i-slice has a fixpoint in the grid, and if the grid had two fixpoints for a given i-slice, then we would immediately obtain a contradiction using Lemma 36. The second property follows immediately from Lemma 37.

C.3 Proof of Lemma 11

We will formally define our line by induction over the dimensions. We will start by defining a line that specifies the sequence of points on the $(n-1)$-surface. In each step of the induction, we will assume that we have defined the visited points on the j-surface for all $j > i$, and we will specify the points on the i-surface that are visited by the line.

For each step of our reduction, we will define a *partial* UNIQUEFORWARDEOPL instance $L_i = (C_i, S_i, V_i)$ which captures the points of the full line that are on the i-surface. The reason that we call them partial instances, is because the successor function S_i that we define will not actually point to the next vertex in the line, since it is not computable in polynomial time. It will instead give the next point on the full line. In all other respects, the instance L_i is a valid instance of UNIQUEFORWARDEOPL, so all circuits will be polynomial-time computable, the potential will be monotonically increasing, and the line will end at the fixpoint of the discrete contraction instance. More formally, the instance L_i will satisfy the following conditions, which will serve as our inductive hypothesis.

1. There will a polynomial-time circuit Point, which given a vertex v of L_i, will return the point in P that corresponds to v.
2. For every vertex v on the line, the point $p = \text{Point}(v)$ is on the i-surface.
3. For every vertex v on the line, let u denote the next vertex on the line. If $p = \text{Point}(S_i(v))$ and $q = \text{Point}(u)$, then $p_j = q_j$ for all $j \geq i$.
4. For the first vertex v on the line, the point $p = \text{Point}(v)$ has $p_j = 0$ for all $j \geq i$.
5. For the last vertex v on the line, the point $p = \text{Point}(v)$ is a solution to the discrete contraction problem.
6. The potential function is monotonically increasing along the line.

Condition 3 above specifies the behavior of the successor circuit. It specifies that, if we ignore dimensions 1 through $i-1$, then the line is connected.

The base case. In the base-case, we define the instance $L_{d-1} = (C_{d-1}, S_{d-1}, V_{d-1})$ in the following way. For each slice $s_x = (*, *, \dots, *, x) \in \text{DSlice}_d$, Property 1 of a discrete contraction map ensures that there is a unique point $p \in P_s$ that is a fixpoint of s, and so is on the $(d-1)$-surface. Our line will consist of these points. It will start at the unique fixpoint of s_0, it will then move to the unique fixpoint of s_1, and then to the unique fixpoint of s_2, and so on until it reaches the unique fixpoint of $\mathcal{D}$. Obviously, when we are at s_1, we cannot easily compute s_2, but our conditions do not require us to do so. We are simply required to produce a point that agrees with s_2 in dimension d, which we can trivially do by moving one step positively in dimension d from s_1.

We can use Property 2 of a discrete contraction map to determine whether a point on the $(d-1)$-surface is on this line. This property states that, if q is the fixpoint of $\mathcal{D}$, then $D_d(p) = \mathsf{up}$ if and only if $q_d > p_d$. So the points on our line will be exactly the points p on the $(d-1)$-surface for which $D_d(p) \in \{\mathsf{up}, \mathsf{zero}\}$.

Formally, we define L_{d-1} in the following way. Each vertex of L_{d-1} will be a point $p \in P$, and so we can trivially define the circuit $\text{Point}(p) = p$.

- For each point $p \in P$ we have that $C_{d-1}(p) = 1$ if and only if both of the following conditions hold.
  - p is on the $(d-1)$-surface.
  - $D_d(p) \in \{\mathsf{up}, \mathsf{zero}\}$.
- For each point $p \in P$ for which $C_{d-1}(p) = 1$, we define $S_{d-1}(p)$ as follows.
  - If p is $D_d(p) = \mathsf{zero}$, then p is the end of the line.
  - Otherwise, $S_{d-1}(p) = p'$, where $p'_d = p_d + 1$, and $p'_i = p_i$ for all $i < d$.
- For each point $p \in P$ for which $C_{d-1}(p) = 1$, we define $V_{d-1}(p) = p_d$.

The successor function gives, for each point p that is on the line, the point directly above p in dimension d. This point agrees with the next point on the line (which is the unique fixpoint of the slice defined by $p_d + 1$) in dimension d, and so satisfies our definition. The potential function simply uses coordinate d of each point, which is monotonically increasing along the line.

Lemma 44. *The instance $(C_{d-1}, S_{d-1}, V_{d-1})$ satisfies Conditions 1 through 6 of the inductive hypothesis.*

Proof. We must check that all conditions of the inductive hypothesis hold.

- Condition 1 requires that the function Point is computable in polynomial time. This is trivially true.
- Condition 2 requires that every point p is on the $(d-1)$-surface, which is enforced by the circuit C_{d-1}.
- Condition 3 requires that if v is a vertex on the line, and u is the next vertex on the line, then the point $p = \text{Point}(S_{d-1}(v))$ agrees with $q = \text{Point}(u)$ in dimension d. This is true, since the dth coordinate of $S_{d-1}(p)$ is $p_d + 1$, and this agrees with the dth coordinate of q.
- Condition 4 requires that the first point on the line has zero in the dth coordinate. This is true by definition.
- Condition 5 requires that the last vertex on the line is a solution to the discrete contraction problem. This is true, since by definition, the line ends at a point p that is on the $(d-1)$-surface that satisfies $D_d(p) = \mathsf{zero}$. Hence, the point satisfies $D_i(p) = \mathsf{zero}$ for all i, and so it is a solution to the discrete contraction problem.
- Condition 6 requires that the potential function monotonically increases along the line. This is true because the potential function of each point is the final coordinate of that point. Since the line passes through each slice s_x until it finds the solution, and since it always moves from the slice s_x to the slice s_{x+1}, we have that the final coordinate of the points on the line monotonically increases.

□

The inductive step. Suppose that we have an instance $L_{i+1} = (C_{i+1}, S_{i+1}, V_{i+1})$ that satisfies the inductive hypothesis. We will now describe how to build the instance $L_i = (C_i, S_i, V_i)$.

Let v and u be two adjacent vertices on the line defined by L_{i+1}. By the inductive hypothesis, the point $p = \text{Point}(S_{i+1}(v))$ and the point $q = \text{Point}(u)$ agree on dimensions $i+1$ through n, but may not agree on dimension i. When we construct L_i, we will fill in this gap, by introducing a new sequence of points that join p with q.

Each of these new points will be in the i-surface. Let s_j define the slice $(*, *, \dots, j, p_{i+1}, p_{i+2}, \dots, p_d)$, ie., the slice where coordinate i is equal to j, and coordinates $i+1$ through d agree with p (and, by the inductive hypothesis, q). Let r^j be the unique fixpoint of s_j, which by definition is a point on the i-surface. If $D_i(p) = \mathsf{up}$, then our line will go through the sequence $r^{p_i}, r^{p_i+1}, \dots, r^{q_i}$, and if $D_i(p) = \mathsf{down}$, then our line will go through the sequence $r^{p_i}, r^{p_i-1}, \dots, r^{q_i}$. This is analogous to how we linked two points on the one-surface by a sequence of points on the zero-surface in our two-dimensional example.

We also introduce a new sequence of points at the start of the line. Let v_{init} be the first vertex of the line L_{i+1}. By the inductive hypothesis, we have that $\text{Point}(v_{\text{init}})$ is the point on the $i+1$-surface that is zero in dimensions $i+1$ through d. We create a new sequence of points starting at the point on the i-surface that is zero in dimensions i through d, and ending at $\text{Point}(v_{\text{init}})$. This line is constructed in the same way as the other lines, by taking the unique i-witness of each slice $(*, *, \dots, j, 0, 0, \dots, 0)$, where j appears in coordinate i of the slice.

As we saw in the two-dimensional example, we need to remember the point p in order to determine whether any given point r^j is on this line. So, a vertex of L_i will consist of a pair (v, r), where v is either

- v is a vertex of L_{i+1}, or
- the special symbol $-$,

and $r \in P$ is a point. For each vertex (v, r) we define $\text{Point}(v, r) = r$, which can clearly be computed in polynomial time. The symbol $-$ will be used in the initial portion of the line, where we have not yet arrived at the first vertex of L_{i+1}.

The circuit $C_i(v, r)$ is defined as follows.

- If $v \neq -$, then the circuit returns 1 if and only if all of the following hold. Let $u = \text{Point}(S_{i+1}(v))$.
  - $C_{i+1}(v) = 1$.
  - r is on the i-surface.
  - r and u agree on coordinates $i+1$ through d.
  - If $D_i(u) = \mathsf{up}$ then $D_i(r) \in \{\mathsf{up}, \mathsf{zero}\}$ and $r_i \geq u_i$.
  - If $D_i(u) = \mathsf{down}$ then $D_i(r) \in \{\mathsf{down}, \mathsf{zero}\}$ and $r_i \leq u_i$.
- If $v = -$, then the circuit returns 1 if and only if all of the following hold.
  - $r_j = 0$ for all $j > i$.
  - r is on the i-surface.
  - $D_i(r) \in \{\mathsf{up}, \mathsf{zero}\}$.

The first bullet specifies the line between $\text{Point}(v)$ and $\text{Point}(S_{i+1}(v))$, while the second bullet specifies the line between the initial vertex of L_i, and the initial vertex of L_{i+1}.

For each pair (v, r) such that $C_i(v, r) = 1$, the circuit $S_i(v, r)$ is defined in the following way.

- If $D_i(r) = \mathsf{up}$, then the circuit returns the vertex (v, r') such that $r'_i = r_i + 1$, and $r'_j = r_j$ for all $j \neq i$.
- If $D_i(r) = \mathsf{down}$, then the circuit returns the vertex (v, r') such that $r'_i = r_i - 1$, and $r'_j = r_j$ for all $j \neq i$.
- If $D_i(r) = \mathsf{zero}$, then the circuit performs the following operation.
  - First it modifies v so that $\text{Point}(v) = r$. This is valid because r is an i-witness with $D_i(r) = \mathsf{zero}$, and therefore r is an $i+1$-witness. Let u denote this new vertex of L_i.
  - Then it computes $r' = S_{i+1}(u)$.
  - Finally, it outputs the vertex (u, r').

The first two items simply follow the direction function D_i. The third item above is more complex. Once we arrive at the point with $D_i(r) = \mathsf{zero}$, we have found the $i+1$-witness that we are looking for, and this is the next point on the line L_{i+1}. So we use this fact to ask S_{i+1} for the next vertex of L_{i+1}, and begin following the next line.

For each pair (v, r) such that $C_i(v, r) = 1$, the circuit $V_i(v, r)$ is defined in the following way.

- If $v \neq -$ and $D_i(r) = \mathsf{up}$, then $V_i(v, r) = (k_i + 1) \cdot (V_{i+1}(v) + 1) + r_i$.
- If $v \neq -$ and $D_i(r) = \mathsf{down}$, then $V_i(v, r) = (k_i + 1) \cdot (V_{i+1}(v) + 1) + k_i - r_i$.
- If $v = -$, then $V_i(v, r) = r_i$.

This definition generalizes the definition that we gave in our two-dimensional example.

Lemma 45. *The instance (C_i, S_i, V_i) satisfies Conditions 1 through 6 of the inductive hypothesis.*

Proof. We prove each condition in turn.

- Condition 1 requires that the circuit Point is computable in polynomial time, which is clearly true.
- Condition 2 requires that every point on the line is on the i-surface. The definition of C_i ensures that this is true.
- Let (v, r) and (u, t) be two adjacent vertices on the line. Condition 3 requires that we have that $r' = \text{Point}(S_i(v, r))$ agrees with t on all dimensions $j \geq i$. This follows because, in all three cases in the definition of S_i, we have that r' is a member of the slice $s_t = (*, *, \dots, *, t_{i+1}, t_{i+1}, \dots, t_d)$, and (u, t) is the unique i-witness of the slice s_t.
- Condition 4 requires that the first vertex v on the line have $\text{Point}(v)_j = 0$ for all $j \geq i$. This is true by definition.
- Condition 5 requires that the last vertex on the line is the solution to the discrete contraction problem. This is implied by the inductive hypothesis, because L_i ends at the last vertex of L_{i+1}.
- Condition 6 requires that the potential function is monotonically increasing along the line. To see this, recall that k_i is the grid width of dimension i. In the definition of V_i, We multiply $V_{i+1}(v)$ by $k_i + 1$ to ensure that there are at least k_i gaps in the potential function between each vertex of L_{i+1}. We fill these gaps using the ith coordinate of r_i, which is either monotonically increasing (in the first case) or monotonically decreasing (in the second case). We also shift the potentials $V_{i+1}(v)$ by 1 to give enough space for the initial path before the first vertex of L_{i+1}.

□

Completing the proof. To complete the proof of Lemma 11, it suffices to note that the instance (C_1, S_1, V_1) is a (non-partial) instance of UniqueForwardEOPL. The inductive hypothesis guarantees that the line is connected, unique, and ends at the unique fixpoint of $\mathcal{D}$. Furthermore, the potential is monotonically increasing along the line.

C.4 Proof of Lemma 13

From UniqueForwardEOPL to unique forward EndOfMeteredLine. We first reduce our UniqueForwardEOPL instance to a unique forward version of EndOfMeteredLine. To avoid introducing another distinct line following problem, we can define unique forward EndOfMeteredLine to be a UniqueForwardEOPL instance where $V(S(v)) = V(v) + 1$ for every vertex v that is on the line. For the reduction from UniqueForwardEOPL, we can use exactly the same proof that we used to reduce EndOfPotentialLine to EndOfMeteredLine.

Recall that the reduction does the following operation. Let v be a vertex in the UniqueForwardEOPL instance, and let $u = S(v)$. If $V(u) \neq V(v) + 1$, that is if the potential does not increase by exactly one between u and v, then let $p = V(u) - V(v)$ be the difference in potentials between the two vertices. If p is negative then we are the end of the line. Otherwise, we introduce $p - 1$ new vertices between u and v, each of which increases the potential by exactly 1.

Intuitively, there is no need to use the predecessor circuit in order to perform this operation. This can be confirmed by inspecting the proof of Theorem 3, and noting that the predecessor circuit is not used in a way that is relevant for us. Specifically, in the procedure defining the successor function S', the predecessor circuit is only used in Step 5, where it is used to check that $P(S(v)) = v$, and hence to determine whether v is the end of the line. In UniqueForwardEOPL, we can use C to perform this task, and so we do not need to use the predecessor circuit at all.

Note also that, as long as the promise for UniqueForwardEOPL is indeed true, then the unique forward EndOfMeteredLine instance will contain exactly one line.

From unique forward EndOfMeteredLine to SinkOfVerifiableLine. Let V be the set of vertices in the unique forward EndOfMeteredLine instance, and let (C, S, V) be the circuits used to define the instance. Let p be a number that is strictly larger than the largest possible potential that V can produce.

The set of vertices in the SinkOfVerifiableLine instance will be pairs (v, i), where $v \in V$ is a vertex, and $i \leq p$ is either an integer, or the special symbol $-$. The idea is that each vertex on the line will be represented by the vertex $(v, -)$. If v is not the end of the line, then we move to $(S(v), -)$.

However, we must make sure that the end of the line is at a known distance from the start of the line, and we use the second element to ensure this. Once we arrive at the vertex u that is the end of the line, we then move to $(u, 1)$, and then to $(u, 2)$, and so on, until we reach $(u, p - V(u))$. This ensures that the end of the line is exactly p steps from the start of the line.

We now formally define the SinkOfVerifiableLine instance. The start vertex will be the start vertex of the unique forward EndOfMeteredLine instance, and the target integer will be p. The circuit S' is defined for a vertex (v, i) so that

- If $C(v) = 1$ and $C(S(v)) = 1$, meaning that v is not the end of the line, and $i = -$, then the circuit returns $(S(v), -)$.
- If $C(v) = 1$ and $C(S(v)) = 0$, meaning that v is the end of the line, then

– If $i = -$ then the circuit returns $(v, 1)$.

– If $i \neq -$ and $i \leq p$, then the circuit returns $(v, i+1)$.

In all other cases the vertex is not on the line, so the value returned by S' is irrelevant. The circuit W' is defined for a vertex (v, i) and an integer k so that

- If $C(v) = 1$ and $i = -$, then the circuit returns 1 if and only and $V(v) = k$.
- If $C(v) = 1$, $i \neq -$, and $C(S(v)) = 0$, then the circuit returns 1 if and only if $k = V(v) + i$ and $V(v) + i \leq p$.

In all other cases the circuit returns 0. Note that, since the potential increases by exactly one at each step of our line, the circuit W does indeed correctly identify the vertices on the line by their distance from the start vertex.

Note that if the unique forward EndOfMeteredLine instance has a unique line, then the promise of the SinkOfVerifiableLine problem is indeed satisfied.

Completing the proof of Lemma 13. To complete the proof, it suffices to apply the result of Hubáček and Yogev [36], which shows that SinkOfVerifiableLine can be reduced to EndOfMeteredLine using a technique of Bitansky, Paneth and Rosen [3]. Moreover, it can be verified that, so long as the promise of the SinkOfVerifiableLine instance is satisfied, then the proof of Hubáček and Yogev produces an EndOfMeteredLine instance that has exactly one line. Therefore, we have reduced UniqueForwardEOPL to unique EndOfPotentialLine.

D The Algorithms and Proofs for Section 5

In this section, we provide an exact algorithm for solving PL-Contraction, which is a promise problem guaranteed to have a rational fixpoint of polynomial bit complexity. Then we extend this algorithm to find an approximate fixpoint of general contraction maps for which there may not be an exact solution of polynomial bit complexity. Our algorithms work for any ℓ_p norm with $p \in \mathbb{N}$, and are polynomial for constant dimension d. These are the first such algorithms for $p \neq 2$. Such algorithms were so far only known for the ℓ_2 and ℓ_∞ norms [34, 57, 58][3]

D.1 Overview: algorithm to find a fixed-point of PL-Contraction

The algorithm does a nested binary search using Lemmas 36 and 37 to find fixpoints of slices with increasing numbers of free coordinates. We illustrate the algorithm in two dimensions in Figure 5. The algorithm is recursive. To find the eventual fixpoint in d dimensions we fix a single coordinate s_1, find the unique $(d-1)$-dimensional fixpoint of $\tilde{f}_{|\mathbf{s}}$, the $(d-1)$-dimensional contraction map obtained by fixing the first coordinate of the input to f to be s_1. Let x the unique fixpoint of $\tilde{f}_{|\mathbf{s}}$ where $x_1 = s_1$. If $f(x_1) > s_1$, then the d-dimensional fixpoint x^* of f has $x_1^* > s_1$, and if $f(x_1) < s_1$, then $x_1^* < s_1$ (Lemma 37). We can thus do a binary search for the value of x_1^*. Once we've found x_1^*, we can recursively find the $(d-1)$-dimensional fixpoint of $\tilde{f}_{|\mathbf{s}}$ where $s_1 = x_1$. The resulting solution will be the d-dimensional fixpoint. At each step in the recursive procedure, we do a binary search for the value of one coordinate of the fixpoint at the slice determined by all the coordinates already fixed. For piecewise-linear functions, we know that all fixpoints are rational with bounded bit complexity (as discussed in Section C.2), so we can find each coordinate exactly.

Using this algorithm we obtain the following theorem.

[3]Our approach does not cover the ℓ_∞ norm, as that would require more work and not give a new result.

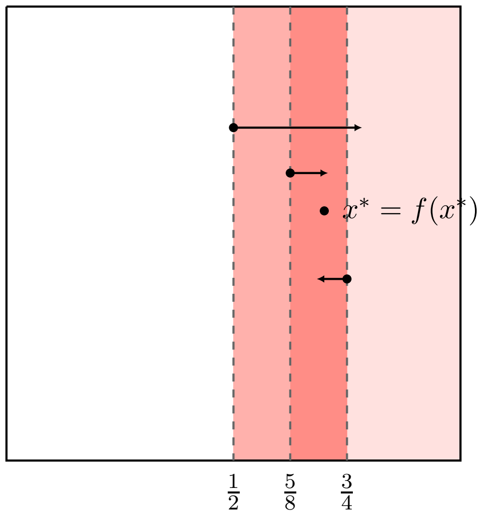


Figure 5: An illustration of the algorithm to find a fixpoint of a piecewise-linear contraction map in two dimensions. The algorithm begins by finding a fixpoint along the slice with $x_1 = 1/2$. The fixpoint along that slice points to the right, so we next find a fixpoint along the slice with $x_1 = 3/4$. The fixpoint along that slice points to the left, so we find the fixpoint along $x_1 = 5/8$. We successively find fixpoints of one-dimensional slices, and then use those to do a binary search for the two-dimensional fixpoint. The red regions are the successive regions considered by the binary search, where each successive step in the binary search results in a darker region.

Theorem 46. *Given a* LinearFIXP *circuit C encoding a contraction map $f : [0,1]^d \to [0,1]^d$ with respect to any ℓ_p norm, there is an algorithm to find a fixpoint of f in time $O(L^d)$ where $L = \text{poly}(|C|)$ is an upper bound on the bit-length of the fixpoint of f.*

The full details of the algorithm can be found in Appendix D.3.

D.2 Overview: algorithm to find an approximate fixed-point of Contraction

Here we generalize our algorithm to find an approximate fixpoint of an arbitrary function given by an arithmetic circuit, i.e., our algorithm solves Contraction, which is specified by a circuit f that represents the contraction map,[4] a p-norm, and ε. Again, let d denote the dimension of the problem, i.e. the number of inputs (and outputs) of f. Let x^* denote the unique exact fixpoint for the contraction map f. We seek an approximate fixpoint, i.e., a point for which $\|f(x) - x\|_p \leq \varepsilon$.

We do the same recursive binary search as in the algorithm above, but at each step of the algorithm instead of finding an exact fixpoint, we will only find an approximate fixpoint of $\tilde{f}_{|\mathbf{s}}$. The difficulty in this case will come from the fact that Lemma 37 does not apply to approximate fixpoints. Consider the example illustrated in Figure 6. In this example, y is the unique fixpoint of the slice restriction along the gray dashed line. By Lemma 37, $(f(y)_1 - y_1)(x^*_1 - y_1) \geq 0$ so if we find y, we can observe that $f(y)_1 > y_1$ and recurse on the right side of the figure, in the region labeled $\mathcal{R}$. If we try to use the same algorithm but where we only find approximate fixopints at each step, we'll run into trouble. In this case, if we found z instead of y, we would observe that

[4]The algorithm works even if f is given as an arbitrary black-box, as long as it is guaranteed to be a contraction map.

$f(z)_1 < z_1$ and conclude that $x_1^* < z_1$, which is incorrect. As a result, we would limit our search to the region labeled $\mathcal{L}$, and wouldn't be able to find x^*.

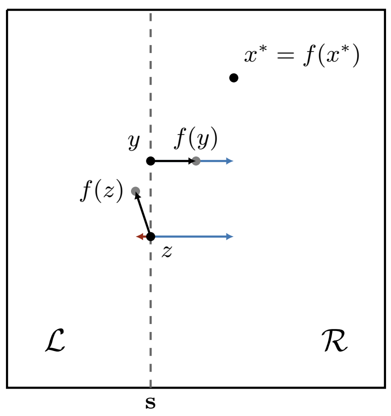


Figure 6: A step in the recursive binary search. Here, x^* is the fixpoint for the original function, y is the fixpoint for the slice restriction $\tilde{f}_{|\mathbf{s}}$ along the dashed gray line, and z is an approximate fixpoint to the slice restriction.

When looking for an approximate fixpoint, we'll have to choose a different precision ε_i for each level of the recursion so that either the point x returned by the ith recursive call to our algorithm satisfies $|f(x)_i - x_i| > \varepsilon_i$ and we can rely on it for pivoting in the binary search, or $|f(x)_i - x_i| \le \varepsilon_i$ and we can return x as an approximate fixpoint to the recursive call one level up. Each different ℓ_p norm will require a different choice of $(\varepsilon_i)_{i=1}^d$.

Using this idea we are able to obtain the following results:

Theorem 47. *For a contraction map $f : [0,1]^d \to [0,1]^d$ with respect to the ℓ_1 norm, there is an algorithm compute a point $v \in [0,1]^d$ such that $\|f(v) - v\|_1 < \varepsilon$ in time $O(d^d \log(1/\varepsilon))$.*

Theorem 48. *For a contraction map $f : [0,1]^d \to [0,1]^d$ under $\|\cdot\|_p$ for $2 \le p < \infty$, there is an algorithm to compute a point $v \in [0,1]^d$ such that $\|f(v) - v\|_p < \varepsilon$ in time $O(p^{d^2} \log^d(1/\varepsilon) \log^d(p))$.*

The full details of the algorithms can be found in Appendix D.4.

D.3 Details: finding a fixed-point of PL-Contraction

Suppose f is piecewise-linear; it follows that the coordinates of the unique fixpoints of $\tilde{f}_{|\mathbf{s}}$ will be rational numbers with bounded denominators. Consider the values of k_i's computed in Section C.2. The analysis in the section tells us that if we consider a slice $\mathbf{s}$ fixing any i coordinates to numbers with denominators $k_d, \dots, k_{d-i+1}$ then the unique fixed-point of $\tilde{f}_{|\mathbf{s}}$ will have coordinates with denominator bounded by k_{d-i}. Furthermore, k_1 is the largest among all k_i's and its bit-length is bounded by polynomial in the size of the circuit C representing the given PL-Contraction instance.

Let $\mathrm{Slice}_d(\mathbf{h})$ be the set of slices with fixed coordinates having rational values with denominator h_i in the ith coordinate:

$$\mathbf{s} \in \mathrm{Slice}_d(\mathbf{h}) \iff \mathbf{s} \in \mathrm{Slice}_d \text{ and } s_i \in \{0/h_i, 1/h_i, \dots, h_i/h_i\}, \ \forall i \in \mathrm{fixed}(\mathbf{s}).$$

Algorithm 1 Algorithm for PL-Contraction

```
1: Input: A slice $\mathbf{s} \in \text{Slice}_d(2^{L_1}, \ldots, 2^{L_d})$ where $\text{fixed}(\mathbf{s}) = [k]$ for some $k \leq d$.
2: Output: A point $y$ such that $\tilde{f}_{|\mathbf{s}}(y) = \tilde{y}_{|\mathbf{s}}$ and $y = y_{|\mathbf{s}}$.
3: function FindFP($\mathbf{s}$)
4:     Let $k = |\text{fixed}(\mathbf{s})|$.
5:     if $k = d$ then return $\mathbf{s}$
6:     end if
7:     Set $k \leftarrow k + 1$. Set $\mathbf{t}^l \leftarrow \mathbf{s}$, $\mathbf{t}^h \leftarrow \mathbf{s}$.
8:     Set $t^l_k \leftarrow 0$, $t^h_k \leftarrow 1$.
9:     Set $v^l \leftarrow$ FindFP($\mathbf{t}^l$), and $v^h \leftarrow$ FindFP($\mathbf{t}^h$).
10:    if $f(v^l)_k = t^l_k$ then return $v^l$.
11:    end if
12:    if $f(v^h)_k = t^h_k$ then return $v^h$.
13:    end if
14:    Set $\mathbf{t} \leftarrow \mathbf{s}$
15:    while $t^h_k - t^l_k > \frac{1}{2^{(L_k+1)}}$ do
16:        Set $t_k \leftarrow \frac{(t^h_k + t^l_k)}{2}$.
17:        Set $v \leftarrow$ FindFP($\mathbf{t}$).
18:        if $f(v)_k = t_k$ then return $v$.
19:        end if
20:        if $f(v)_k > t_k$ then Set $t^l_k \leftarrow t_k$
21:        else Set $t^h_k \leftarrow t_k$.
22:        end if
23:    end while
24:    $t_k \leftarrow$ unique number in $(t^l_k, t^h_k)$ with denominator at most $2^{L_k}$.
25:    return FindFP($\mathbf{t}$).
26: end function
```

From the above discussion, define for each

$$i \in \{1, \ldots, d\}, \quad L_i = \ln(k_{d-i+1}).$$

We will design an algorithm assuming that upper bound of 2^{L_i} on the ith coordinate denominators of fixpoints for any slice restriction $\tilde{f}_{|\mathbf{s}}$ where $\mathbf{s} \in \text{Slice}_d(2^{L_1}, \ldots, 2^{L_d})$.

Analysis. Given a $k < d$ and $\mathbf{s} \in \text{Slice}_d(2^{L_1}, \ldots, 2^{L_d})$ with $\text{fixed}(\mathbf{s}) = [k]$, from Lemma 36 we know that the restricted function $\tilde{f}_{|\mathbf{s}}$ is also contracting. We will show that Algorithm 1 computes a fixpoint of $\tilde{f}_{|\mathbf{s}}$. Since the algorithm is recursive, we will prove its correctness by induction. The next lemma establishes the base case of induction and follows by design of the algorithm and is equivalent to finding a fixpoint of a one dimensional function.

Lemma 49. *If* $|\text{fixed}(\mathbf{s})| = d - 1$ *then* FindFP($\mathbf{s}$) *returns a* v *such that* $v_i = s_i$ *for* $i \in \text{fixed}(\mathbf{s})$ *and* $f(v)_d = v_d$.

Now for the inductive step, assuming FindFP can compute a fixpoint of any q-dimensional contraction map, we will show that it can compute one for $(q + 1)$-dimensional contraction map.

Lemma 50. *Fix some* $\mathbf{s} \in \text{Slice}_d(2^{L_1}, \ldots, 2^{L_d})$ *with* $\text{fixed}(\mathbf{s}) = [k-1]$ *for some* $k - 1 < d$, *and let* $\mathbf{t} = (s_1, s_2, \ldots, s_{k-1}, t_k, *, \ldots, *)$ *for some* $t_k \in \{0/2^{L_k}, 1/2^{L_k}, \ldots, 2^{L_k}/2^{L_k}\}$. *If* FindFP($\mathbf{t}$) *returns* *the unique fixpoint of the restricted function* $\tilde{f}_{|\mathbf{t}}$*, then* FINDFP($\mathbf{s}$) *returns the unique fixpoint of the function* $\tilde{f}_{|\mathbf{s}}$.

Proof. Since f is a contraction map, so is $\tilde{f}_{|\mathbf{s}}$ due to Lemma 36. The induction hypothesis ensures that for any value of $t_k \in \{0/2^{L_1}, 1/2^{L_k}, \ldots, 2^{L_k}/2^{L_k}\}$ if $v = \text{FINDFP}(\mathbf{t})$ then v is the unique fixpoint of the function $\tilde{f}_{|\mathbf{t}}$, i.e., $f(v)_j = t_j$ for $j \in \text{free}(\mathbf{t})$ and $v = v_{|t}$. Now, if $t_k = 0$ then $f(v)_k \geq 0 = t_k = v_k$ and if $t_k = 1$ then $f(v)_k \leq 1 = t_k = v_k$. If either is an equality then v is the unique fixpoint of $\tilde{f}_{|\mathbf{s}}$ as well and the lemma follows.

Otherwise, we know that the kth coordinate of the fixpoint of $\tilde{f}_{|\mathbf{s}}$, which we'll call t^*_k, is between $t^l_k = 0$ and $t^h_k = 1$. Note that when t_k is set to t^*_k, the vector $v = \text{FINDFP}(\mathbf{t})$ will be the unique fixpoint of $\tilde{f}_{|\mathbf{s}}$, since both $\tilde{f}_{|\mathbf{t}}$ and $\tilde{f}_{|\mathbf{s}}$ have unique fixpoints. Thus, it suffices to show that t_k will eventually be set to t^*_k during the execution of the algorithm.

The while loop of Algorithm 1 does a binary search between t^h_k and t^l_k to find t^*_k, while keeping track of the fixpoints of $\tilde{f}_{|\mathbf{t}}$. We first observe that after line 17 of each execution of the loop in Algorithm 1, v is the unique fixpoint of $\tilde{f}_{|\mathbf{t}}$. Therefore, whenever $f(v)_k = t_k$ is satisfied, we will return the unique fixpoint of $\tilde{f}_{|\mathbf{s}}$ and the lemma follows.

The binary search maintains the invariant that if we let $v^l = \text{FINDFP}(t^l)$ and $v^h = \text{FINDFP}(t^h)$ we have $f(v^l)_k > v^l_k$, and $f(v^h)_k < v^h_k$. By Lemma 37, this invariant ensures that t^*_k satisfies

$$t^l_k < t^*_k < t^h_k$$

at all times. Therefore, binary search either returns the desired fixpoint or ends with t^l_k and t^h_k such that $(t^h_k - t^l_k) \leq 1/2^{L_k+1}$ and $t^*_k \in [t^l_k, t^h_k]$. By the assumption we know that t^*_k is a rational number with denominator at most 2^{L_k}. Since there can be at most one such number in $[l_k, h_k]$, t^*_k can be uniquely identified. And therefore the last line of Algorithm 1 returns a fixpoint of $\tilde{f}_{|\mathbf{s}}$. ☐

Applying induction using Lemma 49 as a base-case and Lemma 50 as an inductive step, the next theorem follows.

Theorem 51. FINDFP$(*, *, \ldots, *)$ *returns the unique fixpoint of* f *in time* $O(L^d)$ *where* $L = \max_k L_k$ *is polynomial in the size of the input instance.*

D.4 Details: finding an approximate fixed-point of CONTRACTION

Each different ℓ_p norm will require a different choice of $(\varepsilon_i)_{i=1}^d$. There are two distinct cases to consider for $(\varepsilon_i)_{i=1}^d$:

- For the $p = 1$, we choose $\varepsilon_i = \varepsilon/2^{2i}$.
- For $2 \leq p < \infty$, we choose $\varepsilon_i = \varepsilon^{p^i} p^{-2\sum_{j=0}^{i} p^j}$.

We now prove lemmas showing that these choices of $(\varepsilon_i)_{i=1}^d$ allow us to make progress at each step of the algorithm. In both of the following lemmas, let $\mathbf{s} \in \text{Slice}_d$ with $\text{fixed}(\mathbf{s}) = [k-1]$ for $k - 1 < d$.

Lemma 52. *Let* f *be a contraction map with respect to* $\|\cdot\|_1$*, and let* $\varepsilon_i = \varepsilon/2^i$*. Let* x^* *be the unique fixpoint of* $\tilde{f}_{|\mathbf{s}}$*. Given any point* $v \in [0,1]^d$ *with* $|f(v)_i - v_i| \leq \varepsilon_i$ *for all* $i > k$*, either* $|f(v)_k - v_k| \leq \varepsilon_k$ *or* $(f(v)_k - v_k)(x^*_k - v_k) > 0$.

Proof. Assume that v satisfies $|f(v)_i - v_i| \leq \varepsilon_i$ for all $i > k$, and $|f(v)_k - v_k| > \varepsilon_k$. Without loss of generality, let $f(v)_k - v_k > \varepsilon_k$. Assume towards a contradiction that $x^*_k \leq v_k$. Then

$$\begin{aligned}
\left\| \tilde{f}_{|\mathbf{s}}(v) - \tilde{x^*}_{|\mathbf{s}} \right\|_1 &= \sum_{j=k}^{d} |f(v)_j - x^*_j| \\
&> \varepsilon_k + |v_k - x^*_k| + \sum_{j=k+1}^{d} |f(v)_j - x^*_j| \\
&\geq \varepsilon_k + |v_k - x^*_k| + \sum_{j=k+1}^{d} \left[|v_j - x^*_j| - \varepsilon_j \right] \\
&= \varepsilon_k - \sum_{j=k+1}^{d} \varepsilon_j + \left\| \tilde{v}_{|\mathbf{s}} - \tilde{x^*}_{|\mathbf{s}} \right\|_1 \\
&> \left\| \tilde{v}_{|\mathbf{s}} - \tilde{x^*}_{|\mathbf{s}} \right\|_1 .
\end{aligned}$$

The last line uses the fact that $\varepsilon/2^{2k} \geq \sum_{j=k+1}^{d} \varepsilon/2^{2j}$ for all $k \leq d$. We now have $\left\| \tilde{f}_{|\mathbf{s}}(v) - \tilde{x^*}_{|\mathbf{s}} \right\|_1 \geq \left\| \tilde{v}_{|\mathbf{s}} - \tilde{x^*}_{|\mathbf{s}} \right\|_1$ which contradicts $\tilde{f}_{|\mathbf{s}}$ being a contraction map. This completes the proof. $\square$

Lemma 53. *Let f be a contraction map with respect to $\|\cdot\|_p$, and let $\varepsilon_i = \varepsilon^{p^j} p^{-\sum_{j=0}^{i} p^j}$. Let x^* be the unique fixpoint of $\tilde{f}_{|\mathbf{s}}$. Given any point $v \in [0,1]^d$ with $|f(v)_i - v_i| \leq \varepsilon_i$ for all $i > k$, either $|f(v)_k - v_k| \leq \varepsilon_k$ or $(f(v)_k - v_k)(x^*_k - v_k) > 0$.*

Proof. Assume that v satisfies $|f(v)_i - v_i| \leq \varepsilon_i$ for all $i > k$, and $|f(v)_k - v_k| > \varepsilon_k$. Without loss of generality, let $f(v)_k - v_k > \varepsilon_k$. Assume towards a contradiction that $x^*_k \leq v_k$. Then

$$\begin{aligned}
\left\| \tilde{f}_{|\mathbf{s}}(v) - \tilde{x^*}_{|\mathbf{s}} \right\|_p^p &= \sum_{j=k}^{d} |f(v)_j - x^*_j|^p && (6) \\
&> \varepsilon_k^p + |v_k - x^*_k|^p + \sum_{j=k+1}^{d} |f(v)_j - x^*_j|^p && (7) \\
&\geq \varepsilon_k^p + |v_k - x^*_k|^p + \sum_{j=k+1}^{d} \left(|v_j - x^*_j| - \varepsilon_j \right)^p && (8) \\
&= \varepsilon_k^p + \sum_{j=k}^{d} |v_k - x^*_k|^p - \sum_{j=k+1}^{d} \left[|v_j - x^*_j|^p - \left(|v_j - x^*_j| - \varepsilon_j \right)^p \right] && (9) \\
&= \left\| \tilde{v}_{|\mathbf{s}} - \tilde{x^*}_{|\mathbf{s}} \right\|_p^p + \varepsilon_k^p - \sum_{j=k+1}^{d} \left[|v_j - x^*_j|^p - \left(|v_j - x^*_j| - \varepsilon_j \right)^p \right] && (10) \\
&\geq \left\| \tilde{v}_{|\mathbf{s}} - \tilde{x^*}_{|\mathbf{s}} \right\|_p^p + \varepsilon_k^p - \sum_{j=k+1}^{d} p\varepsilon_j && (11) \\
&> \left\| \tilde{v}_{|\mathbf{s}} - \tilde{x^*}_{|\mathbf{s}} \right\|_p^p . && (12) \\
& && (13)
\end{aligned}$$

Line 11 uses the fact that $a^p - (a-b)^p \leq 1 - (1-b)^p$ for $p > 1$, $a, b \in (0,1)$. Line 12 follows from our choice of $(\varepsilon_i)_{i=1}^d$; we have

$$\begin{aligned}
\sum_{j=k+1}^{d} p\varepsilon_j &= \sum_{j=k+1}^{d} p\varepsilon^{p^j} p^{-2\sum_{j=0}^{i} p^i + p} \\
&< \sum_{j=k+1}^{d} \varepsilon^{p^{k+1}} p^{-2\sum_{j=1}^{i} p^i - p} \\
&< \varepsilon^{p^{k+1}} p^{-2\sum_{j=1}^{k+1} p^i - p} \left[\sum_{j=1}^{k+1} p^{-2p} \right] \\
&< \varepsilon_k^p.
\end{aligned}$$

Again we have a contradiction since $\left\| \tilde{f}_{|\mathbf{s}}(v) - \tilde{x^*}_{|\mathbf{s}} \right\|_1 \geq \left\| \tilde{v}_{|\mathbf{s}} - \tilde{x^*}_{|\mathbf{s}} \right\|_1$ and $\tilde{f}_{|\mathbf{s}}$ is a contraction map. The lemma follows. $\square$

We will also need to establish one more lemma for each case:

Lemma 54. *Let f be a contraction map with respect to $\|\cdot\|_1$ and let $\varepsilon_i = \varepsilon/2^{2i}$. Let x^* be the unique fixpoint of $\tilde{f}_{|\mathbf{s}}$ and let $t_k^h, t_k^l \in [0,1]$ be such that $t_k^h - t_k^l < \varepsilon_k/2$ and the unique fixpoint x^* of $\tilde{f}_{|\mathbf{s}}$ satisfies*

$$t_k^l < x^* < t_k^h.$$

Let $v^l, v^h \in [0,1]^d$ be such that $v^l = v^l_{|\mathbf{s}}$ and $v_k^l = t_k^l$, and $v^h = v^h_{|\mathbf{s}}$ and $v_k^h = t_k^h$. Furthermore, assume that $\left|f(v^h)_i - v_i^h\right| \leq \varepsilon_i$ and $\left|f(v^l)_i - v_i^l\right| \leq \varepsilon_i$ for all $i > k$. Then either $\left|f(v^h)_k - v_k^h\right| \leq \varepsilon_k$ or $\left|f(v^l)_k - v_k^l\right| \leq \varepsilon_k$.

Proof. We will prove this by contradiction. Assume that $\left|f(v^h)_k - v_k^h\right| > \varepsilon_k$ and $\left|f(v^l)_k - v_k^l\right| > \varepsilon_k$. We will show that $\left\| \tilde{f}_{|\mathbf{s}}(v^h) - \tilde{f}_{|\mathbf{s}}(v^l) \right\|_1 - \left\| \tilde{v^h}_{|\mathbf{s}} - \tilde{v^l}_{|\mathbf{s}} \right\|_1 > 0$, which contradicts $\tilde{f}_{|\mathbf{s}}$ being a contraction map. We have

$$\begin{aligned}
\left\| \tilde{f}_{|\mathbf{s}}(v^h) - \tilde{f}_{|\mathbf{s}}(v^l) \right\|_1 - \left\| \tilde{v^h}_{|\mathbf{s}} - \tilde{v^l}_{|\mathbf{s}} \right\|_1 &= \sum_{j=k}^{d} \left[\left| f(v^h)_j - f(v^l)_j \right| - \left| v_j^h - v_j^l \right| \right] \\
&= \left[\left| f(v^h)_k - f(v^l)_k \right| - \left| v_k^h - v_k^l \right| \right] + \sum_{j=k+1}^{d} \left[\left| f(v^h)_j - f(v^l)_j \right| - \left| v_j^h - v_j^l \right| \right] \\
&> 3\varepsilon/2^{2k+1} - \sum_{j=k+1}^{d} 2\varepsilon/2^{2j} \\
&> 0,
\end{aligned}$$

completing the proof. $\square$

Lemma 55. *Let f be a contraction map with respect to $\|\cdot\|_p$ and let $\varepsilon_i = \varepsilon^{p^i} p^{-\sum_{j=0}^{i} p^i}$. Let x^* be the unique fixpoint of $\tilde{f}_{|\mathbf{s}}$ and let $t_k^h, t_k^l \in [0,1]$ be such that $t_k^h - t_k^l < \varepsilon_k$ and the unique fixpoint x^* of $\tilde{f}_{|\mathbf{s}}$ satisfies*

$$t_k^l < x^* < t_k^h.$$

Let $v^l, v^h \in [0,1]^d$ be such that $v^l = v^l_{|\mathbf{s}}$ and $v^l_k = t^l_k$, and $v^h = v^h_{|\mathbf{s}}$ and $v^h_k = t^h_k$. Furthermore, assume that $\left|f(v^h)_i - v^h_i\right| \leq \varepsilon_i$ and $\left|f(v^l)_i - v^l_i\right| \leq \varepsilon_i$ for all $i > k$. Then either $\left|f(v^h)_k - v^h_k\right| \leq \varepsilon_k$ or $\left|f(v^l)_k - v^l_k\right| \leq \varepsilon_k$.

Proof. We will prove this by contradiction. Assume that $\left|f(v^h)_k - v^h_k\right| > \varepsilon_k$ and $\left|f(v^l)_k - v^l_k\right| > \varepsilon_k$. We will show that $\left\|\tilde{f}_{|\mathbf{s}}(v^h) - \tilde{f}_{|\mathbf{s}}(v^l)\right\|_p^p - \left\|\tilde{v^h}_{|\mathbf{s}} - \tilde{v^l}_{|\mathbf{s}}\right\|_p^p > 0$, which contradicts $\tilde{f}_{|\mathbf{s}}$ being a contraction map. We have

$$\begin{aligned}
\left\|\tilde{f}_{|\mathbf{s}}(v^h) - \tilde{f}_{|\mathbf{s}}(v^l)\right\|_p^p - \left\|\tilde{v^h}_{|\mathbf{s}} - \tilde{v^l}_{|\mathbf{s}}\right\|_p^p &= \sum_{j=k}^{d} \left[\left|f(v^h)_j - f(v^l)_j\right|^p - \left|v^h_j - v^l_j\right|^p\right] && (14)\\
&= \left[\left|f(v^h)_k - f(v^l)_k\right|^p - \left|v^h_k - v^l_k\right|^p\right] && (15)\\
&\quad + \sum_{j=k+1}^{d} \left[\left|f(v^h)_j - f(v^l)_j\right|^p - \left|v^h_j - v^l_j\right|^p\right]\\
&\geq \left[\left|f(v^h)_k - f(v^l)_k\right| - \left|v^h_k - v^l_k\right|\right]^p && (16)\\
&\quad + \sum_{j=k+1}^{d} \left[\left(\left|v^h_j - v^l_j\right| - 2\varepsilon_j\right)^p - \left|v^h_j - v^l_j\right|^p\right]\\
&\geq (3\varepsilon_k/2)^p - \sum_{j=k+1}^{d} 2p\varepsilon_j && (17)\\
&> \varepsilon_k^p - \sum_{j=k+1}^{d} p\varepsilon_j && (18)\\
&> 0. && (19)
\end{aligned}$$

Here, line 17 mirrors line 11 from Lemma 53 (which holds for $\varepsilon_i < 1/2$, which is the case here). Line 18 uses the fact that $3/2^p > 2$ for $p > 2$. Finally, line 19 uses the same argument as in line 12 in Lemma 53. □

We now proceed to prove the correctness of our algorithm.

Analysis. We will show that for any norm $\|\cdot\|$ and ε-sequence for $\|\cdot\|$, Algorithm 2 returns a point $v \in [0,1]^d$ such that $\|f(v) - v\| \leq \varepsilon$. To do this, we will show that for any $k < d$ and slice $\mathbf{s} \in \text{Slice}_d$ with $\text{fixed}(\mathbf{s}) = [k]$, the point v returned by ApproxFindFP($\mathbf{s}$) will satisfy

$$|f(v)_i - v_i| \leq \varepsilon_i, \ \forall i \in \text{free}(\mathbf{s}).$$

Since Algorithm 2 is recursive, our proof will be by induction. The next lemma establishes the base case of the induction and follows by design of the algorithm.

Lemma 56. *If $|\text{fixed}(\mathbf{s})| = d - 1$ then* FindFP($\mathbf{s}$) *returns a v such that $|f(v)_d - v_d| \leq \varepsilon_d$ and $v = v_{|\mathbf{s}}$.*

Now for the inductive step, we show that if ApproxFindFP can compute an approximate fixpoint of any q-dimensional contraction map, we will show that it can compute one for the $(q+1)$-dimensional contraction map instances it is given.

Algorithm 2 Algorithm for CONTRACTION

```
1: Input: A slice $\mathbf{s} \in \text{Slice}_d$ such that $\text{fixed}(\mathbf{s}) = [k]$ for some $k \leq d$.
2: Output: A point $v \in [0,1]^d$ such that $|f(v)_i - v_i| \leq \varepsilon_i,\ \forall i \in \text{free}(\mathbf{s})$ and $v = v_{|\mathbf{s}}$.
3: function ApproxFindFP($\mathbf{s}$)
4:     Let $k = |\text{fixed}(\mathbf{s})|$.
5:     if $k = d$ then return $\mathbf{s}$
6:     end if
7:     Set $k \leftarrow k + 1$. Set $\mathbf{t}^l \leftarrow \mathbf{s}$, $\mathbf{t}^h \leftarrow \mathbf{s}$.
8:     Set $t^l_k \leftarrow 0$, $t^h_k \leftarrow 1$.
9:     Set $v^l \leftarrow$ ApproxFindFP($\mathbf{t}^l$), and $v^h \leftarrow$ ApproxFindFP($\mathbf{t}^h$).
10:    if $\left|f(v^l)_k - v^l_k\right| \leq \varepsilon_k$ then return $v^l$.
11:    end if
12:    if $\left|f(v^h)_k - v^h_k\right| \leq \varepsilon_k$ then return $v^h$.
13:    end if
14:    Set $\mathbf{t} \leftarrow \mathbf{s}$
15:    while $t^h_k - t^l_k > \varepsilon_k$ do
16:        Set $t_k \leftarrow \frac{(t^h_k + t^l_k)}{2}$.
17:        Set $v \leftarrow$ ApproxFindFP($\mathbf{t}$).
18:        if $|f(v)_k - v_k| \leq \varepsilon_k$ then return $v$.
19:        end if
20:        if $f(v)_k > t_k$ then Set $t^l_k \leftarrow t_k$
21:        else Set $t^h_k \leftarrow t_k$.
22:        end if
23:    end while
24:    $t_k \leftarrow \frac{(t^h_k + t^l_k)}{2}$.
25:    return ApproxFindFP($\mathbf{t}$).
26: end function
```

Lemma 57. *Fix some* $\mathbf{s} \in \text{Slice}_d$ *with* $\text{fixed}(\mathbf{s}) = [k-1]$ *for some* $k - 1 < d$*, and let* $\mathbf{t} = (s_1, s_2, \ldots, s_{k-1}, t_k, *, \ldots, *)$ *for some* $t_k \in [0,1]$*. If* ApproxFindFP($\mathbf{t}$) *returns a point* v *such that*

$$|f(v)_i - v_i| \leq \varepsilon_i,\ \forall i \in \text{free}(\mathbf{t})$$

and $v = v_{|\mathbf{t}}$*, then* ApproxFindFP($\mathbf{s}$) *returns a point such that*

$$|f(v)_i - v_i| \leq \varepsilon_i,\ \forall i \in \text{free}(\mathbf{s}).$$

Proof. Since f is a contraction map, so is $\tilde{f}_{|\mathbf{s}}$ due to Lemma 36. We assume that for any value of $t_k \in [0,1]$ if $v = \text{FindFP}(\mathbf{t})$ then v satisfies

$$|f(v)_i - v_i| \leq \varepsilon_i,\ \forall i \in \text{free}(\mathbf{t})$$

and $v = v_{|\mathbf{t}}$.

We first observe that after the first two calls to ApproxFindFP($\mathbf{t}^l$) and ApproxFindFP($\mathbf{t}^h$), if $\left|f(v^h)_k - v^h_k\right| \leq \varepsilon_k$ or $\left|f(v^l)_k - v^l_k\right| \leq \varepsilon_k$, we return v^h or v^l, respectively, so the output of ApproxFindFP($\mathbf{s}$) satisfies the requirements of the lemma.

Moreover, in every subsequent call to ApproxFindFP($\mathbf{t}$), if the output v satisfies $|f(v)_k - v_k| \leq \varepsilon_k$, we return v, so we'll assume in what follows that the value returned by the algorithm is from line 25.

In what follows let t_k^* be the value of the kth coordinate of the unique fixpoint of $f_{|\mathbf{s}}$.

The binary search in the while loop maintains the invariant that if we let $v^l = \text{FINDFP}(t^l)$ and $v^h = \text{FINDFP}(t^h)$ we have $f(v^l)_k - v_k^l > \varepsilon_k$, and $f(v^h)_k - v_k^h < -\varepsilon_k$. By Lemmas 52 and 53, this invariant ensures that t_k^* satisfies

$$t_k^l < t_k^* < t_k^h$$

at all times.

Now we just need to argue that when we get to $t_k^h - t_k^l < \varepsilon_k$, the point v returned by the final call to $\text{APPROXFINDFP}(\mathbf{t})$ will satisfy the required conditions. To see this we appeal to Lemmas 54 and 55, noting that $t_k^h - t_k < \varepsilon_k/2$ and $t_k - t_k^l < \varepsilon_k/2$ and both $\left|f(v^h)_k - v_k^h\right|, \left|f(v^l)_k - v_k^l\right| > \varepsilon_k$. It then follows that $|f(v)_k - v_k| \leq \varepsilon$ since either $t_k^l < t_k^* < t_k$ or $t_k < t_k^* < t_k^h$. ☐

Applying induction using Lemma 56 as a base-case and Lemma 57 as an inductive step, we obtain the following theorems:

Theorem 58. *For a contraction map with respect to the ℓ_1 norm,* $\text{APPROXFINDFP}(*, *, \ldots, *)$ *returns a point $v \in [0,1]^d$ such that $\|f(v) - v\|_1 < \varepsilon$ in time $O(d^d \log(1/\varepsilon))$.*

Proof. In the worst case, the ith recursive call to APPROXFINDFP will terminate with $t_i^h - t_i^l < \varepsilon_i = \varepsilon/4^i$ so will require at most $O(\log(1/\varepsilon)i)$ iterations. The total runtime will thus be bounded by $O(d^d \log^d(1/\varepsilon))$. The final point returned will satisfy

$$|f(v)_i - v_i| < \varepsilon_i, \quad \forall i \in [d]$$

which implies that

$$\|f(v) - v\| = \sum_{i=1}^{d} |f(v)_i - v_i| \leq \sum_{i=1}^{d} \varepsilon_i < \varepsilon.$$

☐

Theorem 59. *For a contraction map under $\|\cdot\|_p$ for $2 \leq p < \infty$,* $\text{APPROXFINDFP}(*, *, \ldots, *)$ *returns a point $v \in [0,1]^d$ such that $\|f(v) - v\|_p < \varepsilon$ in time $O(p^{d^2} \log^d(1/\varepsilon) \log^d(p))$.*

Proof. In the worst case, the ith recursive call to APPROXFINDFP will terminate with $t_i^h - t_i^l < \varepsilon_i = \varepsilon^{p^i} p^{-2\sum_{j=0}^{i} p^j}$ so will require $O(p^i \log(1/\varepsilon)\log(p))$ iterations. The total runtime will thus be bound by $O(p^{d^2} \log^d(1/\varepsilon) \log^d(p))$. The final point returned will satisfy

$$|f(v)_i - v_i| < \varepsilon_i, \quad \forall i \in [d]$$

which implies that

$$\|f(v) - v\|_p^p = \sum_{i=1}^{d} |f(v)_i - v_i|^p \leq \sum_{i=1}^{d} \varepsilon_i^p < \varepsilon^p.$$

☐

E MetametricContraction is CLS-Complete

In this section, we define METAMETRICCONTRACTION and show that it is CLS-complete. First, following [16], we define the complexity class CLS as the class of problems that are reducible to the following problem CONTINUOUSLOCALOPT.

Definition 60 (ContinuousLocalOpt [16])**.** *Given two arithmetic circuits computing functions $f : [0,1]^3 \to [0,1]^3$ and $p : [0,1]^3 \to [0,1]$ and parameters $\epsilon, \lambda > 0$, find either:*

(C1) a point $x \in [0,1]^3$ such that $p(x) \le p(f(x)) - \epsilon$ or

(C2) a pair of points $x, y \in [0,1]^3$ satisfying either

(C2a) $\|f(x) - f(y)\| > \lambda \|x - y\|$ or

(C2b) $\|p(x) - p(y)\| > \lambda \|x - y\|$.

In Definition 60, p should be thought of as a *potential* function, and f as a *neighbourhood* function that gives a candidate solution with better potential if one exists. Both of these functions are purported to be Lipschitz continuous. A solution to the problem is either an approximate potential minimizer or a witness for a violation of Lipschitz continuity.

Definition 61 (Contraction [16])**.** *We are given as input an arithmetic circuit computing $f : [0,1]^3 \to [0,1]^3$, a choice of norm $\|\cdot\|$, constants $c \in (0,1)$ and $\delta > 0$, and we are promised that f is c-contracting w.r.t. $\|\cdot\|$. The goal is to find*

(CM1) a point $x \in [0,1]^3$ such that $d(f(x), x) \le \delta$,

(CM2) or two points $x, y \in [0,1]^3$ such that $\|f(x) - f(y)\| / \|x - y\| > c$.

In other words, the problem asks either for an approximate fixpoint of f or a violation of contraction. As shown in [16], Contraction is easily seen to be in CLS by creating instances of ContinuousLocalOpt with $p(x) = \|f(x) - x\|$, f remains as f, Lipschitz constant $\lambda = c + 1$, and $\epsilon = (1 - c)\delta$.

In a *meta-metric*, all the requirements of a metric are satisfied except that the distance between identical points is not necessarily zero. The requirements for d to be a meta-metric are given in the following definition.

Definition 62 (Meta-metric)**.** *Let $\mathcal{D}$ be a set and $d : \mathcal{D}^2 \mapsto \mathbb{R}$ a function such that:*

1. *$d(x, y) \ge 0$;*
2. *$d(x, y) = 0$ implies $x = y$ (but, unlike for a metric, the converse is not required);*
3. *$d(x, y) = d(y, x)$;*
4. *$d(x, z) \le d(x, y) + d(y, z)$.*

Then d is a meta-metric on $\mathcal{D}$.

The problem Contraction, as defined in [16], was inspired by Banach's fixpoint theorem, where the contraction can be with respect to any metric. In [16], for Contraction the assumed metric was any metric induced by a norm. The choice of this norm (and thus metric) was considered part of the definition of the problem, rather than part of the problem input. In the following definition of MetametricContraction, the contraction is with respect to a meta-metric, rather than a metric, and this meta-metric is given as part of the input of the problem.

Definition 63 (MetametricContraction)**.** *We are given as input an arithmetic circuit computing $f : [0,1]^3 \to [0,1]^3$, an arithmetic circuit computing a meta-metric $d : [0,1]^3 \times [0,1]^3 \to [0,1]$, some p-norm $\|\cdot\|_r$ and constants $\epsilon, c \in (0,1)$ and $\delta > 0$, and we are promised that f is c-contracting with respect to d, and λ-continuous with respect to $\|\cdot\|$, and that d is γ-continuous with respect to $\|\cdot\|$. The goal is to find*

(M1) a point $x \in [0,1]^3$ such that $d(f(x), x) \leq \epsilon$,

(M2) or two points $x, y \in [0,1]^3$ such that

(M2a) $d(f(x), f(y))/d(x,y) > c$,

(M2b) $\|d(x,y) - d(x',y')\| / \|(x,y) - (x',y')\| > \delta$, or

(M2c) $\|f(x) - f(y)\| / \|x - y\| > \lambda$.

(M3) points x, y, or x, y, z in $[0,1]^3$ that witness a violation of one of the four defining properties of a meta-metric (Definition 62).

Definition 64 (GeneralContraction)**.** *The definition is identical to that of Definition 63 identical except for the fact that solutions of type (M3) are not allowed.*

So, while MetametricContraction allows violations of d being a meta-metric as solutions, GeneralContraction does not.

Theorem 65. GeneralContraction *is in* CLS.

Proof. Given an instance $X = (f, d, \epsilon, c, \lambda, \delta)$ of GeneralContraction, we set $p(x) \triangleq d(f(x), x)$. Then our ContinuousLocalOpt instance is the following:

$$Y = (f, p, \lambda' \triangleq (\lambda + 1)\delta, \epsilon' \triangleq (1-c)\epsilon).$$

Now consider any solution to Y. If our solution is of type (C1), a point x such that $p(f(x)) > p(x) - \epsilon'$, then we have $d(f(f(x)), f(x)) > d(f(x), x) - (1-c)\epsilon$, and either $d(f(x), x) \leq \epsilon$, in which case x is a solution for X, or $d(f(x), x) > \epsilon$. In the latter case, we can divide on both sides to get

$$\frac{d(f(f(x)), f(x))}{d(f(x), x)} > 1 - \frac{(1-c)\epsilon}{d(f(x), x)} \geq 1 - (1-c) = c,$$

giving us a violation of the claimed contraction factor of c, and a solution of type (M2a).

If our solution is a pair of points x, y of type (C2a) satisfying $\|f(x) - f(y)\| / \|x - y\| > \lambda' \geq \lambda$, then this gives a violation of the λ-continuity of f. If instead x, y are of type (C2b) so that $\|p(x) - p(y)\| / \|x - y\| > \lambda'$, then we have

$$|d(f(x), x) - d(f(y), y)| = |p(x) - p(y)| > (\lambda + 1)\delta \|x - y\|.$$

We now observe that if

$$|d(f(x), x) - d(f(y), y)| \leq \delta(\|f(x) - f(y)\| + \|x - y\|) \quad \text{and } \|f(x) - f(y)\| / \|x - y\| \leq \lambda,$$

then we would have

$$|d(f(x), x) - d(f(y), y)| \leq \delta(\|f(x) - f(y)\| + \|x - y\|) \leq (\lambda + 1)\delta \|x - y\|,$$

which contradicts the above inequality, so either the δ continuity of d must be violated giving a solution to X of type (M2b) or the λ continuity of f must be violated giving a solution of type (M2c). Thus we have shown that GeneralContraction is in CLS. □

Now that we have shown that GeneralContraction is total, we note that since the solutions of GeneralContraction are a subset of those for MetametricContraction, we have the following.

Observation 66. MetametricContraction *can be reduced in polynomial-time to* GeneralContraction.

Thus, by Theorem 65, we have that MetametricContraction is in CLS. Next, we show that MetametricContraction is CLS-hard by a reduction from the canonical CLS-complete problem ContinuousLocalOpt to an instance of MetametricContraction. By Observation 66, we then also have that GeneralContraction is CLS-hard.

Theorem 67. MetametricContraction *is* CLS*-hard.*

Proof. Given an instance $X = (f, p, \epsilon, \lambda)$ of ContinuousLocalOpt, we construct a meta-metric $d(x, y) = p(x) + p(y) + 1$. Since p is non-negative, d is non-negative, and by construction, d is symmetric and satisfies the triangle inequality. Finally, $d(x, y) > 0$ for all choices of x and y so d is a valid meta-metric (Definition 62) Furthermore, if p is λ-continuous with respect to the given p-norm $\|\cdot\|_r$, then d is $(2^{1/r-1}\lambda)$-continuous with respect to $\|\cdot\|_r$. For clarity, in the below proof we'll omit the subscript r when writing the norm of an expression. To see this we observe that $x, x', y, y' \in [0, 1]^n$, we have $\|p(x) - p(x')\| / \|x - x'\| \leq \lambda$ and $\|p(y) - p(y')\| / \|y - y'\| \leq \lambda$, so

$$\frac{\|d(x,y) - d(x',y')\|}{\|(x,y) - (x',y')\|} = \frac{\|p(x) - p(x') + p(y) - p(y') + 1 - 1\|}{\|(x,y) - (x',y')\|} \leq \frac{\lambda \|x - x'\| + \lambda \|y - y'\|}{\|(x,y) - (x',y')\|}$$
$$\leq \frac{\lambda \|x - x'\| + \lambda \|y - y'\|}{2^{1-1/r}(\|x - x'\| + \|y - y'\|)} \leq 2^{1/r-1}\lambda.$$

We output an instance $Y = (f, d, \epsilon' = \epsilon, c = 1 - \epsilon/4, \delta = \lambda, \lambda' = 2^{1/r-1}\lambda)$.

Now we consider solutions for the instance Y and show that they correspond to solutions for our input instance X. First, we consider a solution of type (M1), a point $x \in [0, 1]^3$ such that $d(f(x), x) \leq \epsilon' = \epsilon$. We have $p(f(x)) + p(x) + 1 \leq \epsilon$, but this can't happen since $\epsilon < 1$ and p is non-negative, so solutions of this type cannot exist.

Now consider a solution that is a pair of points $x, y \in [0, 1]^3$ satisfying one of the conditions in (M2). If the solution is of type (M2a), we have $d(f(x), f(y)) > cd(x, y)$, and by our choice of c this is exactly

$$\frac{d(f(x), f(y))}{d(x,y)} > (1 - \epsilon/4)$$

and

$$\begin{aligned} p(f(x)) + p(f(y)) + 1 &> (1 - \epsilon/4)(p(x) + p(y) + 1) \\ &\geq p(x) + p(y) - 3\epsilon/4 \end{aligned}$$

so either $p(f(x)) > p(x) - \epsilon$ or $p(f(y)) > p(y) - \epsilon$, and one of x or y must be a fixpoint solution to our input instance. Solutions of type (M2b) or (M2c) immediately give us violations of the λ-continuity of f, and thus solutions to X.

This completes the proof that MetametricContraction is CLS-hard. □

So combining these results we have the following.

Theorem 68. MetametricContraction *and* GeneralContraction *are* CLS*-complete.*

Finally, as mentioned in the introduction, we note the following. Contemporaneously and independently of our work, Daskalakis, Tzamos, and Zampetakis [17] defined the problem MetricBanach, which is like MetametricContraction except that it requires a metric, as opposed to a meta-metric. They show that MetricBanach is CLS-complete. Since every metric is a meta-metric, MetricBanach can be trivially reduced in polynomial-time to MetametricContraction. Thus, their CLS-hardness result is stronger than our Theorem 67. The containment of MetricBanach in CLS is implied by the containment of MetametricContraction in CLS. To prove that MetametricContraction is in CLS, we first reduce to GeneralContraction, which we then show is in CLS. Likewise, the proof in [17] that MetricBanach is in CLS works even when violations of the metric properties are not allowed as solutions, so they, like us, actually show that GeneralContraction is in CLS.