On the Complexity of Nash Equilibria and Other Fixed Points

Etessami, K., & Yannakakis, M. (2010). On the Complexity of Nash Equilibria and Other Fixed Points. SIAM Journal on Computing, 39(6), 2531-2597. 10.1137/080720826

On the Complexity of Nash Equilibria and Other Fixed Points

Kousha Etessami
LFCS, School of Informatics
University of Edinburgh

Mihalis Yannakakis
Department of Computer Science
Columbia University

Abstract

We reexamine what it means to compute Nash equilibria and, more generally, what it means to compute a fixed point of a given Brouwer function, and we investigate the complexity of the associated problems. Specifically, we study the complexity of the following problem: given a finite game, Γ, with 3 or more players, and given $\epsilon > 0$, compute an approximation within ϵ of some (actual) Nash equilibrium. We show that approximation of an actual Nash Equilibrium, even to within any non-trivial constant additive factor $\epsilon < 1/2$ in just one desired coordinate, is at least as hard as the long standing square-root sum problem, as well as a more general arithmetic circuit decision problem that characterizes P-time in a unit-cost model of computation with arbitrary precision rational arithmetic; thus placing the approximation problem in P, or even NP, would resolve major open problems in the complexity of numerical computation.

We show similar results for market equilibria: it is hard to estimate with any nontrivial accuracy the equilibrium prices in an exchange economy with a unique equilibrium, where the economy is given by explicit algebraic formulas for the excess demand functions.

We define a class, FIXP, which captures search problems that can be cast as fixed point computation problems for functions represented by algebraic circuits (straight line programs) over basis $\{+, *, -, /, \max, \min\}$ with rational constants. We show that the (exact or approximate) computation of Nash equilibria for 3 or more players is complete for FIXP. The price equilibrium problem for exchange economies with algebraic demand functions is another FIXP-complete problem. We show that the piecewise linear fragment of FIXP equals PPAD.

Many other problems in game theory, economics, and probability theory, can be cast as fixed point problems for such algebraic functions. We discuss several important such problems: computing the value of Shapley's stochastic games, and the simpler games of Condon, extinction probabilities of branching processes, probabilities of stochastic context-free grammars, and termination probabilities of Recursive Markov Chains. We show that for some of them, the approximation, or even exact computation, problem can be placed in PPAD, while for others, they are at least as hard as the square-root sum and arithmetic circuit decision problems.

1 Introduction

A wide variety of problems from many fields (economics, game theory, probability, etc.) can be cast in the form of finding a solution to a fixed point equation $x = F(x)$. Computing a Nash equilibrium is one prominent such problem that has attracted a lot of attention in economics, and more recently in the computer science community. Nash's theorem says that every (finite) game has an equilibrium, i.e., a set of mixed strategies for the players such that no player can improve its payoff by changing its strategy unilaterally [48]. Nash proved his theorem using Brouwer's fixed point theorem: every continuous function F from a compact convex body to itself has a fixed point. There are many other applications of Brouwer's theorem (and related fixed point theorems, e.g., Banach, Kakutani) such as price equilibria, values of games, probabilities of events in stochastic

models and others. The problem is that the proof of Brouwer's theorem is nonconstructive, i.e., it establishes the existence of one or more fixed points without showing how to compute one.

The problem of computing Nash Equilibria, and more generally computing fixed points of Brouwer functions, has a long and rich history, dating back at least to the fundamental algorithm of Scarf [52].[1] Given a continuous function F and $\epsilon > 0$, Scarf partitions the domain into simplices of sufficiently small diameter δ (depending on ϵ and the modulus of continuity of the function F) and navigates through the simplices to produce a point x' such that $||F(x') - x'||_\infty < \epsilon$. The point x' is *almost* fixed by F, but it may be far from the actual fixed points. Let us call such a point x' a *weak* ϵ-fixed point (weak ϵ-FP), to distinguish it from a point x that is *near* a fixed point x^* (i.e., $||x^* - x'||_\infty < \epsilon$) which we will call a *strong* ϵ-fixed point. (The names are due to the fact that for the kind of 'well-behaved' functions that are encountered in most applications, weak approximation reduces to strong; see Section 2, Proposition 2, for a formal result that captures this.) To establish the existence of an actual fixed point, the simplicial partition can be refined more and more, so that the diameter, δ, of the simplices tends to 0; then the sequence of weakly approximate fixed points must have (by compactness) a subsequence that converges to a point, which must be an actual fixed point x^*. However, as Scarf pointed out ([52]), this latter part of the argument (existence of a convergent subsequence) is nonconstructive in general. A number of other algorithms have been proposed both for general fixed points and for Nash equilibria. Note that the goal of the algorithms is to compute or approximate a (any) fixed point or Nash equilibrium, not a specific one; computing a specific one, for example the one with highest payoff, is NP-hard [24].

In [50], Papadimitriou introduced a complexity class, PPAD, to capture problems like (approximate) fixed points and Nash, and showed that a certain discretized version of the Brouwer fixed point problem is complete for PPAD. The class PPAD lies between (the search problem versions of) P and NP. The Nash problem has been investigated intensely recently in the theoretical computer science community, and last year in a breakthrough set of papers [13, 10, 11] it was shown that computing an (exact) Nash equilibrium for 2 players is PPAD-complete, and so is the problem of computing a ϵ-Nash Equilibrium (ϵ-NE) for any number of players. An ϵ-NE is a profile of mixed strategies where no player can improve its own payoff by more than ϵ by switching strategies unilaterally. ϵ-NEs correspond in a precise sense (they are polynomially equivalent, see section 2) to weak ϵ-fixed points of Nash's function.

Equilibria are the central solution concept in game theory and economics, and are meant to characterize (i.e., predict or prescribe) the possible outcome(s) in settings with rational agents who want to maximize their payoffs. We would therefore like to compute these predicted outcomes. As is usual in computer science, all input data (the game payoff tables in this case) are assumed to be rational for computational purposes. One major difference between games with 2 and 3 players is that in the 2-player case there are always rational NEs and thus they can be computed exactly, whereas for 3 and more players this is not the case: in general all NEs can be irrational. The same phenomenon occurs in most applications of Brouwer's theorem: the domain is not discrete (the theorem depends after all on the function being continuous) and the fixed points are in general irrational. This is also a familiar phenomenon in many other applications in science and engineering, where the quantities of interest are described by nonlinear equations that have in general irrational solutions. What does it mean then to compute fixed points and equilibria in these cases? The usual approach for computing irrational numbers, since we cannot print all their digits, is to compute a rational value that approximates them within some desired precision (error) ϵ; for example, π is approximately 3.14159 up to five decimal digits of precision, i.e. within error $\epsilon < 10^{-5}$. Similarly,

[1] Scarf's algorithm in turn builds on ideas from the Lemke-Howson algorithm for computing an exact Nash Equilibrium for 2-player games [42].

for fixed points and equilibria that are irrational, we would like to approximate such a solution within a specified precision, i.e., to compute a rational point that differs from a solution (fixed point or equilibrium) by at most ϵ in every coordinate. Note that this is different from computing a ϵ-NE, which can be very far from any actual Nash equilibria (see Corollary 11 for a precise statement about how far it can be). There are analogous situations in the physical world: a physical system may be only slightly off-balance, i.e., the net effect of forces may be very small, but still the equilibria may be far away.

So how hard is it to compute or approximate within ϵ a Nash equilibrium (any one) for 3 or more players? Is it in PPAD? Is it even in NP? And if not, what is the right class that captures these problems, i.e., the class for which they are complete? These are some of the questions we address in this paper.

First, we show that placing the problem of approximating an actual Nash equilibrium in NP (with any nontrivial approximation error) will imply a breakthrough on longstanding open problems. In the *Square Root Sum* problem (SQRT-SUM for short) we are given positive integers $d_1, \ldots, d_n$ and k, and we want to decide whether $\sum_{i=1}^{n} \sqrt{d_i} \leq k$. This problem arises in many contexts, e.g., in geometric computations where the square root sum represents the sum of Euclidean distances between given pairs of points with integer (or rational) coordinates; for example, determining whether the length of a specific spanning tree, or a given TSP tour of given points on the plane is bounded by a given threshold k amounts to answering such a problem. This problem is solvable in PSPACE, but it has been a major open problem since the 1970's (see, e.g., [22, 49, 59]) whether it is solvable even in NP (or better yet, in P).

A related (and in a sense more powerful and fundamental) problem is the PosSLP problem: given a division-free straight-line program, or equivalently, an arithmetic circuit with operations $+, -, *$ and inputs 0 and 1, and a designated output gate, determine whether the integer N that is the output of the circuit is positive. As shown in [1], the class P^{PosSLP}, i.e., decision problems that can be solved in polynomial time using an oracle (i.e., a subroutine) for PosSLP, is equal to the class of discrete decision problems that can be solved in polynomial time in the Blum-Shub-Smale model of real computation [5] using rational numbers as constants. This is a powerful model, which is equivalent to the unit cost algebraic RAM model in which all operations on rationals take unit time, no matter how large the numbers; in particular the SQRT-SUM problem can be decided in polynomial time in this model [59]. Importantly, the division operator is exact rational division, not integer division; it is known that with integer division (the floor function) all of PSPACE can be decided in the unit-cost model in polynomial time [54, 4]. Allender et. al. [1] showed that PosSLP and SQRT-SUM lie in the Counting Hierarchy, an analog of the polynomial-time hierarchy for counting classes like $\#P$. Thus, it is unlikely that either of these problems is PSPACE-complete, but it remains an important open question whether either problem can be decided in P or even in NP. It is worth pointing out that the problem EquSLP, which asks whether a given arithmetic straight-line program over $\{+, *, -\}$ with integer inputs produces output value exactly equal to 0, a problem which looks a lot easier than PosSLP, is already equivalent to the well-studied problem of polynomial identity testing (PIT), see [1]. PIT (equivalently, EquSLP) can be decided in co-RP, but no NP algorithm is known for it, and in fact an NP upper bound would yield as a consequence difficult lower bounds (see [35]). PosSLP easily subsumes PIT and appears to be much harder than PIT, and it seems reasonable to conjecture that an NP algorithm for PosSLP does not exist.

We show that SQRT-SUM and PosSLP reduce to the problem of approximation of 3-player Nash equilibria. Specifically, for any $\epsilon > 0$, they reduce to this problem: given a game, Γ, with the property (promise) that it has a unique Nash equilibrium, and such that a particular strategy is played in the unique NE with probability either 0 or at least $1 - \epsilon$, decide which of the two is the case for Γ. This means that any non-trivial approximation of the desired coordinate in the unique

NE, say to within any constant $c < 1/2$, would enable us to distinguish the two cases for small enough $\epsilon > 0$.

We show similar results for market equilibria: given an exchange economy with algebraic excess demand functions that has a unique price equilibrium (in the unit price simplex), it is hard to determine whether the price of a commodity in the unique equilibrium will be very close to 0 or very close to 1.

Note that these hardness results hold even when there is a unique equilibrium, and thus there is no issue of which equilibrium to select. The issue of uniqueness and problems of equilibrium selection in games and markets with multiple equilibria have received a lot of attention; if the notion of equilibrium captures the predicted outcome, which one should it be when there are multiple equilibria and how is it selected? See, e.g., [30] for a general theory of equilibrium selection in games (and note the remarks in the foreword by Aumann on the role of uniqueness), and see [46], Chapter 17 for discussion on uniqueness (and local uniqueness) in market equilibria, sufficient conditions, and references. When there is a unique equilibrium, then there is of course no ambiguity, which is a desirable property: there is one predicted outcome. One would like to be able to compute it.

The hardness proof has several interesting consequences. One implication is that ϵ-Nash equilibria for doubly-exponentially small $\epsilon > 0$ (i.e. $\epsilon = 1/2^{2^{n^c}}$ for some c) can still be at distance almost 1 from the (unique) Nash equilibrium (see Corollary 11). We show that this phenomenon does not arise for 2-player games: for every 2-player game G and every rational $\delta > 0$, we can pick a rational $\epsilon > 0$ of bit-size polynomial in that of δ and the size of the game such that every ϵ-Nash equilibrium is within δ of a Nash equilibrium of G. Another difference between 3-player and 2-player games is that in 2-player games the crux of the problem is in determining the support of a NE (which pure strategies have nonzero probabilities); once we know the support, we can easily compute in P-time a NE with that support. However this is not the case for 3-player games: we show that even if we know that the game has a unique NE, and that the NE has full support (all the pure strategies of all the players have nonzero probability), it is still SQRT-SUM- and PosSLP-hard to approximate the NE, specifically, it is hard to determine whether a particular strategy is played with probability less than ϵ or more than $1 - \epsilon$ for any constant $\epsilon > 0$ (i.e. very close to 0 or to 1). We also show that it is $\#P$-hard to compute a desired bit of the (unique) NE, where the index of the desired bit is specified in binary.

We define a new complexity class FIXP of problems that capture fixed point problems for algebraically defined functions, over a compact convex domain. FIXP is the class of search problems that can be expressed as fixed point problems for functions represented by polynomial size algebraic circuits over the basis $\{+, -, *, /, \max, \min\}$ with rational constants.

We show that the Nash equilibrium problem for 3 players (or more) is complete for FIXP. Nash is complete both in the sense of exact and approximate computation. This result shows an intimate connection between Nash equilibria and fixed points of algebraic functions, and explains why the proofs of Nash's theorem (and algorithms) for games with a general number of players (3 or more) use a fixed point theorem (Brouwer or the related Kakutani theorem), and ultimately some type of compactness argument. Recall that, in the case of 2 players, there is a direct, constructive proof of Nash's theorem, namely the Lemke-Howson algorithm [42], which does not rely on any fixed point theorem nor compactness argument.

Another well-known application of fixed point theorems is the existence of market equilibria. We show that computing a price equilibrium in an exchange economy with excess demand functions that are algebraic (over the same basis $\{+, -, *, /, \max, \min\}$) is another complete problem for FIXP.

We show that the class FIXP is rather robust in several senses. Specifically, FIXP does not change with the introduction of other operators to the basis, such as $\sqrt[k]{}$ and fractional powers. Moreover, we shall construct a new family of algebraic $\{+, *, \max\}$ functions (without division) whose fixed points are precisely the Nash Equilibria of a given game, from which it follows that algebraic circuits over the basis $\{+, *, \max\}$, i.e., without division, are already sufficient to characterize all of FIXP. Furthermore, we shall show that the domain assumed for the Brouwer functions is rather flexible: it can be the standard (unit) n-simplex, or the (unit) n-cube, or any convex polytope specified by (rational) linear inequalities, and it can even be an ellipsoid specified by its (rational) center and associated matrix. We show that all of these domains yield the same class FIXP (both with respect to approximation as well as decision problems), via a suitably weak notion of polynomial-time (real-valued) search problem reduction. Finally, note that Nash's functions are given by algebraic formulas, rather than circuits. It therefore follows from the FIXP-completeness of Nash equilibria that FIXP is already captured by fixed point functions given by algebraic formulas over basis $\{+, -, *, /, \max, \min\}$. Thus, overall the class FIXP is rather robust is several ways: with respect to the set of algebraic operators, whether we allow functions specified by algebraic circuits or just formulas, and also with regard to what (compact convex) domain we choose for the functions. See Sections 2.3 and 4 for details.

An upper bound on the complexity of the discrete computational tasks associated with search problems in FIXP (e.g., the approximation and decision problems) is PSPACE. We know no better bound in general.

There are a number of other well-studied problems from different areas that can be cast as fixed point problems of suitably defined functions given by simple algebraic formulas, and thus are also in FIXP. We study several important such problems that can be cast in a fixed point framework. Despite extensive work and a very rich theory developed over the years for these models, the complexity of fundamental problems for them remain open. We now discuss several of these models. *Stochastic games* were first introduced by Shapley [56] in 1953, and have been extended in various directions and studied extensively since then. A simpler version, called simple stochastic games was introduced by Condon [12] in computer science and has attracted a lot of attention. The quantities of interest in these games are to compute or bound the values of the games (they are unique) and to find optimal strategies for the players. *Branching processes* (BP) were first introduced, in the 1-type case, by Galton and Watson in the 19th century to model population dynamics, and generalized to the multi-type case by Kolmogorov and Sevastyanov ([39]) motivated by biology. They are a basic probabilistic model for many applications (e.g., biological processes [28, 36] and many others). The most basic quantities of interest here are the extinction probabilities of entity types. *Stochastic context-free grammars* (SCFG) are a model in common use in Natural Language Processing [45] and biological sequence analysis [16]. The probabilities of the language and sublanguages generated by the grammar are quantities of interest here. *Recursive Markov chains* (RMC), a more powerful model that encompasses in a precise sense both BPs and SCFGs, were introduced in [19] to model recursive probabilistic programs (see also [18] for an equivalent model). Basic quantities of interest here are the termination probabilities.

In all of the above models, one can define an appropriate function F such that the desired quantities x^* are a fixed point of F; in fact in all cases except for general RMCs, the function and the domain can be defined so that x^* is the *unique* fixed point (for RMCs it is the least nonnegative fixed point). In all these problems, the function F is just a tool to get a handle on the problem. For problems like Nash and market equilibria, weak ϵ-fixed points have a game-theoretic/economic meaning (they correspond to, e.g., ϵ-NEs, i.e., strategy profiles where the players have only a small incentive to deviate) and thus are also of interest, besides of course the approximation of the actual equilibria, which we would like to compute. For the other models mentioned above

however, weak ϵ-fixed points have no significance, unless they help us find, approximate, or answer questions about, the quantities of interest which are the actual fixed points. For example, consider the approximation of the value of a simple stochastic game. We will observe that one can compute easily in polynomial time a weak ϵ-fixed point of the associated function F, for any $\epsilon > 0$ which is constant or even inverse-polynomial ($\epsilon = 1/n^c$ for a constant c); however, this is not useful since it does not tell us how to compute even a constant approximation to the value of the game which is what we are interested in. The same phenomenon occurs in branching processes, RMCs etc. It is easy to obtain a weak ϵ-approximation for $\epsilon = 1/n^c$, but we do not know how to approximate the extinction and termination probabilities.

The distinction between strong (near) and weak (almost) approximate fixed points was noted early on, sometimes with statements which on the surface seem contradictory. For example, Scarf in his original paper [52], remarks that obtaining strong ϵ-fixed points from weak ϵ-fixed points is non-constructive for general mappings. On the other hand, Anderson [2] gives several "almost implies near" theorems, notes a theorem of [41] that for every Brouwer function F and $\epsilon > 0$ there is a $\delta > 0$ such that every weak δ-fixed point is a strong ϵ-fixed point, and shows furthermore that for a class of smooth functions, δ is in fact linear in ϵ. What is going on? Scarf's remark concerns algorithms that use F as a black box, i.e. that work for all possible instances of all problems and have restricted access to the function. Indeed, impossibility results are known in the black-box oracle model, showing that no finite number of queries suffice to compute a strong approximation for a Lipschitz continuous function given as a black box [57], whereas a weak approximation can be obtained with a finite (exponential) number of queries in this model (and an exponential number is required [31]). Anderson's results concern a single function analysed with respect to the precision only. These results represent two extremes: in one case, the model concerns algorithms that work for all possible functions and place a very severe black box limitation on the access to the function. In the other case, the theorem considers one function in isolation, which in effect corresponds to considering a single instance of a problem (for example, Nash for a specific game Γ), i.e., it does not take into account the dependence of the complexity on the instance.

In a concrete problem (like Nash, stochastic games, etc.) we have an intermediate situation between the two extremes, and it is important to set up the framework properly to study the complexity of the problem. We have concrete functions (for example Nash's function for a game), so the black box results are not relevant; for example, we certainly can compute approximations to actual Nash equilibria in finite time, no worse than exponential. On the other hand, we do not have a single function, but rather a class of functions, one for every instance of the problem, and we want to determine the complexity as a function of the instance size, as well as precision. Indeed, our results show that from a quantitative computational perspective, "almost" certainly does not imply "near" for Nash and other classes of Brouwer functions. If we want to study the complexity of these problems, it is in fact essential to distinguish between these notions: blurring the distinction between the computation of ϵ-Nash equilibria and approximate computation of actual Nash equilibria (and more generally, between weak and strong approximate fixed points) can lead to unwarranted and potentially false conclusions, e.g., that the square-root-sum problem is in NP, and that P-time in the unit-cost exact rational arithmetic model is contained in NP.

For some types of functions, weak and strong approximation for sufficiently small ϵ and even exact computation can be related. We define a general class of *polynomial piecewise linear functions*, and show that for them exact fixed point computation is in PPAD; the piecewise linear class includes simple stochastic games, the discretized Brouwer problem of [50] which is PPAD-complete, and the subclass of FIXP, denoted *Linear-FIXP*, where the circuits are restricted to the basis $\{+, \max\}$ and with multiplication by rational constants only, i.e., do not use multiplication or division, except by a constant. Indeed, we show (exact) fixed points of polynomial piecewise linear functions,

Linear-FIXP, and PPAD are all polynomially equivalent.

For Shapley's stochastic games, which have a nonlinear F, the problem is in FIXP, thus Shapley reduces to Nash, both in the exact sense, as well as in the approximate and decision version. Furthermore, we show that the (strong) approximation problem for Shapley's games is reducible to weak for sufficiently small ϵ (of the form $\epsilon = 1/2^{poly}$) and is thus in PPAD. However, bounding the value of the game, e.g., deciding whether Player 1 can achieve reward $\geq r$ is harder: we show that it is at least as hard as SQRT-SUM, and hence placing it in PPAD (or NP) would solve a longstanding open problem.

For branching processes, SCFGs, and a corresponding subclass of Recursive Markov Chains, called 1-RMCs, we show that the problem of computing the extinction (termination) probabilities is in FIXP. The challenging part here is to constrain the domain of the function in a polynomial-time computable way so that we get a Brouwer function with the desired probabilities forming the *unique* fixed point in the domain. The decision problem (comparing the probabilities with a given rational r) is SQRT-SUM- and PosSLP-hard [19]; we do not know the status of the approximation problem for the relevant probabilities.

The rest of this paper is organized as follows. In Section 2 we set up the framework, and give basic definitions and properties. We define total search problems where the solutions may be real-valued (generally irrational) vectors, and define several types of discrete computational problems associated with such search problems in Section 2.1. We present notation and background on games and Nash equilibria in Section 2.2. We then discuss in Section 2.3 the formulation of search problems as fixed point problems, define some important properties of function classes and give some basic results on strong and weak approximation of fixed points for such functions. In Section 3 we give reductions from the Square Root Sum and PosSLP problems to the Nash equilibrium problem, and present the corollaries of these reductions that we described earlier for Nash and market equilibria. In Section 4 we define the class FIXP, and prove that the Nash equilibrium problem is complete for the class. We use the FIXP-completeness of Nash equilibria, together with an alternative fixed point characterization of Nash equilibria, to show that the class of fixed point problems with the reduced basis $\{+, *, \max\}$ and rational constants, is already equally powerful. We also show FIXP-completeness of the price equilibrium problem for algebraically defined excess demand functions. In Section 5 we define and discuss the class of piecewise-linear fixed point problems. We present several examples of such problems (simple stochastic games, linearly interpolated functions, Nash for 2-player and polymatrix games), and show that this class of problems is equivalent to Linear-FIXP and to PPAD. In Section 6 we study Shapley's stochastic games. We show that Shapley is in FIXP, the approximation problem is in PPAD, and the decision problem is SQRT-SUM-hard. Section 7 concerns branching processes, SCFGs, and 1-RMCs. We give the relevant definitions and show that the computation of their termination probabilities is in FIXP. Section 8 gives some concluding remarks.

2 Preliminaries

We will describe first a general framework for search problems where the solution sets may be real-valued, and thus not computable exactly. We will then give definitions and brief background for games and Nash equilibria. Finally we will set up the framework for expressing search problems as fixed point problems for a class of functions, define some interesting classes of functions and types of approximation, and give some of their basic properties.

2.1 Total Search Problems

A *search problem* Π has a set of instances, represented by finite strings over a fixed finite alphabet Σ, and each instance I has an associated set $Sol(I)$ of "acceptable" *solutions*. Numbers in the input (such as weights, rewards, probabilities etc.) are assumed to be rational as usual for computational purposes, represented by the numerator and denominator written in binary; the *size* of a rational number is the number of bits in the numerator and denominator. The size $|I|$ of the instance I is the length of the string that represents it. As usual, it is assumed that, given a string over Σ, one can determine in polynomial time if the string is an instance of the problem.

The problems we will be interested in here (equilibria, fixed points, probabilities, etc.) are *total*: every instance I has a nonempty set $Sol(I)$ of solutions. For some problems there may be a unique solution (for example, probabilities of certain events in a stochastic model), while for others there may be multiple solutions (for example, Nash equilibria of a game). Unlike usual discrete problems where solutions are also finite (and represented as strings), the search problems that we study here have solutions that are in general real-valued vectors of finite dimension, d_I, that is polynomially bounded in the size $|I|$ of the instance. We would like to solve the following problems:

1. Exact Computation: Given input I, compute a solution x in $Sol(I)$. Note that if there are multiple solutions, then any one of them is a correct output. If there are rational solutions, we can output them explicitly, i.e., the problem is discrete and can be studied in the standard discrete Turing machine model. In several problems, the solutions may be inherently irrational, so we cannot output them explicitly. In this case the exact computation problem could be studied in a real computation model, such as the model of [5]. However, our focus is on the standard Turing machine model. In this model we can only compute some desired finite information about a solution, such as compute bounds on it, or approximate it up to a desired precision. Attention has to be paid in the formulation of the discrete problems to stay faithful to the search problem and not make it harder: in particular, if there are multiple solutions, any one of them should enable us to answer the question. There are several types of information that one may want to compute about a solution, leading to various discrete problems, and this can make a difference in the complexity.

2. Partial Computation: Compute a specified number of bits of the solution. Given instance I and integer $k > 0$ in unary, compute the binary representation of some solution (any one), up to the first k bits after the decimal point. We would like to do this in time polynomial in $|I|$ and k.

3. Decision Problem: Given instance I, rational vector r and a comparison operator vector θ (e.g., $\geq, \leq$, etc.) return a truth value that holds for at least one of the solutions, i.e. if all solutions x satisfy $x\theta r$ then return 'Yes', if none satisfies $x\theta r$ then return 'No', and if some do and some do not, then either answer is correct. Alternatively, and more usually, the decision question may be posed on a particular entry: "return 'Yes' if $x_1 \geq r$ for all solutions, 'No' if $x_1 < r$ for all solutions, and otherwise (if solutions exist with both answers) then either answer is fine". This formulation of the decision problems reflects the standard way of turning optimization problems and other problems with output to decision problems, except since there can be many different solutions on which a given true/false predicate may have different truth values, we want to treat an answer as correct as long as it is based on knowing *any* solution and evaluating the predicate on it. Thus, we consider both 'yes' and 'no' as correct answers in cases where there exist distinct solutions for which the predicate of interest is both true and false. We would like running time polynomial in $|I|$ and in the size (number of bits) of r. Note that the decision problem, as formulated above, requires simply to return a truth value of the predicate that holds for some solution, rather than answering whether there exists a solution that satisfies the predicate. The **Existence question**, given instance I and rational r, is there a solution x with $x_1 \geq r$?, is equivalent to the decision question for problems with

unique solutions, but not otherwise. For many search problems the existence question is NP-hard while the search problem is not. For example, for 2-player Nash equilibria, the existence question is NP-hard [24], while the search problem is not unless NP=coNP. Moreover, one can define trivial search problems for which the existence question is NP-hard: consider graph coloring, where the solutions are all legal colorings, x_1 is the number of colors used, and the question is $x_1 \leq 3$?. The search problem is trivial (compute any legal coloring of the graph) but determining whether there exists a solution with $x_1 \leq 3$ is NP-hard. In the case of search problems with multiple solutions, one has to be careful in formulating the decision version in a way that does not make it harder than the search problem itself: knowing any solution should enable one to answer easily the decision question (by evaluating the predicate on that solution).

4. Approximation Problem: Given instance I and rational $\epsilon > 0$, compute a vector x that is within (additive) $\epsilon > 0$ of some solution, i.e., such that there is a $x^* \in Sol(I)$ such that $|x^* - x|_\infty \leq \epsilon$. Alternatively we could require an approximation of only a particular entry of a solution vector, e.g., approximate x_1^* within additive $\epsilon > 0$. We would like polynomial time in $|I|$ and in $\log(1/\epsilon)$; this permits approximation within 2^{-k} in time polynomial in I and k. For several problems we will show that the approximation problem is hard for some class, by showing hardness of a corresponding **Promised Gap Decision Problem** PGD(a,b): Given instance I, rationals $a < b$, and the promise that either all solutions $x \in Sol(I)$ have $x_1 \leq a$ or they all have $x_1 > b$, determine which of the two is the case.

There are some simple relations between these problems. Clearly, the partial computation of a solution up to k bits after the decimal point is a 2^{-k}-approximation. The other direction does not necessarily hold: in principle, no matter how small the error ϵ in the approximation, we cannot be sure what the first bit is, for example we cannot tell whether a probability is 1 or not.

For search problems whose solutions are rational, of size polynomial in the input size, if we can solve the approximation problem in polynomial time, then we can also solve the exact computation problem: Suppose that the polynomial $q(n)$ is a bound on the size (number of bits in numerator and denominator) of the solutions for instances of size n. Given an instance I, compute an approximation v within $\epsilon = 1/2^{3q(|I|)}$ of a solution. Then compute for each component of v the closest rational number whose denominator is bounded by $2^{q(|I|)}$; this can be done in polynomial time by the continued fraction method (see, e.g., [26]). The obtained vector y is uniquely determined, because each component of v is $1/2^{3q(|I|)}$-close to a rational with denominator at most $2^{q(|I|)}$ and any two such rationals are at least $1/2^{2q(|I|)}$ apart. The vector y must therefore be the solution that is ϵ-close to v.

For problems with a unique solution, as already noted the Existence and the Decision problem are equivalent; using them and binary search we can solve the partial computation (and approximation) problem in time polynomial in the input and output. The converse does not hold in principle, because a partially computed solution is in base 2, so we cannot tell with it how the actual solution compares for example with 1/3, no matter how many bits we have after the decimal point.

In general the above three discrete problems are related but not equivalent, i.e. for a search problem Π, the associated Partial Computation, Decision, and Approximation problems may well have different complexity.

Reductions between (real valued) Search Problems.

In the usual case of discrete search problems, a (polynomial) reduction from problem A to problem B consists of two polynomial-time computable functions: a function f that maps instances I of A to instances $f(I)$ of B, and a second function g that maps solutions y of the instance $f(I)$ of B to solutions x of the instance I of A. For search problems with real-valued solutions, we have to specify what kind of functions g are allowed, since our model of computation is the standard

(discrete) Turing machine model, and thus we cannot talk about polynomial-time complexity for functions on real numbers (as we could if we instead used a real model of computation). The main criterion is that the definition should enable us to easily transfer back (discrete) solutions for B to (discrete) solutions for A, for the discrete problems of interest associated with the search problems, e.g., the decision, and approximation problems. It is sufficient for our purposes in this paper to restrict the reverse function g to have a particularly simple form; of course if a problem A reduces to B under a simple reduction, it also reduces under more powerful ones. Specifically, we will restrict the function g to be a *separable linear* transformation with polynomial-time computable rational coefficients; i.e., $x = g(y)$, where each coordinate x_i of x is obtained by a linear transformation $g_i(y) = a_i y_j + b_i$ on some coordinate y_j of y, where the coefficients a_i, b_i are rationals computable from I in polynomial time. We will call such reductions, given by functions f, g, where f is P-time computable and g is a separable linear transformation with P-time computable coefficients, **SL-***reductions*. A reduction of this form from A to B induces corresponding P-time reductions for the Decision and the Approximation problems. For the Decision problem, i.e., to answer for example the question '$x_i \geq r$?' for a solution to instance I of A, we construct the instance $f(I)$ of B and ask the question '$y_j \geq (r - b_i)/a_i$?' if $a_i > 0$, or '$y_j \leq (r - b_i)/a_i$?' if $a_i < 0$ (if $a_i = 0$ the answer is obvious). Similarly, it is easy to see that the Approximation problem for A reduces to that for B. Furthermore, in all our reductions we can take the coefficients a_i, b_i to be (positive or negative) powers of 2, in which case the reduction induces also a reduction from the Partial computation problem for A to that of B.

2.2 Games and Nash Equilibria

In a (normal form) game Γ with k players, the instance consists of k (disjoint) finite sets of *pure strategies* $S_i, i = 1, \ldots, k$, and k rational-valued *payoff functions* u_i from the product strategy space $S = \Pi_i S_i$ to $\mathbb{Q}$. The elements of S, i.e., combinations of pure strategies, one for each player, are called *pure strategy profiles*. The assumption of rational values is for computational purposes. Each rational number r is represented as usual by its numerator and denominator in binary, and we use $size(r)$ to denote the number of bits in the representation. The size $|\Gamma|$ of the instance (game) Γ is the total number of bits needed to represent all the information in the game: the strategies of all the players and their payoffs for all $s \in S$.

A *mixed strategy* for a player i is a probability distribution on its set S_i of pure strategies. A *mixed strategy profile* x is a combination of mixed strategies for all the players. Letting $n_i = |S_i|$ and $n = \sum_i n_i$, a mixed strategy profile x can be represented by a non-negative vector of length n (i.e., its entries are indexed by all the players' pure strategies) that is a probability distribution on the set of pure strategies of each player. We will use x_i to denote the subvector of x corresponding to player i (i.e. the mixed strategy of player i in the profile x), and use $x_{i,j}$ to denote the probability with which player i plays his strategy j in x. Thus, a vector x_i is a mixed strategy of player i iff it belongs to the *unit simplex* $\Delta_{n_i} = \{y \in R^{n_i} | y \geq 0; \sum_{j=1}^{n_i} y_j = 1\}$ and the vector x is a mixed strategy profile iff it belongs to the product of the k unit simplexes for the k players, $\{x \in R^n \mid x \geq 0; \sum_{j=1}^{n_i} x_{i,j} = 1$ for $i = 1, \ldots, k\}$. We let D_Γ denote the set of all mixed profiles for game Γ.

The *support* of a mixed strategy profile x is the set of pure strategies that have nonzero probability in x. The profile is *fully mixed* if all the pure strategies of all players have nonzero probability. We use the notation $(i{:}j)$ to identify the pure strategy j of player i, as well as its representation as a mixed strategy that assigns probability 1 to strategy j and probability 0 to the other strategies of player i. When it is convenient, we will sometimes identify a pure strategy profile s with the corresponding mixed strategy profile that assigns probability 1 to the pure strategies of s and probability 0 to the other strategies. The payoff function of each player can be extended from pure

strategy profiles to mixed profiles, and we will overload notation and use u_i to also denote the expected payoff function for player i. Thus the (expected) payoff $u_i(x)$ of mixed profile x for player i is $\sum x_{1,j_1} \dots x_{k,j_k} u_i(j_1, \dots, j_k)$ where the sum is over all pure strategy profiles $(j_1, \dots, j_k) \in S$. We use the notation x_{-i} to denote the subvector of x induced by the pure strategies of all players except for player i. If y_i is a mixed strategy of player i, we use $(y_i; x_{-i})$ to denote the mixed profile where everyone plays the same strategy as x except for player i, who plays mixed strategy y_i.

A *Nash equilibrium* (NE) is a strategy profile x^* such that no player can increase its payoff by switching its strategy unilaterally. Formally, x^* is a NE if for all $i = 1, \dots, k$ and every mixed strategy y_i for player i, $u_i(x^*) \geq u_i(y_i; x^*_{-i})$. It is sufficient to check switches to pure strategies only, i.e., x^* is a NE iff $u_i(x^*) \geq u_i((i{:}j); x^*_{-i})$ for every pure strategy $j \in S_i$, for each player $i = 1, \dots, k$.

Nash showed that every finite game has at least one NE [48]. Thus, the problem of computing a Nash equilibrium for a given game I is a total search problem, where $Sol(I)$ is the set of NE of game I.

Two-player games (with rational payoff functions) have rational NEs, thus exact computation is possible, and as shown in [10, 9] the problem is PPAD-complete. PPAD is a class of discrete total search problems, introduced in [50], that captures the basic principles of path-following algorithms like Lemke-Howson [42] and Scarf [52]. Formally, a search problem Π is in PPAD if each instance I has a set $S(I)$ of candidate solutions which are (strings) polynomially bounded in the input size $|I|$, and there are polynomial-time algorithms for the following tasks: (a) test whether a given string I is an instance of Π and if so compute an initial candidate solution s_0 in $S(I)$, (b) given I, s, test whether $s \in S(I)$ and if so compute a successor $succ_I(s) \in S(I) \cup \{\perp\}$ and a predecessor $pred_I(s) \in S(I) \cup \{\perp\}$, where the symbol $\perp$ stands for 'nil' (i.e. there is no successor or predecessor), such that $pred_I(s_0) = \perp$, $succ_I(s_0) \in S(I)$, and for all $u, v \in S(I)$, $succ_I(u) = v$ iff $u = pred_I(v)$. The *pred* and *succ* functions induce a directed graph $G(I) = (S(I), E)$, where $E = \{(u, v) | succ_I(u) = v, pred_I(v) = u\}$. The graph $G(I)$ consists of a set of directed paths, cycles, and isolated nodes, and s_0 is the source node of one of the paths. The desired solution set to the instance I of the search problem, $Sol(I)$, is the set of nodes of $G(I)$, other than s_0, that have indegree + outdegree = 1, i.e., are endpoints of the paths; note that $Sol(I) \neq \emptyset$ because there must be at least one more endpoint besides s_0. As usual, the class is closed under P-time reductions, i.e., if search problem A reduces to problem B, and B satisfies the above definition, then A is considered also to belong to PPAD. The class PPAD lies somewhere between P and TFNP: all search problems in PPAD are total, and furthermore, for a given instance I, we can guess a solution $s \neq s_0$ and verify that it is a source or sink in the graph $G(I)$, i.e. that $s \in Sol(I)$.

For three or more players, it is possible that all the Nash equilibria are irrational, and thus cannot be computed exactly. In fact, as shown by Bubelis in [7], any real root of any univariate polynomial can arise as the (scaled) probability of a particular strategy or the payoff of a particular player in a Nash equilibrium of a 3-player game, whose strategies and payoff functions depend on the degree and the coefficients of the polynomial. In another result, Datta showed that every real algebraic variety is isomorphic to the set of fully mixed Nash equilibria of a 3-player game [15]: For every real algebraic variety V given by a set of polynomial equations in n variables, a 3-player game can be constructed whose set of fully mixed Nash equilibria is isomorphic to V. This theorem does not give a reduction because in general the game will have more Nash equilibria besides the fully mixed ones; in fact, a set of polynomial equations may have no solutions, whereas every game has one or more NEs. Also, the number of pure strategies of the game in Datta's construction is exponential in n. Nevertheless, the result indicates the richness of 3-player games.

Since the NE are irrational, we would like to address the associated discrete computational problems, i.e. compute or approximate a NE up to specified precision, or solve the related decision problem; this is one of the main subjects of this paper. Note that approximating a (actual) NE

is different from the notion of an ϵ-NE (and the related notion of well-supported ϵ-NE) studied previously and shown PPAD-complete [13, 25, 11]. An ϵ-NE is a strategy profile x such that no player can improve its payoff by more than ϵ by switching unilaterally to another strategy. That is, for every player i and every mixed strategy y_i of player i, $u_i(x) \geq u_i(y_i; x_{-i}) - \epsilon$ (again, it is sufficient to check switches to pure strategies only). Thus, a profile x is an ϵ-NE if it is *almost* at equilibrium, rather than being *near* an equilibrium. A well-supported ϵ-NE is an ϵ-NE x that has in addition the property that every pure strategy $(i{:}j)$ in the support of x has (expected) payoff $u_i((i{:}j); x_{-i}) \geq u_i(x) - \epsilon$. As shown in [13] the notions of ϵ-NE and well-supported ϵ-NE are polynomially equivalent.

In general, an ϵ-NE (whether well-supported or not) can be very far from all actual NEs. Specifically, our results will imply (see Corollary 11) examples of games, $\Gamma(n)$, of size $\Theta(n)$, such that there are $1/(2^{2^{\Omega(n^c)}})$-NEs which have distance 1 (in l_∞) to any actual NE. So, even for an ϵ so small that it requires exponentially many bits in binary relative to the size of the game just to write down ϵ, an ϵ-NE can have the worst-case possible distance in some coordinate from any actual NE, namely distance 1. On the other hand, all profiles that are sufficiently close to a NE are also ϵ-NEs: more precisely, for every game I and $\epsilon > 0$, we can take a $\delta > 0$ of encoding size polynomial in the size of I and ϵ, such that every profile x' such that $|x' - x^*|_\infty < \delta$ for some NE x^*, is an ϵ-NE ([43]).

2.3 Fixed Point Problems.

Nash proved his theorem in [48] using Brouwer's theorem, which asserts that every continuous function F from a convex compact domain D to itself has at least one fixed point, i.e. there is a point $x^* \in D$ such that $x^* = F(x^*)$. Clearly, for every finite game Γ, the set D_Γ of mixed strategy profiles is convex and compact (it is a convex polytope). Nash defined for every finite game Γ a continuous function F_Γ from D_Γ to itself and showed that the fixed points of F_Γ are precisely the Nash equilibria of Γ. Specifically, for each player i and strategy $j \in S_i$, the (i,j) component of the function F_Γ is defined as follows: $F_\Gamma(\mathbf{x})_{(i,j)} \doteq \frac{\mathbf{x}_{i,j} + \max\{0, g_{i,j}(\mathbf{x})\}}{1 + \sum_{l=1}^{m_i} \max\{0, g_{i,l}(\mathbf{x})\}}$, where $g_{i,j}(\mathbf{x}) = u_i((i{:}j); x_{-i}) - u_i(x)$ is the net "gain" of player i if he switches to pure strategy j (the "gain" $g_{i,j}(\mathbf{x})$ is a polynomial in $\mathbf{x}$, which may be positive, 0, or negative for a given profile).

Another well-known application of fixed point theorems is the existence of market equilibria. Consider the following *exchange equilibrium* problem [53]. We have m agents and n commodities. Each agent has an endowment (supply) of commodities which he brings to the market, sells it at the prevailing prices and buys his preferred bundle of commodities. Let e^l be the endowment vector of commodities for agent l, thus, e^l_i is the amount of commodity i that agent l brings to the market. If p is the vector of prices for the commodities, agent l receives $\langle p, e^l \rangle = \sum_i p_i e^l_i$ amount of money, and with this he buys the vector of commodities $d^l(p)$ (his 'demand' vector), which is the bundle of commodities that optimizes his preferences (his utility) at price vector p, using the money he obtained by selling his/her endowment at price vector p. The difference between the agent's demand and supply, $g^l(p) = d^l(p) - e^l$, is called the *excess demand* of agent l. Since the money he receives $\langle p, e^l \rangle$ is equal to the money he spends, $\langle p, d^l(p) \rangle$, the excess demand function satisfies the constraint $\langle p, g^l(p) \rangle = \sum_{i=1}^n p_i g^l_i(p) = 0$, called Walras' law. Two other standard assumptions on the excess demand functions are that (i) they are homogeneous of degree 0 (i.e., for all $\alpha > 0$, $g^l_i(\alpha p) = g^l_i(p)$) and thus price vectors may be normalized to lie on the unit simplex Δ_n, and (ii) they are continuous on the unit simplex. Let $g_i(p) = \sum_l g^l_i(p)$ be the (total) market excess demand for each commodity i. The functions $g_i(p)$ clearly also satisfy Walras' law, $\sum_{i=1}^n p_i g_i(p) = 0$, and the same constraints (i) and (ii). A vector p of prices is an *equilibrium* if $g_i(p) \leq 0$ for all i (demand

does not exceed supply), with equality for all commodities i that have $p_i > 0$.

There is always at least one equilibrium, and the proof is via Brouwer's theorem. Namely, the equilibria are the fixed points of the function $F : \Delta_n \mapsto \Delta_n$, defined by the formula $F_i(p) = \frac{p_i + \max(0, g_i(p))}{1 + \sum_k \max(0, g_k(p))}$. Conversely, Brouwer's theorem can be derived from the price equilibrium theorem [60]. Namely, given a Brouwer function $f : \Delta_n \to \Delta_n$, one can define a total market excess demand function $g : \Delta_n \to R^n$ where $g(p) = f(p) - (\langle p, f(p)\rangle / \langle p, p\rangle)p$. It is easy to see that g satisfies the constraints of an excess demand function (e.g., $\langle p, g(p)\rangle = 0$ for all p , Walras' law) and hence has an equilibrium. Furthermore, any price equilibrium is a fixed point of f.

In this paper we study search problems that can be cast in a fixed point framework: every instance I of the search problem Π is associated with a continuous function F_I mapping a compact convex domain D_I to itself, such that the set $Sol(I)$ of solutions is $Fix(F_I)$, the set of fixed points of F_I. More generally, we will also allow polynomial time SL-reduction from the search problem Π for I to the fixed point problem for F_I, i.e., it is not necessary to require $Sol(I) = Fix(F_I)$. Thus, the solutions $Sol(I)$ of the search problem may be obtained from the fixed points of F_I by projecting them to some of the components, and then possibly scaling and translating the components, i.e., applying separate polynomial-time linear transformations with rational coefficients on the components.

We now define some important types of fixed points problems. Consider a fixed point search problem Π and the class $\mathcal{F}$ of functions F_I indexed by the instances I of Π. We say that the class $\mathcal{F}$ (and its fixed point problem Π) is **polynomially continuous** if there is a polynomial $q(z)$ such that for all instances I and all rational $\epsilon > 0$, there is a rational $\delta > 0$ such that $size(\delta) \leq q(|I| + size(\epsilon))$ and such that for all $x, y \in D_I$, $|x - y|_\infty < \delta \;\Rightarrow\; |F_I(x) - F_I(y)|_\infty < \epsilon$. It is easy to see that this definition is robust with respect to the norm: replacing the L_∞ norm in this definition by any other L_k norm, $k \in \mathbb{N} \cup \{\infty\}$, would not alter the classes of functions that are polynomially continuous. Also, note that all Lipschitz continuous functions, i.e. functions that satisfy $|F_I(x) - F_I(y)|_k \leq C_I |x - y|_k$ for all $x, y \in D_I$, with Lipschitz constant $C_I \leq 2^{poly(|I|)}$, are clearly polynomially continuous.

We say that the class $\mathcal{F}$ (and the associated problem Π) is **polynomially contracting** with respect to norm L_k, $k \in \mathbb{N} \cup \{\infty\}$, if there is some polynomial $q(z)$ such that for all instances I there is some rational $\beta < 1 - 2^{-q(|I|)}$, such that for all $x, y \in D_I$, $|F_I(x) - F_I(y)|_k < \beta |x - y|_k$. The contraction property is sensitive to the norm: a class may be contracting with respect to one norm L_k but not with respect to another norm L_t. Clearly, if a class is polynomially contracting with respect to any norm L_k then it is polynomially continuous. A contracting function F_I (whether polynomially contracting or not) has a unique fixed point x^* by Banach's theorem.

We say that $\mathcal{F}$ (and Π) is **polynomially computable** if there is a polynomial $q(z)$ such that (a) the domain D_I for every instance I is a convex polytope (bounded convex polyhedron) described by a set of linear inequalities with rational coefficients that can be computed from I in time $q(|I|)$, and (b) given a rational vector $x \in D_I$, the image $F_I(x)$ is rational and can be computed from I and x in time $q(|I| + size(x))$.

For example, Nash's class of functions $\mathcal{F} = \{F_\Gamma \mid \Gamma \text{ a game}\}$ is polynomially continuous and polynomially computable, but it is not contracting.

In the problems that we will discuss, the domain will usually be a box, or a simplex, or a cartesian product of simplexes. A *box* (hyper-rectangle) $B[a, b]$, where $a < b$ are two rational vectors, is the set of points $\{x | a \leq x \leq b\}$. The unit cube is the box $B[0, 1]$ where 0, 1 are the all-0 and all-1 vectors. We will use $[0, 1]^n$ to denote the unit cube in $\mathbb{R}^n$. A *simplex* is the convex hull of affinely independent points. The *unit simplex* Δ_n is the convex hull of the n unit vectors in $\mathbb{R}^n$, i.e. $\Delta_n = \{x \mid x \geq 0,\ \sum_i x_i = 1\}$. In much of the literature on fixed points it is often customary,

for simplicity, to take the domain of the function to be a box or a simplex, usually the unit cube or the unit simplex. If we have a function F on a convex compact domain D, we can embed D into a suitable large box or simplex D' that contains D, define a continuous function π from D' to D that is the identity on D, and compose π with F to form a continuous function F' on D' whose fixed points are the fixed points of F. For example, the function π from D' to D could be taken to be the projection function that maps every point $x \in D'$ to its closest point in D under the L_2 metric; there is a unique such point since distance is a strictly convex function and D is a bounded closed convex set (computing the closest point is a convex programming problem). The embedding function π does not have to be a projection however; it could be taken to be any other continuous function that is the identity on D. The box or simplex D' can be translated and scaled to a unit cube or unit simplex D''. Of course, if we are interested in properties such as polynomial computability of the function, then we have to make sure that the properties get preserved in the transformation.

Lemma 1 *Every polynomially computable fixed point search problem* Π *is SL-reducible to another polynomially computable fixed point problem* Φ *that has the same set of instances and such that the domain for each instance is a unit cube. If the problem* Π *is polynomially continuous then* Φ *is also polynomially continuous. A similar SL-reduction exists to a problem* Ψ *with a unit simplex as the domain for every instance.*

Proof. Let Π be a polynomially computable fixed point search problem. Every instance I has an associated bounded polyhedral domain D_I and a (continuous) function F_I mapping D_I to itself. Let $d = d_I$ be the dimension of the space (the number of variables of F_I). We will define another polynomially computable fixed point problem Φ with the same instances that has a different associated function G_I with domain $[0,1]^d$.

Compute in polynomial time (using linear programming) two rational vectors $l < u$ that bound respectively from below and above the points in D_I, i.e. $D_I \subset B[l,u]$. We will define a continuous mapping π from $B[l,u]$ to D_I and compose it with F_I to get a function F'_I from $B[l,u]$ to D_I whose fixed points are the same as the fixed points of F_I. Then we will scale and translate the box $B[l,u]$ to map it to the unit cube $[0,1]^d$.

We define the mapping π from $B[l,u]$ to D_I as follows. Compute from I a set C of linear inequalities defining the domain D_I. Compute the affine hull of the domain; this can be done with linear programming (see, e.g., [55] section 8.2). Thus, we can partition the set C into a set C_1 of linear equations and a set C_2 of linear inequalities that are satisfied strictly by some feasible points in D_I. We can assume without loss of generality that the constraints are non-redundant, thus in particular the equations in C_1 are linearly independent, and their number k is equal to the codimension of the affine hull. Let $C_1 = \{a_i x = b_i | i = 1, \ldots, k\}$ and $C_2 = \{a_i x \leq b_i | i = k+1, \ldots, m\}$. Compute a point c in the relative interior of the polyhedron that lies far from the hyperplanes in C_2. In particular, solve the following LP:
Maximize r
Subject to:
$a_i x = b_i$; for $i = 1, \ldots, k$
$a_i x + r \leq b_i$ for $i = k+1, \ldots, m$

Let (c, r^*) be the computed optimal solution. Note that the components of c and the value r^* are rationals of polynomial bit complexity in the size $|I|$ of the instance.

If $k > 0$ (the polytope D_I is not full-dimensional), compute in polynomial time a basis of $d-k$ rational vectors for the subspace parallel to the affine hull (i.e., a basis for the null space of the matrix formed by the row vectors $a_i, i = 1, \ldots, k$), and let Q be the $d \times (d-k)$ matrix with the

basis vectors as columns. Thus, the points of the affine hull are of the form $c + Qv, v \in \mathbb{R}^{d-k}$. The projection $\rho(x)$ of a point $x \in \mathbb{R}^d$ on the affine hull is $\rho(x) = c + Q(Q^TQ)^{-1}Q^T(x-c)$, where the superscript T denotes the transpose of a matrix as usual. Since Q has full column rank, the matrix Q^TQ is invertible. Note that for any point x, the point $\rho(x)$ satisfies the set C_1 of linear equations, and if x satisfies C_1 then $\rho(x) = x$. If $k = 0$ then let Q be the identity matrix I_d and $\rho(x) = x$.

Given a point $x \in B[l,u]$, compute $\theta_i(x) = \max((a_i\rho(x) - a_ic)/(b_i - a_ic), 1)$ for $i = k+1, \ldots, m$, and let $\theta(x) = \max\{\theta_i(x) | i = k+1, \ldots, m\}$. Define $\pi(x) = c + (\rho(x) - c)/\theta(x)$. Note that for every $x \in B[l,u]$, the point $\pi(x)$ satisfies C_1 because both $\rho(x)$ and c satisfy C_1. For $i = k+1, \ldots, m$, we have $a_i\pi(x) = a_ic + a_i(\rho(x) - c)/\theta(x) \leq b_i$. Thus, $\pi(x) \in D_I$. Furthermore, if $x \in D_I$ then $\rho(x) = x$, and $\theta_i(x) = 1$ for all $i = k+1, \ldots, m$; thus, $\theta(x) = 1$ and hence $\pi(x) = x$. The mapping π is clearly continuous; in fact it is easy to see that it is polynomially continuous since all the numbers that enter in the transformation (coefficients of the constraints, entries of Q, etc.) are rationals of polynomial size in $|I|$, and the quantities $b_i - a_ic$ for $i = k+1, \ldots, m$ are at least $r^* \geq 2^{-poly(|I|)}$.

Let F'_I be the function from $B[l,u]$ to $D_I \subset B[l,u]$ obtained by composing π with F_I. The function F'_I is continuous and its fixed points are all in D_I and are precisely the fixed points of F_I. If F_I is polynomially continuous then so is F'_I.

Let g be the linear transformation from the unit cube $[0,1]^d$ to the box $B[l,u]$ that maps each point $y \in [0,1]^d$ to the point $x = g(y)$ defined by $x_i = g_i(y) = (u_i - l_i) \cdot y_i + l_i$. Let g^{-1} be the inverse linear transformation; i.e. g^{-1} maps each point $x \in B[l,u]$ to the point $y \in [0,1]^d$ where $y_i = (x_i - l_i)/(u_i - l_i)$. Let G_I be the function from the unit cube $[0,1]^d$ to itself defined by $G_i = g^{-1} \circ F'_i \circ g$; i.e. a point $y \in [0,1]^d$ is first mapped to the point $x = g(y)$ in $B[l,u]$, then this is mapped to the point $x' = F'_I(x) \in D_I$, and then this is mapped back to the point $y' = g^{-1}(x')$ in $[0,1]^d$. Clearly $y' = y$ iff $x' = x$, and the latter happens only if $x \in D_I$ and $F_I(x) = x$; i.e. there is a 1-1 correspondence between the fixed points of G_I and the fixed points of F_I. Thus, the problem Π of computing a fixed point of F_I reduces to the fixed point problem Φ for G_I, where the function f that maps instances of Π to instances of Φ is just the identity function, and the function that maps solutions of the instance $f(I) = I$ of Φ to solutions of the instance I of Π is the above function g. If Π is polynomially computable, then clearly so is Φ, and if Π is polynomially continuous, then so is Φ.

The analogous reduction to a fixed point problem with domain the unit simplex is similar. Take a large simplex D'_I in $\mathbb{R}^d$ that contains D_I and which is a scaling and translation of the simplex $\Sigma_d = \{y \in \mathbb{R}^d | y \geq 0; \sum_i y_i \leq 1\}$, i.e. D'_I is the convex hull of a point l that is smaller in all coordinates than all points of D_I and the points $l + U \cdot e_i, i = 1, \ldots, d$ where e_i is the unit vector for coordinate i and U is a large enough constant so that $D_I \subset D'_I$; e.g., U is a constant that exceeds $\max\{\sum_i (x_i - l_i) | x \in D_I\}$. Define the mapping π from D'_I to D_I in exactly the same way as above. Let $\hat{G}_I$ be the mapping from Σ_d to itself defined by $\hat{G}_I = \hat{g}^{-1} \circ F_i \circ \pi \circ \hat{g}$; where $\hat{g}$ is the linear transformation $\hat{g}(y) = l + Uy$ for $y \in \Sigma_d$. As in the case of the unit cube, it is easy to see that y is a fixed point of $\hat{G}_I$ iff $\hat{g}(y)$ is a fixed point of F_I. Also, $\hat{G}_I$ is polynomially computable, and if F_I is polynomially continuous then so is $\hat{G}_I$.

There is a 1-1 mapping between the simplex Σ_d and the $(d+1)$-unit simplex $\Delta_{d+1} = \{y \in \mathbb{R}^{d+1} | \sum y_i = 1\}$ where a point $y = (y_1, \ldots, y_d) \in \Sigma_d$ corresponds to the point $(y_1, \ldots, y_d, 1 - \sum_{i=1}^d y_i) \in \Delta_{d+1}$. Composing with $\hat{G}_I$, this induces a mapping G_I from the unit simplex Δ_{d+1} to itself whose fixed points correspond 1-1 to the fixed points of $\hat{G}_I$ and to the fixed points of F_I. ∎

For a fixed point search problem, we are interested in the associated problems mentioned before: compute an exact fixed point if possible, or the decision and approximation problems. We refer to the approximation of an actual fixed point as

Strong Approximation: Given instance I, and rational $\epsilon > 0$ as input, compute a rational vector $x' \in D_I$ such that there exists $x^* \in D_I$, where $x^* = F_I(x^*)$ and $|x^* - x'|_\infty < \epsilon$.

This is in contrast with the following weak version, which is specific to the formulation of the search problem as a fixed point problem:

Weak Approximation: Given instance I and given a rational $\epsilon > 0$ as input, compute a rational vector $x' \in D_I$ such that $|F_I(x') - x'|_\infty < \epsilon$.

Remarks: 1. A search problem may be expressible as a fixed point problem in many different ways, using different functions. For example, besides Nash's function, there are other functions whose fixed points are also the NEs [23, 3]. In general, the notion of a weak approximation depends on the function that is used. The notion of a (strong) approximation does not depend on the function; it is inherent to the search problem itself.

2. As we shall show below, for polynomially continuous functions, strongly approximate fixed points are also weakly approximate with a 'small' change in the approximation error ϵ, in the sense that if we want a weak ϵ-approximation for some rational $\epsilon > 0$ then we can pick an ϵ' of size polynomial in the size of ϵ and the instance such that all strongly ϵ'-approximate fixed points are also weakly ϵ-approximate. The converse does not hold: a weak approximate fixed point may be very far from all actual fixed points, even for polynomially continuous functions. This holds in particular for Nash's functions, as we will see in the next section.

We now give basic facts about the relationship between these different problems.

Proposition 2 *Let $\mathcal{F}$ be the class of functions associated with a fixed point search problem.*

1. *If $\mathcal{F}$ is polynomially continuous, then weak approximation for $\mathcal{F}$ is P-time (many-one) reducible to strong approximation.*

2. *If $\mathcal{F}$ is polynomially continuous and polynomially computable, then weak approximation for $\mathcal{F}$ is in PPAD.*

3. *If $\mathcal{F}$ is polynomially contracting (with respect to any L_k norm) then strong approximation is P-time reducible to weak approximation. Consequently, if $\mathcal{F}$ is both polynomially contracting and polynomially computable, then the strong approximation problem for $\mathcal{F}$ is in PPAD.*

Proof.

1. Given instance I and rational $\epsilon > 0$, we wish to compute a weakly ϵ-approximate fixed point of the function F_I corresponding to I. Let δ be the continuity constant relative to $\epsilon/2$ for the class $\mathcal{F}$, i.e., δ is a rational constant such that for all $x, y \in D_I$, if $|x - y| < \delta$ then $|F_I(x) - F_I(y)| < \epsilon/2$; the size of δ is upper-bounded by $q(|I| + \texttt{size}(\epsilon/2))$, for some polynomial q. Let $\epsilon' = \min(\epsilon/2, \delta)$. Consider any strong ϵ'-approximate FP x' for F_I. Then there is an exact FP, x^*, s.t. $|x' - x^*| < \epsilon'$. Thus, $|F_I(x') - x'| \leq |F_I(x') - F_I(x^*)| + |F_I(x^*) - x^*| + |x^* - x'| < \epsilon/2 + 0 + \epsilon/2 = \epsilon$.
2. By Lemma 1 we can SL-reduce the fixed point problem Π for $\mathcal{F}$ to another fixed point problem Φ for a polynomially continuous and polynomially computable class $\mathcal{G}$ with the unit simplex as the domain. Every instance I of Π, corresponding to the function $F_I \in \mathcal{F}$ with domain $D_I \subset \mathbb{R}^d$ is mapped to the same instance of Φ with corresponding function $G_I = g^{-1} \circ F_I \circ \pi \circ g$ with domain the unit simplex Δ_{d+1}, where g is a separable linear mapping that scales and translates (the first d components of) the unit simplex Δ_{d+1} to a large simplex D'_I that contains D_I; the mapping π maps D'_I to D_I and is the identity on D_I, and g^{-1} is the inverse

mapping from D'_I to the unit simplex Δ_{d+1}. For each coordinate $i = 1, \ldots, d$, $g_i(y) = l_i + Uy_i$ for some vector l and scalar U, which are rational of size polynomial in $|I|$. Note that the range $G_I(\Delta_{d+1})$ of G_I is mapped by g to D_I.

We are given an instance I of the fixed point problem Π and a rational $\epsilon > 0$ and we want to compute a weak ϵ-approximate fixed point of F_I. Let $\epsilon' > 0$ be a (rational) continuity constant of G_I that corresponds to ϵ/U, i.e, for any two points $y, y' \in \Delta_{d+1}$, if $|y - y'| \leq \epsilon'$ then $|G_I(y) - G_I(y')| \leq \epsilon/U$, where the norms are with respect to L_∞. Since U has polynomial size in $|I|$, the rational ϵ' has size polynomial in $|I|$ and $\texttt{size}(\epsilon)$. Suppose that y is a weak ϵ'-approximate fixed point of G_I and let $y' = G_I(y)$. Then $|y - y'| = |y - G_I(y)| \leq \epsilon'$, and thus $|y' - G_I(y')| \leq \epsilon/U$. Let $x' = g(y')$. Since y' is in the range of G_I, the point x' is in D_I, and $F_I(x') = g(G_I(y'))$. Thus, $|x' - F_I(x')| = |g(y') - g(G_I(y'))| \leq U|y' - G_I(y')| \leq \epsilon$. That is, if y is a weak ϵ'-fixed point of G_I, then $x' = g(G_I(y))$ is a weak ϵ-fixed point of F_I. Hence it suffices to show that the weak approximation problem for a polynomially continuous and polynomially computable fixed point problem Φ with the unit simplex Δ_{d+1} as domain is in PPAD.

Index the $d + 1$ coordinates of the unit simplex Δ_{d+1} as $0, 1, \ldots, d$. Let $\epsilon'' = \epsilon'/2(d + 1)$. Pick $\delta = 1/N > 0$ for an integer N, with $\delta \leq \epsilon''$ such that, for all $y, y' \in \Delta_{d+1}$, $|y - y'| < \delta$ implies that $|G_I(y) - G_I(y')| < \epsilon''$. By assumption, $\texttt{size}(\delta)$ is polynomially bounded in $|I|$ and $\texttt{size}(\epsilon'')$, hence in $|I|$ and $\texttt{size}(\epsilon)$. We can now apply Scarf's or Kuhn's algorithm [52, 40] to G_I, using a simplicization where the diameter of the subsimplices is bounded by δ. Specifically, following the regular simplicization of [40], the unit simplex Δ_{d+1} is subdivided into subsimplices using vertices from a regular grid of points whose coordinates are integer multiples of $1/N = \delta$, and each vertex v is 'colored' with a label $l(v) \in \{0, 1, \ldots, d\}$, namely a coordinate l such that $G_I(v)_l < v_l$, say the smallest such coordinate; if v is not a fixed point there is clearly such a coordinate (because the sum of the components of v and of $G_I(v)$ is equal to 1), and if v is a fixed point, we can pick l to be the smallest coordinate with $v_l = \max_i(v_i)$. With this labeling, the $d+1$ unit vectors e_i at the corners of the unit simplex Δ_{d+1} are labeled i, and all the vertices on the facet $y_j = 0$ are labeled with an index $\neq j$. By Sperner's lemma, there is a panchromatic subsimplex in the subdivision, i.e., a subsimplex whose $d+1$ vertices have different colors. It is easy to see that every point y in a panchromatic subsimplex is a weak ϵ'-fixed point of G_I [52]: First note that, for each coordinate i, the subsimplex contains a vertex v such that $G_I(v)_i \leq v_i$, hence $G_I(y)_i - y_i = (G_I(y)_i - G_I(v)_i) + (G_I(v)_i - v_i) + (v_i - y_i)$ $\leq \epsilon'' + \delta$. And since $\sum_i G_I(y)_i = \sum_i y_i = 1$, it follows also that $y_i - G_I(y)_i \leq d(\epsilon'' + \delta)$. Therefore, $|G_I(y) - y| \leq d(\epsilon'' + \delta)$. Since $\epsilon'' = \epsilon'/2(d+1)$ and $\delta \leq \epsilon''$, we have $|G_I(y) - y| < \epsilon'$, i.e., y is a weak ϵ'-fixed point. Note however: a panchromatic subsimplex may not contain any fixed points.

The algorithms of [52, 40, 41] find a panchromatic subsimplex, by first augmenting the simplicization to create an artificial starting subsimplex which has all the colors except one, and then moving from subsimplex to (geometrically) adjacent subsimplex that includes at least the same set of colors until it arrives finally to a panchromatic subsimplex. The algorithms yield readily all the ingredients of the PPAD formulation except for the direction of the edges of the underlying graph $G(I)$, as we will explain below. We will describe the formulation for Kuhn's regular subdivision and algorithm [40], which is somewhat simpler. Take N to be a multiple of d, i.e. $N = md$. First, the unit simplex Δ_{d+1} is augmented by adding one more layer of subsimplices on each side to form the expanded simplex $\bar{\Delta}_{d+1} = \{x | \sum_i x_i = 1, x_i \geq -1/N$ for all $i = 0, \ldots, d\}$. Every new vertex v in the subdivision (i.e. vertex with a negative coordinate) is given as label $l(v)$ the smallest index l for which $v_l = \max_i(v_i)$. Note that the

new layers do not contain any panchromatic subsimplex, because if a vertex v has $v_j \leq 0$ then $l(v) \neq j$.

Each cell C (subsimplex) of the subdivision can be defined by a vertex X_0 of the subdivision (i.e., a vector in $\bar{\Delta}_{d+1}$ whose coordinates are multiples of $1/N$) and a permutation π of $\{1, \ldots, d\}$. The vertices of the cell C are $X_0, X_1, \ldots, X_d$ where each vertex X_i is obtained from the previous by adding $1/N$ in coordinate $\pi(i)$ and subtracting $1/N$ from $\pi(i) - 1$, i.e. $X_i = X_{i-1} + u_{\pi(i)}$, where $u_j = (e_j - e_{j-1})/N$ is the vector that has $1/N$ in coordinate j, $-1/N$ in $j-1$, and 0 elsewhere. We can alternatively view the cell C as having its $d+1$ vertices arranged on a directed cycle where the edges are labelled 1-to-1 by the coordinates, and the difference between two consecutive vertices connected by an edge labelled j is u_j (where $u_0 = (e_0 - e_d)/N$, i.e., arithmetic on the indices is $\mathrm{mod}(d+1)$); X_0 is the vertex with incoming edge labelled 0, and the cycle is $X_0 \stackrel{\pi(1)}{\to} X_1 \stackrel{\pi(2)}{\to} X_2 \ldots \stackrel{\pi(d)}{\to} X_d \stackrel{0}{\to} X_0$. Note that $\sum_{j=0}^{d} u_j = 0$, and hence $X_0 = X_d + u_0$. For every cell C of the subdivision there is a unique corresponding cycle and a unique pair (X_0, π). Conversely, for every $X_0 \in \bar{\Delta}_{d+1}$ whose coordinates are multiples of $1/N$ and every permutation π of $\{1, \ldots, d\}$, we can generate the sequence $X_0, X_1, \ldots, X_d$; if all the points are in $\bar{\Delta}_{d+1}$ then they form a cell in the subdivision.

The problem is formulated in PPAD as follows. Let $col(C)$ for a cell C denote the set of colors of its vertices. The set $S(I)$ of candidate solutions, which is the set of nodes of the graph $G(I)$ for the instance I, is the set of cells C such that $\{0, \ldots, d-1\} \subseteq col(C)$. Each cell is described by a string of polynomial length giving the initial vertex X_0 of the cell and the permutation π of $\{1, \ldots, d\}$. Furthermore, given a pair (X_0, π) we can compute in polynomial time the sequence $X_0, X_1, \ldots, X_d$, determine if (X_0, π) represents indeed a cell, evaluate the function G_I on the vertices to compute their colors, and thus determine whether the cell is in $S(I)$. The artificial starting candidate solution is the cell C_0 represented by the pair (X_0, π) with $X_0 = (\frac{m+1}{N}, \frac{m}{N}, \frac{m}{N}, \ldots, \frac{m}{N}, -\frac{1}{N})$ (recall that $N = md$) and $\pi = 1, 2 \ldots, d$. Each vertex X_i, $i = 0, 1 \ldots, d-1$ of the cell C_0 has $\frac{m+1}{N}$ in coordinate i, $-\frac{1}{N}$ in coordinate d, and $\frac{m}{N}$ in the rest of the coordinates; thus $l(X_i) = i$. The final vertex X_d is $(\frac{m}{N}, \frac{m}{N}, \ldots, \frac{m}{N}, 0)$ and thus has some color $l(X_d) \neq d$; we can assume without loss of generality that $l(X_d) = 0$ (we can index the coordinates at the beginning so that this holds). Thus, $col(C_0) = \{0, \ldots, d-1\}$.

The edges of the graph $G(I)$ connect two cells in $S(I)$ iff they share a facet whose d vertices have colors $\{0, \ldots, d-1\}$. We will specify the directions of the edges later. Note that every cell (subsimplex) C with $col(C) = \{0, 1, \ldots, d-1\}$ has two vertices that have the same color, and thus it has two facets f_1, f_2 with colors $\{0, 1, \ldots, d-1\}$, namely the two facets obtained by omitting one of the two vertices. If neither of the two facets is on the boundary of $\bar{\Delta}_{d+1}$, then there are two (geometrically) adjacent cells C_1', C_2' that share respectively the facets f_1, f_2 with C; both cells C_1', C_2' belong to $S(I)$ and they are the two nodes adjacent to C in the graph. If one of the facets, say f_1, lies on the boundary of $\bar{\Delta}_{d+1}$, then it is easy to see that C must be the starting cell C_0: Since the vertices of facet f_1 are labeled $0, \ldots, d-1$, these coordinates cannot be negative, thus f_1 must lie on the boundary plane $x_d = -\frac{1}{N}$ of $\bar{\Delta}_{d+1}$. At all the vertices of f_1, the coordinates $0, \ldots, d-1$ sum to $1 + \frac{1}{N} = \frac{md+1}{N}$, and since each one of them is the maximum coordinate at some vertex of f_1 and differs at most by $\frac{1}{N}$ in the other vertices, it follows that each vertex of f_1 has value $\frac{m+1}{N}$ in one of the first d coordinates and $\frac{m}{N}$ in the rest. That is, the vertices of f_1 are precisely the vertices $X_0, \ldots X_{d-1}$ of the starting cell C_0, and hence the other vertex of C must be $X_0 = (\frac{m}{N}, \ldots \frac{m}{N}, 0)$, and $C = C_0$. In this case there is of course only one adjacent cell. If C is panchromatic, then there is also only one adjacent cell. Thus, the only nodes in the graph $G(I)$ that have only one adjacent

node are the starting cell C_0 and the panchromatic cells.

For a cell $C \in S(I)$ and a facet f of C with $col(f) = \{0, \ldots, d-1\}$ the adjacent cell C' that shares f can be determined as follows. Let w be the vertex of C that is not in f, and let $w \xrightarrow{i} v_1 \rightarrow \ldots v_d \xrightarrow{j} w$ be the cycle corresponding to C with the edges labelled uniquely by the coordinates. Thus the facet $f = \{v_1, \ldots, v_d\}$. Let $w' = w + u_i - u_j$ (which is also equal to $v_1 - u_j$ and $v_d + u_i$). The adjacent cell C' corresponds to the cycle $w' \xrightarrow{j} v_1 \rightarrow \ldots v_d \xrightarrow{i} w'$, i.e. w is replaced by w' and the labels of the two adjacent edges are swapped. Thus for example, for $C = C_0$ the starting artificial cell with cycle $X_0 \xrightarrow{1} X_1 \rightarrow \ldots X_d \xrightarrow{0} X_0$, the face f to the adjacent cell is $\{X_1, \ldots, X_d\}$, and the adjacent cell is $w' \xrightarrow{0} X_1 \rightarrow \ldots X_d \xrightarrow{1} w'$ where $w' = (\frac{m-1}{N}, \frac{m+1}{N}, \frac{m}{N}, \ldots, \frac{m}{N}, 0)$.

The algorithms by Scarf, Kuhn et. al. determine the direction of the edges encountered by the algorithm online, based on the history of the execution: the algorithm remembers from which edge it came into a cell, and this determines the outgoing edge. This is not appropriate for the PPAD formulation: The edges of a node (cell), including the direction, have to be computed only from the representation of the node; if we include an additional bit in the representation indicating from which edge the node was entered, this effectively doubles the number of nodes and creates a second easily computable endpoint of a path, namely the copy of the artificial starting node with the adjacent edge coming in. Thus, we need a different way to specify the directions that is based only on the cell representation.

We define the direction of the edges as follows. As above, let $C \in S(I)$ be a cell, f a facet of C with $col(f) = \{0, \ldots, d-1\}$, w the vertex of C that is not in f, $w \xrightarrow{i} v_1 \rightarrow \ldots v_d \xrightarrow{j} w$ the cycle corresponding to C. Define two permutations σ, τ as follows: σ is the permutation of the colors $0, \ldots, d-1$ as they appear in the sequence of the vertices $v_1, \ldots, v_d$ of the facet f; τ is the permutation of the coordinates $0, 1, \ldots, d$ as they occur on the labels of the edges along the cycle starting from the vertex w that is not on the facet. We direct the edge (C, C') of C corresponding to facet f out of C (i.e. $C \rightarrow C'$) if the two permutations σ, τ have opposite sign (one is even and the other is odd), and we direct it into C if they have the same sign. For example, for the edge incident to the starting node C_0, the permutations are $\sigma = 1, 2, \ldots, d-1, 0$ and $\tau = 1, 2, \ldots, d, 0$ which have opposite signs, thus the edge is directed out of C_0. Clearly, the directions of the edges are also easy to compute.

We have to show now that the definition is consistent and has the desired properties:
Property 1: If the above construction computes for a node C an edge $C \rightarrow C'$ or $C' \rightarrow C$ then the construction computes for the other node C' the same edge directed in the same way.
Property 2: If a node C has two incident edges then one edge is directed into C and the other out of C.

Proof of Property 1. Using our previous notation, let $w \xrightarrow{i} v_1 \rightarrow \ldots v_d \xrightarrow{j} w$ be the cycle for cell C with facet $f = \{v_1, \ldots, v_d\}$. Then $w' \xrightarrow{j} v_1 \rightarrow \ldots v_d \xrightarrow{i} w'$ is the cycle for cell C' where $w' = w + u_i - u_j$. At cell C', we will have an edge connecting it to a neighboring cell C'' with cycle $w'' \xrightarrow{i} v_1 \rightarrow \ldots v_d \xrightarrow{j} w''$ where $w'' = w' + u_j - u_i = w$. Thus, $C'' = C$. The color permutation σ' for C' and f is clearly the same as the permutation σ for C and f. The coordinate permutation τ' for C' and f is obtained from the permutation τ for C, f by transposing i and j, thus it has opposite sign. So the edge will be directed at node C' consistently with C.

Proof of Property 2. Let $w \xrightarrow{i} v_1 \rightarrow \ldots v_d \xrightarrow{j} w$ be the cycle for cell C and facet $f = \{v_1, \ldots, v_d\}$

with $col(f) = \{0, \ldots, d-1\}$. If w has color d then C is panchromatic and there is only one incident edge. The same is true if $C = C_0$. So suppose $C \neq C_0$ and w has the same color as v_k. Then there is another facet f' with $col(f') = \{0, \ldots, d-1\}$, namely $f' = v_{k+1} \rightarrow \ldots v_d \rightarrow w \rightarrow \ldots v_{k-1}$ and the cell C has a second edge corresponding to this facet f'. The coordinate permutation τ' for the facet f' is the ordering of the $d+1$ edge labels of C starting from v_k; thus, τ' is obtained from the permutation τ for f by shifting cyclically k places. If d is even (i.e. $d+1$ is odd) then $sign(\tau') = sign(\tau)$ (every cyclic shift is even) and if d is odd then $sign(\tau') = sign(\tau)(-1)^k$. The color permutation σ' for the facet f' is obtained from the permutation σ for f by first moving the color $l(v_k)$ to the beginning (i.e. $k-1$ transpositions) and then shifting cyclically left k places the d colors. If d is odd then $sign(\sigma') = sign(\sigma)(-1)^{k-1}$ (because the cyclic shifts don't change the sign). If d is even, then $sign(\sigma') = sign(\sigma)(-1)^{k-1}(-1)^k = -sign(\sigma)$.

If d is even we have $sign(\tau')sign(\sigma') = -sign(\tau)sign(\sigma)$. If d is odd we have $sign(\tau')sign(\sigma') = sign(\tau)(-1)^k sign(\sigma)(-1)^{k-1} = -sign(\tau)sign(\sigma)$. Thus, in both cases, σ' and τ' have the same sign iff σ and τ have opposite signs. Therefore the edge of C corresponding to f' is directed in the opposite way than the edge corresponding to f.

3. We know that for each $F_I \in \mathcal{F}$, there is a contraction factor β_I, $0 \leq \beta_I < 1 - 2^{-q(|I|)}$, for some polynomial $q()$, such that for all $x, y \in D_I$, $|F_I(x) - F_I(y)|_k < \beta_I |x-y|_k$. Let x^* be the Banach fixed point of F_I. We have $|x - x^*|_k \leq |F_I(x) - x|_k + |F_I(x) - x^*|_k$. But since $F(x^*) = x^*$, we have: $|x - x^*|_k \leq |F_I(x) - x|_k + \beta_I |x - x^*|_k$. Hence $(1-\beta_I)|x - x^*|_k \leq |F_I(x) - x|_k$. Thus $|x - x^*|_k \leq (1/(1-\beta_I))|F_I(x) - x|_k$. In other words, if we want to find a vector x' such that $|x' - x^*|_k < \epsilon$, then it suffices to compute an x' such that $|F_I(x') - x'|_k < \epsilon(1-\beta_I)$. Recall that for any vector $z \in \mathbb{R}^n$, $|z|_\infty \leq |z|_k \leq \sqrt[k]{n}|z|_\infty$. Since $(1-\beta_I) \geq 2^{-q(|I|)}$, let $\epsilon' = (\epsilon/(\lceil \sqrt[k]{n} \rceil))2^{-q(|I|)}$, where n is the dimension of vectors x and $F_I(x)$. Then for every point x' such that $|F_I(x') - x'|_\infty < \epsilon'$, we have $|x^* - x'|_\infty \leq |x^* - x'|_k < \epsilon$. Thus a weak ϵ'-approximate fixed point of F_I, is a strong ϵ-approximate fixed point of F_I.

■

In many search problems, which are formulated as fixed point problems for a class of functions, the functions are just tools and weak approximation does not a priori have any significance for the search problem itself. In the particular case of games and Nash's function, the weak approximation is relevant as it is closely related to the notion of ϵ-Nash equilibria.

Proposition 3 *Computing an ϵ-NE (for a given game, Γ, and given $\epsilon > 0$) is P-time equivalent to computing a weak ϵ'-approximate fixed point of Nash's function (for a given instance of Nash's function, F_Γ, and given $\epsilon' > 0$).*

Proof. For the one direction, suppose that we are given a (normal form) game Γ and a rational $\epsilon' > 0$, and we want to compute a weak ϵ'-approximate fixed point of Nash's function F_Γ. Let n be the maximum number of pure strategies for a player, and let $\epsilon = \epsilon'/(n+1)$. Suppose that x is an ϵ-NE of Γ. This implies that $g_{i,j}(x) \leq \epsilon$ for all i, j. Recall that $F_\Gamma(x)_{(i,j)} \doteq \frac{x_{i,j} + \max\{0, g_{i,j}(x)\}}{1 + \sum_{l=1}^{m_i} \max\{0, g_{i,l}(x)\}}$. Thus $F_\Gamma(x)_{(i,j)} + F_\Gamma(x)_{(i,j)}(\sum_{l=1}^{m_i} \max\{0, g_{i,l}(x)) = x_{i,j} + \max\{0, g_{i,j}(x)\}$, and so $|F_\Gamma(x)_{(i,j)} - x_{i,j}| = |\max\{0, g_{i,j}(x)\} - F_\Gamma(x)_{(i,j)}(\sum_{l=1}^{m_i} \max\{0, g_{i,l}(x))| \leq (m_i + 1)\epsilon \leq \epsilon'$. Therefore, x is a weak ϵ'-approximate fixed point.

For the other direction, we show that there is a polynomial $q(z)$, such that for any normal form game Γ, for each rational $\epsilon > 0$ there is a $\epsilon' > 0$ such that $\texttt{size}(\epsilon') \leq q(\texttt{size}(\epsilon) + \texttt{size}(\Gamma))$, and such that if x' is a weak ϵ'-FP of Nash's function F_Γ, then x' is a ϵ-NE.

For a given game Γ, let M denote the maximum difference, over all players i, between the maximum payoff value and minimum payoff value for player i under any pure strategy profile. Let $c = 1 + M$, and let n be the maximum number of pure strategies of a player. Suppose that we are given a rational $\epsilon > 0$ and we want to compute an ϵ-NE for Γ. Assume without loss of generality that $\epsilon \leq 1$. Take $\epsilon' = \epsilon^2/(4c^2n^3)$, and let x be a weak ϵ'-approximate fixed point of Nash's function, i.e., $|F_\Gamma(x) - x|_\infty < \epsilon'$. Let $\phi_{i,j}(\mathbf{x}) = \max\{0, g_{i,j}(\mathbf{x})\}$. Note that we simply need to show that $\phi_{i,j}(\mathbf{x}) \leq \epsilon$ for all i and j. This guarantees that no player can switch to any other (pure or mixed) strategy unilaterally and improve its payoff by more than ϵ. Since $|F_\Gamma(x) - x|_\infty < \epsilon'$, using the definition of F_Γ, we have for every player i and each strategy j, $|\phi_{i,j}(x) - x_{i,j} \sum_l \phi_{i,l}(x)| < \epsilon'(1 + \sum_l \phi_{i,l}(x)) < \epsilon^2/2cn^2$. Let $\psi_{i,j}(x) = -g_{i,j}(x)$ for those i, j such that $g_{i,j}(x) \leq 0$. Since $\sum_{j=1}^{m_i} x_{i,j} g_{i,j}(x) = 0$, we have $\sum x_{i,j}\phi_{i,j}(x) = \sum x_{i,j}\psi_{i,j}(x)$, where the sum on the left hand side is over those strategies j with $\phi_{i,j} > 0$ and on the right hand side over those strategies j with $\psi_{i,j}(x) > 0$. (A strategy that is not in the support of x is in neither side.) If one of the strategies j on the rhs has $x_{i,j} \geq \epsilon/2cn^2$ then we know $(\epsilon^2/(2cn^2)) > |\phi_{i,j}(x) - x_{i,j} \sum_l \phi_{i,l}(x)| = |0 - x_{i,j} \sum_l \phi_{i,l}(x)| \geq (\epsilon/2cn^2) \sum_l \phi_{i,l}(x)$. Therefore, $\sum_l \phi_{i,l}(x) < \epsilon$. So assume all the strategies on the rhs have $x_{i,j} < \epsilon/2cn^2$. Then the rhs (and hence also the lhs) is $< \epsilon/2n$. Suppose wlog that $j = 1$ gives the maximum $\phi_{i,j}(x)$. If $x_{i,1} \leq 1/2n$ then $\epsilon^2/2cn^2 > \phi_{i,1}(x) - x_{i,1} \sum_l \phi_{i,l}(x) \geq \phi_{i,1}(x) - (1/2)\phi_{i,1}(x) = (1/2)\phi_{i,1}(x)$, so $\phi_{i,1}(x) < \epsilon^2/n^2 < \epsilon$. On the other hand, suppose $x_{i,1} > 1/2n$. Then the lhs is $> (1/2n)\phi_{i,1}(x)$, and we know that the lhs is smaller than $\epsilon/2n$, therefore $\phi_{i,1} < \epsilon$, and we are done. ∎

An immediate corollary of Proposition 2, part 2, and Proposition 3, is that computing an ϵ-NE is in PPAD ([13, 50]). We will use Proposition 2, parts 2 and 3, in Sections 5 and 6 to place several more problems in PPAD, and to give a fixed point characterization of PPAD.

3 Nash Equilibria are as hard as the Square Root Sum Problem and Arithmetic Circuit Decision Problems

Recall the problems SQRT-SUM and PosSLP from Section 1. In the SQRT-SUM problem we are given positive integers $d_1, \ldots, d_n$ and k, and we want to determine if $\sum_{i=1}^n \sqrt{d_i} \leq k$. In the PosSLP problem, we are given a division-free straight-line program, or equivalently, an arithmetic circuit with operations $+, -, *$ and inputs 0 and 1, and a designated output gate, and we want to determine if the integer N that is the output is positive. Formally, the input arithmetic circuit is a sequence of gates $g_0, g_1, \ldots, g_m$, where the first two gates read in the constants 0,1, i.e., $g_0 := 0$, $g_1 := 1$, and the rest of the gates perform (wlog binary) operations, i.e., $g_i := g_j \circ g_k$, with $j, k < i$, where the operator is $\circ \in \{+, -, *\}$; the final gate g_m computes the output number N. The best upper bound known for them is the (4th level of the) Counting Hierarchy [1] (i.e. just below PSPACE), and no lower bound is known (i.e., no hardness result above polynomial time is known).

In this section we will reduce these problems to the problem of computing a Nash equilibrium of a given game. Then we will present a number of consequences of the reductions. Recall the definition of the Promised Gap Decision Problem PGD(a, b) for a search problem, given in Section 2: Given instance I, rationals $a < b$, and the promise that all solutions $x \in Sol(I)$ have $x_1 \leq a$ or they all have $x_1 > b$, determine which of the two is the case.

Theorem 4

1. *The SQRT-SUM and PosSLP problems are P-time (many-one) reducible to the promised-gap-decision problem PGD(0,1) for 4-player Nash, even restricted to games that have a unique*

Nash equilibrium. That is, given a game that is guaranteed to have a unique Nash equilibrium, and such that the unique NE plays a certain pure strategy either with probability 0 or probability 1, it is SQRT-SUM-hard and PosSLP-hard to distinguish between the two cases.

2. *The SQRT-SUM and PosSLP problems are P-time reducible to the PGD(0,1 − ϵ) problem for 3-player Nash Equilibria, for any constant $\epsilon > 0$ (and even with $\epsilon = 2^{-poly}$). Furthermore, this holds even for the restriction to games with a unique Nash equilibrium.*

Results analogous to those stated in Theorem 4 also hold for approximation of *the* equilibrium payoff of one player, in a game with a unique NE.

We will first prove Theorem 4 for PosSLP, and then for SQRT-SUM. We are given an arithmetic circuit C over basis $\{+, -, *\}$ with fan-in 2, with inputs 0, 1, and a single output gate, and we want to determine whether the value of the circuit, $val(C)$, is positive. The structure of the reductions will be as follows. We first derive from C another circuit S over $\{+, *, /\}$ (no subtraction, but with division) with input 1/2, and with two output gates $out1, out2$, which has the following properties: (i) all gates of the circuit S have values in the (open) interval $(0, 1)$, and (ii) the values of the two output gates are not equal, and $val(C) > 0$ iff $val(out1) > val(out2)$. Second, we construct from S a game Γ with three players which has a unique Nash equilibrium, the equilibrium is fully mixed, and its value (the probability) on two distinguished pure strategies is proportional to $val(out1)$ and $val(out2)$, i.e., $val(out1)/K$ and $val(out2)/K$ for some K. This part of the reduction is done in two substeps. First we construct from S a game G with a fixed number of players that has these properties (the precise number of players is 10, but it is not important) and then we use a transformation from many players to three players that preserves the desired properties. In the third and final step we have a separate transformation for each of the two claims in Theorem 4 from the game Γ to a suitable 4-player or 3-player game.

Step 1: Transformation of the circuit.

We are given a circuit C over basis $\{+, -, *\}$ with inputs 0, 1. First, multiply the output by 2 (=1+1) and subtract 1 (as in [1]) to obtain a circuit C' that has nonzero value: $val(C') > 0$ if $val(C) > 0$ and $val(C') < 0$ if $val(C) \leq 0$. Then transform the circuit C' to a circuit where subtraction occurs only at the output gate, and which computes the same value: Replace every gate g of C with two gates g^+ and g^- for the positive and negative parts, such that $g = (g^+ - g^-)$. Viewing the transformation bottom-up, the input gates $g_0 := 0$ and $g_1 := 1$ are replaced by $g_0^+ := 0$, $g_0^- := 0$, $g_1^+ := 1$, $g_1^- := 0$. Each addition gate $g_k := g_i + g_j$ in the original circuit is replaced by two gates: $g_k^+ := g_i^+ + g_j^+$ and $g_k^- := g_i^- + g_j^-$. Likewise, a subtraction gate $g_k := g_i - g_j$ is replaced by $g_k^+ := g_i^+ + g_j^-$ and $g_k^- := g_i^- + g_j^+$. Finally, a multiplication gate $g_k := g_i * g_k$ can be replaced by $g_k^+ := (g_i^+ * g_j^+) + (g_i^- * g_j^-)$ and $g_k^- := (g_i^+ * g_j^-) + (g_i^- * g_j^+)$ (note that we need two multiplication gates and one addition gate for each of these). Clearly, the transformation only blows up the circuit linearly. For the output gate, g_{out}, we could add a subtraction gate $g_{out} := g_{out}^+ - g_{out}^-$, thus computing the same output as the original circuit. Let C'' be the circuit without this subtraction gate g_{out}, and which has g_{out}^+, g_{out}^- as its two output gates. Clearly C'' uses only $\{+, *\}$, the two output gates have unequal values, and $val(g_{out}^+) > val(g_{out}^-)$ iff $val(C) > 0$. All gates of C'' compute nonnegative values.

Propagate the 0 inputs in the circuit C'' in a straightforward way, to identify and eliminate all gates that have 0 value. Also shortcut and remove all addition gates that have fan-in 1 (i.e., addition gates that have an operand with value 0). Of course, if one of the output gates has value 0 and is eliminated in this process, then we are done, so we may assume that this is not the case. Thus, we are left with a circuit D over $\{+, *\}$ with two output gates that has only input 1, and all the gates of D have (strictly) positive values.

Let d be the depth of the circuit D. A circuit over $\{+, *\}$ of depth d with input 1 can only generate numbers of double exponential magnitude. Specifically, the largest number that can be generated is $2^{2^{d-1}}$, by computing $1+1$ and then squaring it successively $d-1$ times. Let $t = 1/2^{2^d}$. We can compute t by a circuit T of size $d+1$ with input $1/2$ by successively squaring it d times, i.e. let T be the circuit (SLP) $h_0 := 1/2, h_1 := h_0 * h_0, \ldots, h_d := h_{d-1} * h_{d-1}$.

The circuit S consists of the circuit T followed by a transformed version of D, where each gate g_k of D is transformed to a gate g'_k as follows. The input gate $g_1^+ := 1$ of D is transformed to $g'_1 := h_d$. Each addition gate $g_k := g_i + g_j$ of D is transformed to $g'_k := g'_i + g'_j$ in S. Each multiplication gate $g_k := g_i * g_j$ is transformed to $g'_k := g'_i * g'_j / h_d$; in more detail, we break up the expression into two steps, performing first the multiplication, followed by the division (the order is important): $g''_k := g'_i * g'_j; g'_k := g''_k / h_d$. The output gates $out1, out2$ of S are the gates $(g^+_{out})', (g^-_{out})'$ corresponding to the output gates of D.

Clearly, the size of S is linear in the size of the given circuit C. We show now that S has the claimed properties for Step 1.

Lemma 5 *The circuit S has the following properties.*

1. *All the gates have values in the open interval $(0, 1)$.*
2. *The two output gates $out1, out2$ have unequal values. Furthermore, $val(out1) > val(out2)$ iff $val(C) > 0$.*

Proof. A straightforward induction shows that the value of h_i for $i = 0, 1, \ldots, d$ is $1/2^{2^i}$. Thus, $val(h_d) = 1/2^{2^d} = t$.

We show that the value of each gate g'_k in S is t times the value of the corresponding gate g_k in S, i.e. $val(g'_k) = t \cdot val(g_k)$. This can be shown easily by induction on the height of the gate. The basis holds because the assignment gate $g_1^+ := 1$ of D was transformed to $g'_1 := h_d$ in S. The induction step for an addition gate $g_k := g_i + g_j$ follows immediately from the induction hypothesis. For a multiplication gate $g_k := g_i * g_j$ we have $val(g'_k) := val(g'_i) \cdot val(g'_j)/val(h_d)$ $= (t \cdot val(g_i)) \cdot (t \cdot val(g_j))/t = t \cdot val(g_k)$.

Since all the gates g_k of D have positive value, smaller than 2^{2^d}, it follows that all the corresponding gates g'_k of S have value in the interval $(0, 1)$. Furthermore, this holds also for the intermediate gates g''_k that are introduced by the multiplication gates, because $g''_k := g'_i * g'_j$ is the product of two positive numbers smaller than 1. It clearly holds also for the gates h_i, so it holds for all the gates of S. Furthermore, the values of the output gates of S satisfy part 2 because they are proportional to their values in D, which satisfy the claim. ∎

Step 2: Construction of the game. We proceed now to Step 2 of the reduction, which consists of two substeps. In substep 2a we will construct a 10-player game G from S, and in substep 2b we will reduce it to a 3-player game Γ. The game G has 10 players, that are paired in 5 pairs i, i' for $i = 1, \ldots, 5$. The strategies of each unprimed player i are partitioned into blocks, and the corresponding primed player i' has one strategy for each block of i. If n is the number of gates of the circuit S, then players 1,2,3 have n blocks each, and the primed players $1', 2', 3'$ have a corresponding set of n strategies each; players 4,5 have $3n$ blocks each and players $4', 5'$ have $3n$ strategies. The sole purpose of the primed players is to ensure that in any Nash equilibrium of the game, the probability of each unprimed player is divided evenly among its blocks. The 'matching pennies' game is a simple way of enforcing this (various versions of this game have been used in reductions before, e.g. [13, 10]). In our game, the payoff functions of the players are defined so that the following holds. Given a pure strategy profile $s = (s_1, \ldots, s_{5'})$, the payoff function $u_i(s)$

of each unprimed player i consists of two terms, $u_i(s) = \mu_i(s) + \mu'_i(s)$ where the first term $\mu_i(s)$ depends only on the strategies of the unprimed players, and the second term $\mu'_i(s)$ depends only on the strategies of i and i'; in particular $\mu'_i(s) = M$ if the block of the strategy of i corresponds to the strategy of i', and $\mu'_i(s) = 0$ otherwise, where M is much greater than the maximum absolute value of $\mu_i(s)$ over all pure strategy profiles s, say $M > 6n \max_s |\mu_i(s)|$. The payoff $u_{i'}(s)$ of the primed player i' is $-M$ if the strategy of i' corresponds to the block of the strategy of i, and is 0 otherwise.

Lemma 6 *Suppose that a game G has a pair of players i, i', such that the (pure) strategies of player i are partitioned into blocks, the blocks are in 1-1 correspondence with the (pure) strategies of player i' and the payoffs of i, i' are of the form given above. Then every Nash equilibrium of G has the following properties.*

1. *The total probability allocated to the strategies in each block of player i is the same for all the blocks.*

2. *All strategies of player i' have nonzero probability.*

Proof. 1. Let x be any Nash equilibrium of G. Suppose that x gives unequal total probability to some blocks of player i, and let R be the set of blocks that have maximum probability. Then the strategies of player i' that correspond to blocks in R must have 0 probability in x, because otherwise i' can improve his expected payoff. The expected payoff for player i of any strategy in a block in R with respect to the rest of the profile x is at most $\max_s |\mu_i(s)|$, whereas the expected payoff of any strategy in a block that corresponds to a maximum probability strategy of i' is at least $M/3n - \max_s |\mu_i(s)|$, since i' has at most $3n$ strategies. Since $M > 6n \max_s |\mu_i(s)|$, all the strategies of player i that belong to a block in R must have probability 0, contradicting the assumption that R is the set of blocks that have maximum probability. Therefore all the blocks of player i have the same total probability in x.

2. If some strategy of player i' has probability 0 in x, then all the strategies of player i in the corresponding block must have probability 0 in x, which contradicts part 1. ∎

It remains to define now the strategies and payoffs μ_i of the unprimed players $1, \ldots, 5$. Players 1,2,3 are called the *primary* players, and 4,5 are the *auxiliary* players. Let us order the gates of the circuit S topologically, and denote them as $q_1, \ldots, q_n$, where the first gate is the input assignment $q_1 := 1/2$, all the other gates correspond to operations $+, *, /$, with the last two gates being the output gates $out1, out2$. The primary players have n blocks each, corresponding to the n gates, and players 4 and 5 have $3n$ blocks each, one block for each primary player and each gate. Each block of each player contains two strategies. We will index the strategies of a primary player $i = 1, 2, 3$ with triples (i, j, b) where $j = 1, \ldots, n$ gives the index of the block of i (and the corresponding gate of S) and $b = 0, 1$. We will index the strategies of an auxiliary player $i = 4, 5$ with 4-tuples (i, k, j, b), where $k = 1, 2, 3$ corresponds to a primary player, $j = 1, \ldots, n$ corresponds to a gate of S, and $b = 0, 1$; the pair (k, j) gives the block of the strategy (i, k, j, b). The game G will be defined so that it has a unique Nash equilibrium, and the probability of each strategy (i, j, b) of a primary player $i = 1, 2, 3$ in the NE is equal to $val(q_j)/n$ if $b = 0$, and is equal to $(1 - val(q_j))/n$ if $b = 1$.

The payoff functions μ_i of the unprimed players are obtained by superimposing 'gadget' subgames for the gates of the circuit S, in particular we have one gadget for each primary player and gate. The gadget (subgame) G_{ij} corresponding to primary player $i = 1, 2, 3$ and gate $q_j, j = 1, \ldots, n$ *affects* the payoff of player i iff he plays a strategy in his j-th block (i.e. $(i, j, 0)$ or $(i, j, 1)$), and it affects the payoff of an auxiliary player $l = 4, 5$ iff he plays a strategy in his (i, j) block (i.e.

$(l, i, j, 0)$ or $(l, i, j, 1)$). Thus, the payoff μ of each (unprimed) player for each of his own strategies is affected by exactly one gadget G_{ij}, namely the one corresponding to the block of the strategy. Let $Af(i, j)$ be the set of strategies affected by gadget G_{ij}.

Intuitively, the purpose of the gadget G_{ij} is to implement the operation of gate q_j of S and assign the correct value to the two strategies in the j-th block of primary player i. One way to do this is to use gadgets similar to [13] for the $+, *$ operations, and build another gadget for division (this is what we did in an earlier proof, as hinted in the conference version of this paper [21]). We will instead provide here one parameterized gadget that works for all the operations, including division (as well as for additional operations like roots and fractional powers, as we will see later on). Besides its elegance and generality, the gadget has the important added benefit that it ensures uniqueness of the NE. The gadget is based on a construction by Bubelis (Theorem 2 in [7]). We state the properties of the gadget here in a form that will be useful to us, for this and subsequent reductions. We include the details of the construction for completeness, and to prove the properties that we will need.

Lemma 7 *[7] Let $f(x) = c_0 x^m + \ldots + c_m$ be a polynomial with $f(0) \leq 0$ and $f(1) \geq 0$, and which has a unique root α in the interval $[0, 1]$. Then we can construct a game G_f with three players, a primary player 1 and two auxiliary players 2,3, where the primary player has two strategies 0,1, and the auxiliary players each have a set $\{0, 1, \ldots, m\}$ of $m + 1$ strategies. The game G_f has the following properties. Every Nash equilibrium assigns probability α to the strategy 0 of the primary player (and $1 - \alpha$ to the strategy 1). Furthermore, if $\alpha \neq 0, 1$, then the game has a unique Nash equilibrium, and the NE is fully mixed. The payoff functions of the auxiliary players for each pure strategy profile do not depend on the polynomial f and are 0,1, or -1; the payoff function of the primary player for a pure strategy profile (s_1, s_2, s_3) is c_{s_2} if $s_1 = 1$ and 0 if $s_1 = 0$.*

Proof. Our statement here differs somewhat from the statement of Theorem 2 of [7], which concerns the payoff of a player rather than the probabilities of the NE, but the construction is the same. As indicated in the statement of the lemma, the primary player 1 has two strategies 0,1, and the auxiliary players 2,3 have each $m + 1$ strategies $0, 1, \ldots, m$. Let $s = (s_1, s_2, s_3)$ be a pure strategy profile. We use the following notation: If P is a predicate of the profile s, we will use $\chi(P(s))$ to denote the indicator function of the predicate, i.e., $\chi(P(s)) = 1$ if $P(s)$ is true and $\chi(P(s)) = 0$ if $P(s)$ is false. The payoff functions of the players, denoted h_i^f, are as follows.
Player 1: $h_1^f(s) = c_{s_2}$ if $s_1 = 1$, and 0 otherwise (i.e., if $s_1 = 0$).
Player 2: $h_2^f(s) = 1$ if $s_2 = s_3$, and 0 otherwise.
Player 3: $h_3^f(s) = \chi(s_2 + 1 \equiv s_3 \mod (m+1)) - \chi(s_1 = 0)\chi(s_2 = s_3) - \chi(s_1 = 1)\chi(s_3 = 0)$. In other words: If $s_1 = 0$ then player 3 receives payoff 1 if $s_2 + 1 \equiv s_3 \mod (m + 1)$, payoff -1 if $s_2 = s_3$, and 0 otherwise. If $s_1 = 1$ then player 3 receives payoff 1 if $s_2 + 1 \equiv s_3 \mod (m+1)$ and $s_3 \neq 0$, payoff -1 if $s_3 = 0$ and $s_2 \neq m$, and payoff 0 otherwise.

Note that the payoffs of the auxiliary players do not depend on the polynomial f, so we can omit the superscript f and simply denote them as $h_i(s), i = 2, 3$.

We analyze now the Nash equilibria of the game. We follow the proof in [7]. Consider a Nash equilibrium $x = (x_1, x_2, x_3)$. We shall show that $x_1(0) = \alpha$ (and $x_1(1) = 1 - \alpha$). Let $\xi = x_1(0)$. The mixed strategy of player 1 induces a subgame G'_ξ for players 2 and 3, and clearly (x_2, x_3) must be a NE for this subgame. Let h'_2, h'_3 be the payoff functions of players 2,3 in this subgame. For a given pure strategy profile $s' = (s_2, s_3)$ for players 2 and 3, the payoff $h'_2(s')$ of player 2 is $h'_2(s') = 1$ if $s_2 = s_3$, and 0 otherwise. The payoff $h'_3(s')$ of player 3 is $h'_3(s') = \chi(s_2 + 1 \equiv s_3 \mod (m+1)) - \xi\chi(s_2 = s_3) - (1 - \xi)\chi(s_3 = 0)$.

Let v_2, v_3 be the expected payoffs for players 2,3 respectively in the equilibrium (x_2, x_3) of G'_ξ (and in the NE x of G_f). Then the expected payoff of each pure strategy of player 2 (resp. 3) is upper bounded by v_2 (resp. v_3), with equality if the strategy is in the support of x. For player 2 we get the inequalities $x_3(j) \leq v_2$ for the strategies $j = 0, \ldots, m$ of player 2. Note that this implies in particular that $v_2 > 0$ since the strategies in the support of x_3 have positive probability.

For player 3, the strategy 0 yields the inequality $x_2(m) - \xi x_2(0) + \xi - 1 \leq v_3$. The strategies $j = 1, \ldots, m$ of player 3 yield respectively the inequalities $x_2(j-1) - \xi x_2(j) \leq v_3$. The sum of all the left hand sides of these inequalities for player 3 is $\sum_{j=0}^{m} x_2(j) - \xi \sum_{j=0}^{m} x_2(j) + \xi - 1 = 0$. Thus, $v_3 \geq 0$.

If $v_3 = 0$ then all the player 3 constraints must be satisfied with equality because otherwise the sum of the constraints would yield $0 < v_3$; hence $x_2(j-1) = \xi x_2(j)$, for $j = 1, \ldots, m$. Therefore, $x_2(j) = \xi^{m-j} x_2(m)$, for $j = 0, \ldots, m-1$. Since $\sum_{j=0}^{m} x_2(j) = 1$, it follows that $x_2(j) = \xi^{m-j} / \sum_{i=0}^{m} \xi^i$ for all $j = 0, \ldots, m$ (where, $\xi^0 = 1$ for all ξ including $\xi = 0$).

Suppose that $x_2(j) = \xi^{m-j} / \sum_{i=0}^{m} \xi^i$ does not hold for all j (and hence $v_3 > 0$). Since both, the left-hand sides $x_2(j)$ and the right-sides $\xi^{m-j} / \sum_{i=0}^{m} \xi^i$ sum to 1, there must be indexes for which we get strict inequalities in both directions. Hence there is an index j such that $x_2(j) > \xi^{m-j} / \sum_{i=0}^{m} \xi^i$ whereas $j' = (j-1) \mod (m+1)$ satisfies $x_2(j') \leq \xi^{m-j'} / \sum_{i=0}^{m} \xi^i$; if $j = 0$ the second inequality is $x_2(m) \leq 1 / \sum_{i=0}^{m} \xi^i$. The first inequality implies in particular that $x_2(j) > 0$. Furthermore, the combination of the two inequalities on $x_2(j)$ and $x_2(j')$ implies that the constraint corresponding to strategy j of player 3 is strict, and therefore $x_3(j) = 0$. This means that the player 2 constraint $x_3(j) \leq v_2$ corresponding to strategy j of player 2 is strict (because $v_2 > 0$), and therefore $x_2(j) = 0$, contradicting the fact that $x_2(j) > 0$. We conclude that $x_2(j) = \xi^{m-j} / \sum_{i=0}^{m} \xi^i$ for all $j = 0, \ldots, m$. Therefore, all the left-hand sides of the player 3 constraints are 0, which implies that $v_3 = 0$.

The expected payoff of pure strategy 1 of player 1 is:

$$u_1((1{:}1); x_{-1}) = \sum_{j=0}^{m} c_j x_2(j) = \sum_{j=0}^{m} c_j \xi^{m-j} / \sum_{i=0}^{m} \xi^i = f(\xi) / \sum_{i=0}^{m} \xi^i$$

and the expected payoff of pure strategy 0 is $u_1((1{:}0); x_{-1}) = 0$. The expected payoff of player 1 in the mixed profile x is $u_1(x) = (1-\xi) f(\xi) / \sum_{i=0}^{m} \xi^i$. Since x is a NE we must have $(1-\xi) f(\xi) \geq 0$ and $(1-\xi) f(\xi) \geq f(\xi)$. If $\xi \neq 0, 1$ these imply that $f(\xi) = 0$. If $\xi = 0$ then the first inequality becomes $f(0) \geq 0$ and since the hypothesis of the lemma is that $f(0) \leq 0$, it follows that $f(\xi) = f(0) = 0$. Similarly, if $\xi = 1$, the second inequality becomes $0 \geq f(1)$, and since the hypothesis of the lemma is that $f(1) \geq 0$, it follows again that $f(\xi) = f(1) = 0$. Thus, in all cases $f(\xi) = 0$, and since f has a unique root α in the interval $[0, 1]$ it follows that $\xi = \alpha$.

The mixed strategy of player 1 in the Nash equilibrium x is uniquely determined: $x_1(0) = \xi = \alpha$ and $x_1(1) = 1 - \alpha$. This implies that the mixed strategy of player 2 is also uniquely determined: $x_2(j) = \alpha^{m-j} / \sum_{i=0}^{m} \alpha^i$.

Suppose that $\alpha \neq 0$. Then $x_2(j) > 0$ for all j, and therefore all the constraints of player 2 are satisfied with equality. Thus, $x_3(j) = v_2$ for all $j = 0, \ldots, m$. Since $\sum_{j=0}^{m} x_3(j) = 1$, we have $x_3(j) = v_2 = 1/(m+1)$ for all $j = 0, \ldots, m$. Therefore, if $\alpha \neq 0$ then there is a unique Nash equilibrium. If $\alpha \neq 0, 1$ then the NE is fully mixed. ∎

In our reduction from PosSLP we will use Lemma 7 for a linear polynomial $f(x) = c_0 x + c_1$ to get a gadget for each operation, where in all cases the coefficients are such that $f(0) < 0$, $f(1) > 0$ and f has a unique root that lies in the open interval $(0, 1)$, hence the game G_f has a unique, fully mixed equilibrium. Since f has degree 1, all three players have two strategies each, labeled 0,1. For each gate $q_j, j = 1, \ldots, n$ of S we have a gadget G_{1j} which 'computes' the value of the

gate in the strategy $(1, j, 0)$ of the primary player 1 (i.e. the probability of the strategy in the unique NE is forced to $val(q_j)/n$), and then we have two gadgets G_{2j}, G_{3j} that copy the value to the corresponding strategies $(2, j, 0)$, $(3, j, 0)$ of the other two primary players 2,3. The gadget (subgame) G_{ij} corresponding to a primary player $i = 1, 2, 3$ and gate $q_j, j = 1, \ldots, n$ of S is a copy of the gadget G_f of the lemma, for some linear polynomial f, where players $i, 4, 5$ play the role of the players $1, 2, 3$ respectively in G_f, the strategies $(i, j, 0)$ and $(i, j, 1)$ in the j-th block of player i play the role of strategies 0,1 of the primary player 1 of G_f, and the strategies $(k, i, j, 0)$, $(k, i, j, 1)$ in the (i, j) block of the players $k = 4, 5$ play the role of the strategies 0,1 of the auxiliary players $2, 3$ of G_f. For the gadgets G_{1j}, the linear polynomial f is defined according to the operation of the gate. For the copying gadgets G_{2j}, G_{3j} the linear polynomial f does not depend on the operation of the gate.

Recall that the payoff function u_i of every unprimed player i consists of two terms μ_i and μ'_i, and we have already defined the second term in the paragraph before Lemma 6. The first term μ_i depends only on the strategies of the unprimed players. Let $s = (s_1, \ldots, s_5)$ be a pure strategy profile of the unprimed players. The payoff $\mu_i(s)$ depends on the block of s_i, which corresponds to one of the gadgets G_{kj}. We define in turn the payoff of all unprimed players. Except for primary player 1, the payoffs for the other players do not depend on the operations of the gates.

Payoff of auxiliary players 4,5. Let $s_4 = (4, k, j, b_2)$, where $k = 1, 2, 3$ is a primary player, $j = 1, \ldots, n$ is the index of a gate, and $b_2 = 0, 1$. Strategy s_4 is in block (k, j) of player 4 which corresponds to the gadget G_{kj}. If $s_k \notin Af(k, j)$ or $s_5 \notin Af(k, j)$ then $\mu_4(s) = 0$. Suppose that $s_k = (k, j, b_1)$ and $s_5 = (5, k, j, b_3)$ for some bits b_1, b_3. Then $\mu_4(s) = h_2(b_1, b_2, b_3) = \chi(b_2 = b_3)$ where h_2 is the payoff function of player 2 in the game G_f of Lemma 7; recall that the payoff of an auxiliary player does not depend on the polynomial.
The payoff function μ_5 of the auxiliary player 5 is defined similarly: Let $s_5 = (5, k, j, b_3)$. If at least one of s_k and s_4 is not in $Af(k, j)$ then the payoff is 0; if they both are in $Af(k, j)$ and $s_k = (k, j, b_1)$, $s_4 = (4, k, j, b_2)$, then $\mu_5(s) = h_3(b_1, b_2, b_3)$, where h_3 is the payoff function of player 3 in the game of Lemma 7.

Payoff of primary players 2,3. For a primary player $k = 2, 3$, let $s_k = (k, j, b_1)$, where $j = 1, \ldots, n$ is the index of a gate, and $b_1 = 0, 1$. Strategy s_k is in block j of player k which corresponds to the gadget G_{kj}. If $s_4 \notin Af(k, j)$ or $s_5 \notin Af(k, j)$ then $\mu_k(s) = 0$. Suppose that $s_4 = (4, k, j, b_2)$ and $s_5 = (5, k, j, b_3)$ for some bits b_2, b_3. If $b_1 = 0$ then $\mu_k(s) = 0$. If $b_1 = 1$ and $b_2 = 0$ then $\mu_k(s) = 1$. If $b_1 = 1$ and $b_2 = 1$ then $\mu_k(s) = -n$ if $s_1 = (1, j, 0)$, and $\mu_k(s) = 0$ otherwise. Note that if a mixed strategy profile gives probability p_j/n to the strategy $(1, j, 0)$ of player 1, then the payoff of primary player $k = 2, 3$ for the strategies in block j is the same as the payoff of the primary player in the gadget G_f for the polynomial $f(x) = x - p_j$.

Payoff of primary player 1. Let $s_1 = (1, j, b_1)$, where $j = 1, \ldots, n$ is the index of a gate q_j of S, and $b_1 = 0, 1$. Strategy s_1 is in block j of player 1 which corresponds to the gadget G_{1j}. If $s_4 \notin Af(1, j)$ or $s_5 \notin Af(1, j)$ then $\mu_k(s) = 0$. Suppose that $s_4 = (4, 1, j, b_2)$ and $s_5 = (5, 1, j, b_3)$ for some bits b_2, b_3. The gadget G_{1j} corresponds to a linear polynomial f that depends now on the gate q_j.

- Assignment (input) gate $q_1 := 1/2$. The polynomial is $f(x) = x - 1/2$. The payoff $\mu_1(s)$ is set equal to the payoff $h_1^f(b_1, b_2, b_3)$ of the primary player 1 in the gadget G^f of Lemma 7 for this polynomial f. That is, $\mu_1(s) = 1$ if $b_1 = 1$ and $b_2 = 0$, $\mu_1(s) = -1/2$ if $b_1 = 1$ and $b_2 = 1$, and $\mu_1(s) = 0$ otherwise.

- Addition gate $q_j := q_k + q_l$. The polynomial is $f(x) = x + c_1$, where we would like to set $c_1 = -(val(q_k) + val(q_l))$. However, of course we cannot (and do not want to) compute

explicitly the values of the lower level gates q_k, q_l. So, we use instead the values 'computed' in the probabilities of the corresponding strategies $(2,k,0)$ and $(2,l,0)$ of primary player 2, in the same way as we did for the copying gadgets in the definition of the payoffs for players 4 and 5. That is, $\mu_1(s) = 1$ if $b_1 = 1$ and $b_2 = 0$; $\mu_1(s) = -n$ if $b_1 = 1$, $b_2 = 1$, and ($s_2 = (2,k,0)$ or $s_2 = (2,l,0)$); and $\mu_1(s) = 0$ otherwise.

- Multiplication gate $q_j := q_k * q_l$. The polynomial is $f(x) = x + c_1$, where we would like to set $c_1 = -(val(q_k) * val(q_l))$, but we use instead the values computed in the probabilities of the corresponding strategies $(2,k,0)$ and $(3,l,0)$ of players 2 and 3. That is, we set $\mu_1(s) = 1$ if $b_1 = 1$ and $b_2 = 0$; $\mu_1(s) = -n^2$ if $b_1 = 1$, $b_2 = 1$, and $s_2 = (2,k,0)$, and $s_3 = (3,l,0)$; and $\mu_1(s) = 0$ otherwise.

- Division gate $q_j := q_k/q_l$. The polynomial is $f(x) = c_0 x + c_1$, where we would like to have $c_0 = val(q_l)$ and $c_1 = -val(q_k)$, and use in their place the values computed in the probabilities of the corresponding strategies $(2,l,0)$ and $(2,k,0)$ of player 2. Thus, we set $\mu_1(s) = n$ if $b_1 = 1$, and $b_2 = 0$, and $s_2 = (2,l,0)$; $\mu_1(s) = -n$ if $b_1 = 1$, $b_2 = 1$, and $s_2 = (2,k,0)$; and $\mu_1(s) = 0$ otherwise.

This concludes the definition of the game G. We will show now that G has the desired properties.

Lemma 8 *The game G defined above has a unique Nash equilibrium. The equilibrium is fully mixed. For each primary player $i = 1,2,3$ and each gate $q_j, j = 1,\ldots,n$ of S, the probability of the strategy $(i,j,0)$ in the NE is $val(q_j)/n$ and the probability of the strategy $(i,j,1)$ in the NE is $(1 - val(q_j))/n$.*

Proof. Let y be any Nash equilibrium of G. By Lemma 6 all the primed players $1',\ldots,5'$ have fully mixed strategies in y. Furthermore, for each unprimed player, the NE y assigns the same total probability to each block: probability $1/n$ for the primary players, and $1/3n$ for the auxiliary players. Thus, as far as the unprimed players are concerned, the only choice for y is how to distribute the probability in each block between the two strategies of the block. Clearly, the strategies of the primed players do not matter in this regard and can be ignored. We shall show that the restriction of y to the unprimed players is unique and fully mixed. It will follow then that the probabilities in y for the strategies of the primed players are also unique.

Specifically, we will use induction on the index $j = 1,\ldots,n$ of a gate q_j and then on the index $i = 1,2,3$ of a primary player to show that the probabilities in the NE y of the strategies in $Af(i,j)$ for each primary player $i = 1,2,3$ are uniquely defined, and furthermore, the probability $y(i,j,0) = val(q_j)/n$ and $y(i,j,1) = (1 - val(q_j))/n$.

In the basis case, $j = 1$, $i = 1$ and $q_1 := 1/2$. Then G_{11} is a copy of the gadget G_f for the polynomial $f(x) = x - 1/2$. Clearly, $f(0) < 0$, $f(1) > 0$ and there is a unique root $\alpha = 1/2$ in the interval $[0,1]$. Consider the restriction of y to the blocks of strategies of players $1,4,5$ in $Af(1,1)$. Multiply these probabilities by n for the primary player and by $3n$ for the auxiliary players to obtain a mixed strategy profile y' for G_f. We claim that y' is a NE for G_f. In proof, consider the primary player 1 of G_f and one of his two strategies $b = 0,1$ (the argument is similar for the auxiliary players). The expected payoff $h_1^f((1{:}b); y'_{-1})$ for 1 of strategy $b = 0,1$ with respect to the rest of the profile y' for the other players is equal to $(3n)^2$ times the expected payoff $\mu_1((1{:}(1,1,b)); y_{-1})$ in G of the strategy $(1,1,b)$ for player 1 with respect to the rest of the profile y for the other players. Thus, player 1 can improve his expected payoff in G_f by changing strategy unilaterally iff player 1 can improve his payoff in G by changing accordingly his probability allocation between the two strategies $(1,1,0),(1,1,1)$ in his first block. A similar argument applies to the auxiliary

players. Since y is a NE for G, it follows that y' is a NE for G_f. From Lemma 7 the NE y' is unique and fully mixed, hence the values of y for the strategies in $Af(1,1)$ are nonzero and uniquely determined. Since y' gives probability $1/2$ (the root of $f(x)$) to the strategy 0 of the primary player, it follows that y gives probability $1/2n$ to strategy $(i,j,0)$, and the remaining probability $1/2n$ to strategy $(i,j,1)$. Also, the components of the NE y corresponding to the strategies in $Af(1,1)$ of the auxiliary players are nonzero and unique.

Consider the copying gadgets G_{21} and G_{31}. More generally, the same argument applies to all the copying gadgets $G_{ij}, i = 2,3$ and $j = 1,\ldots,n$. Let $p_j = n \cdot y(1,j,0)$, and assume (inductively) that $y(1,j,0) = val(q_j)/n$ and $y(1,j,1) = (1-val(q_j))/n$. Thus, $p_j = val(q_j)$ is uniquely determined and is in the open interval $(0,1)$. We will show that $y(i,j,0) = val(q_j)/n$ and $y(i,j,1) = (1-val(q_j))/n$ for the other two primary players $i = 2,3$. Let $f(x) = x - val(q_j)$, and G_f the corresponding gadget of Lemma 7. From the definition of the payoff function μ_i, for $i = 2,3$, note that for every triple of strategies $s_i = (i,j,b_1)$, $s_4 = (4,i,j,b_2)$, $s_5 = (5,i,j,b_3)$, the expected payoff μ_i of player i, if players $i,4,5$ play pure strategies s_i, s_4, s_5 and the other players play according to y, is equal to the payoff $h_1^f(b_1,b_2,b_3)$ of the primary player 1 in G_f. This is of course true also for the auxiliary players 4,5 (their payoff does not even depend on f). Thus, the gadget G_{ij} acts exactly like a copy of the gadget G_f for the above polynomial $f(x) = x - val(q_j)$. The rest of the proof is the same as in the simple assignment case. We consider again the restriction of y to the blocks of strategies of players $i,4,5$ in $Af(i,j)$. Multiply these probabilities by n for the primary player i and by $3n$ for the auxiliary players to obtain a mixed strategy profile y' for G_f. We claim that y' is a NE for G_f. In proof, consider the primary player 1 of G_f. The expected payoff $h_1^f((1{:}b); y'_{-1})$ of player 1 for each of his strategies $b = 0,1$ is equal to $(3n)^2\mu_i((i{:}(i,j,b)); y_{-i})$. Since player i of G cannot improve his payoff in y, the primary player 1 of G_f cannot improve his payoff in y' either. Similarly with the auxiliary players of G_f. It follows that y' is a NE of G_f for $f(x) = x - val(q_j)$. Since all gates in S have value in the open interval $(0,1)$, it follows that $f(0) < 0 < f(1)$, there is a unique root $\alpha = val(q_j)$, and y' is uniquely determined and fully mixed. Therefore, the probabilities in y of the strategies of players $i,4,5$ in $Af(i,j)$ are also uniquely determined and positive, and $y(i,j,0) = val(q_j)/n$, $y(i,j,1) = (1 - val(q_j))/n$.

Consider the strategies $(1,j,b_1)$ of a block $j > 1$ of the primary player 1 of G. They are affected by the gadget G_{1j} which corresponds to a gate q_j of S. The gadget (more specifically, the payoffs of player 1) depends on the operation of the gate. The arguments are similar for the three types of operations. Consider an addition gate $q_j := q_k + q_l$. Inductively we know that $y(2,k,0) = val(q_k)/n$ and $y(2,l,0) = val(q_l)/n$. Given this, the gadget G_{1j} behaves like the gadget G_f for the polynomial $f(x) = x - (val(q_k) + val(q_l))$, in the following sense: for every triple of (pure) strategies $s_1 = (1,j,b_1)$, $s_4 = (4,1,j,b_2)$, $s_5 = (5,1,j,b_3)$ in $Af(1,j)$, the expected payoff μ_1 of player 1 if players $1,4,5$ play pure strategies s_1, s_4, s_5 and the other players play according to y, is equal to the payoff $h_1^f(b_1,b_2,b_3)$ of the primary player 1 in G_f. Similarly for the auxiliary players 4,5. Note that since all gates of S have value in $(0,1)$, it holds again that $f(0) < 0 < f(1)$ and f has a unique root $\alpha = val(q_k) + val(q_l)$. Thus, G_f has a unique, fully mixed equilibrium y'. The rest of the argument is the same as for the copying gadgets. The strategies of player 1 in $Af(1,j)$ have probability $y(1,j,0) = y'_1(0)/n = (val(q_k) + val(q_l))/n = val(q_j)/n$ and $y(1,j,1) = y'_1(1)/n = (1 - val(q_j))/n$, and the strategies of the auxiliary players in $Af(1,j)$ have value $3n$ times smaller than the corresponding strategies of the auxiliary players in G_f. Thus, they are all uniquely determined and positive.

For a multiplication gate $q_j := q_k * q_l$, the gadget G_{1j} behaves like the gadget G_f for the polynomial $f(x) = x - (val(q_k) * val(q_l))$. For a division gate $q_j := q_k/q_l$ the gadget G_{1j} behaves like the gadget G_f for the polynomial $f(x) = val(q_l)x - val(q_k)$. Note that in both cases, $f(0) < 0 <$

$f(1)$, and there is a unique root, which is equal to $val(q_j)$. Thus, G_f has a unique NE y' and it is fully mixed. By the same arguments as before, we conclude that all the strategies in $Af(1,j)$ are uniquely determined and have nonzero probability in y, and furthermore, $y(1,j,0) = y'_1(0)/n = val(q_j)/n$ and $y(1,j,1) = y'_1(1)/n = (1 - val(q_j))/n$.

It follows that the restriction of the NE y to the strategies of the unprimed players is unique and fully mixed. By Lemma 6, the NE y gives nonzero probability also to all the strategies of the primed players. Furthermore, it is easy to see that these probabilities are uniquely determined from the probabilities of the unprimed players. Consider for example player $1'$ and his n strategies $(1', j)$ $j = 1, \ldots n$ corresponding to the n blocks of player 1. The expected payoff $u_1((1{:}(1,j,0)); y_{-1})$ of strategy $(1,j,0)$ for player 1 with respect to the profile y for the rest of the players is $u_1((1{:}(1,j,0)); y_{-1}) = \mu_1((1{:}(1,j,0)); y_{-1})) + My(1', j)$, where the first term $\mu_1((1{:}(1,j,0)); y_{-1}))$ depends only on the restriction of y to the unprimed players, thus is uniquely determined. The expected payoffs $u_1((1{:}(1,j,0)); y_{-1})$ must be equal for all $j = 1, \ldots, n$ because the strategies $(1,j,0)$ are all in the support of the NE y. Thus, the NE y satisfies the $n-1$ equations $\mu_1((1{:}(1,j,0)); y_{-1})) + My(1', j) = \mu_1((1{:}(1,j,0)); y_{-1})) + My(1', 1)$ for $j = 2, \ldots, n$ along with the equation $\sum_{j=1}^{n} y(1', j) = n$. These equations determine uniquely the probabilities $y(1', j)$ from the restriction of y on the unprimed players. An analogous argument applies to all the other primed players. Therefore the Nash equilibrium is unique and fully mixed. ∎

We can transform now the game G to a 3-player game Γ. We use a reduction of Bubelis (Theorem 3 in [7]). The following proposition summarizes the relevant properties of the reduction.

Proposition 9 *[7] For any (finite) game G with d players we can construct a 3-player game G' such that the (pure) strategies of player 1 in G' correspond 1-1 to the (pure) strategies of all the players in G, and the following properties hold.*
1. For every Nash equilibrium y of G', if y_{ij} denotes the probability with which player 1 plays in y the strategy (i,j) corresponding to the jth strategy of player i in G, then the vector x defined by $x_{ij} = dy_{ij}$ is a Nash equilibrium in G.
2. Conversely, for every Nash equilibrium x of G, there is a Nash equilibrium y of G' such that player 1 plays in y each strategy (i,j) with probability x_{ij}/d. Furthermore, if x is fully mixed (all strategies are in the support), then there is a unique corresponding such Nash equilibrium y in G', and y is also fully mixed.
3. If the payoffs in G are rationals, then the payoffs in G' are also rational and the game G' can be constructed in polynomial time in the size of G.

Apply the transformation of the proposition to the 10-player game G that we constructed from the circuit S to construct a 3-player game. Let Γ be this game. Let (pure) strategies 1,2 of player 1 in Γ correspond to the strategies $(1, n-1, 0)$, $(1, n, 0)$ of the primary player 1 in G, where q_{n-1} and q_n are the two output gates $out1, out2$ of the circuit S. Then the following holds.

Lemma 10 *The 3-player game Γ defined above has a unique Nash equilibrium. The equilibrium is fully mixed. The probabilities in the NE of the strategies 1,2 of player 1 are proportional to the values of the output gates $out1$, $out2$ of the circuit S, i.e they are respectively $val(out1)/K$, $val(out2)/K$ for some K.*

Proof. The claim about uniqueness and full support of the Nash equilibrium of Γ follows immediately from Lemma 8 and Proposition 9. The probabilities of the strategies 1,2 of player 1 in the NE of Γ are respectively $val(out1)/K$, $val(out2)/K$, where $K = 10n$. ∎

We will use now the game Γ to show Theorem 4 for PosSLP.

Proof of Theorem 4 for PosSLP.
1. Add a fourth player to Γ to obtain a game H. The first three players have the same strategies and payoff as in Γ (thus their payoff is independent of Player 4). The fourth player has two strategies 1,2 and the following payoff function. Player 4 receives 1 if he plays the same strategy as player 1, and receives 0 otherwise. Consider any Nash equilibrium $y = (y_1, y_2, y_3, y_4)$ of H. Clearly the first three components (y_1, y_2, y_3) form an equilibrium of Γ since the payoff of the first three players does not depend on the actions of Player 4. The expected payoff of strategy $j = 1, 2$ of Player 4 with respect to this NE is equal to the probability of strategy $j = 1, 2$ of Player 1 in the NE, i.e. it is equal to $val(outj)/K$. Therefore, Player 4 plays strategy 1 with probability 1 if $val(out1) > val(out2)$, i.e. if $val(C) > 0$, and plays strategy 1 with probability 0 if $val(out1) < val(out2)$, i.e. if $val(C) \leq 0$. (Recall that the earlier transformation assured that $val(out1) \neq val(out2)$.) Since the game Γ has a unique Nash equilibrium, the same is true for H; the NE of H is fully mixed except for the fourth player that plays a pure strategy.

2. We could reduce H to a 3-player game using Proposition 9. This gives a constant approximability gap, but not close to 1. Namely, the gap resulting from this reduction is 1/4. Also the NE of the resulting 3-player game is not unique because the NE of H is not fully mixed (however the strategy of player 1 is the same in all NE). One way to show a gap close to 1, is by proving a weighted version of the Proposition, where the different players are assigned different weights, and then giving the bulk of the weight to player 4. This was our initial proof given in the conference version of the paper (see Lemma 5 in [21]). We will give here instead a more direct reduction from the game Γ, which also ensures the uniqueness of the NE.

Form a new 3-player game Γ' from Γ by adding a new strategy n to player 2 and two new strategies n_1, n_2 to player 3. The players have also the strategies of Γ which we call old strategies. Denote the payoff functions of Γ and Γ' by u_i and u'_i respectively. The payoff functions u'_i are defined as follows. Let $s = (s_1, s_2, s_3)$ be a pure strategy profile. Player 1 gets payoff $u'_1(s) = u_1(s)$ if both s_2, s_3 are old strategies, and $u'_1(s) = 0$ otherwise. The payoff of Player 2 is $u'_2(s) = L$ if both s_2 and s_3 are new strategies, it is $u'_2(s) = rL + u_2(s)$ if both s_2 and s_3 are old strategies, and it is 0 otherwise, where $L > 1$ is greater than the absolute value of all the payoffs of Γ, and r is an (arbitrary) large integer of polynomial bit-size, e.g., $r = 2^{poly(|\Gamma|)}$. The payoff of Player 3 is $u'_3(s) = -L + u_3(s)$ if both s_2 and s_3 are old strategies; it is $u'_3(s) = -L + \chi(s_3 = n_{s_1})$ if both s_2 and s_3 are new strategies; and it is 0 otherwise. Recall that $\chi(P(s))$ denotes the indicator function of a predicate $P(s)$, thus $\chi(s_3 = n_{s_1}) = 1$ if either $s_1 = 1$ and $s_3 = n_1$, or $s_1 = 2$ and $s_3 = n_2$.

Let $y = (y_1, y_2, y_3)$ be a Nash equilibrium of Γ', where y_1, y_2, y_3 are respectively the mixed strategies of players 1,2,3. For $j = 2, 3$, let $y_j(o), y_j(n)$ denote respectively the total probabilities of the old and new strategies of player j; for example, $y_3(n) = y_3(n_1) + y_3(n_2)$.

Claim 1 *1. The probabilities $y_2(o), y_2(n), y_3(o), y_3(n)$ are all positive.*
2. Furthermore, $r - 1 \leq y_3(n)/y_3(o) \leq r + 1$.

Proof. Players 2 and 3 play a weighted version of matching pennies with respect to the new/old type of their strategies. The first part of the claim is similar to the ordinary matching pennies. If $y_2(o) = 0$ (resp. if $y_2(n) = 0$) then Player 3 will put all the probability in the old (resp. new) strategies, and therefore player 2 will put also all the probability in the old (resp. new) strategies, contradiction. A similar argument shows that we cannot have $y_3(o) = 0$ or $y_3(n) = 0$. Thus, $y_2(o), y_2(n), y_3(o), y_3(n)$ are all positive.

For the second part, suppose first that $y_3(n) > (r+1)y_3(o)$. The expected payoff $u'_2((2{:}n); y_{-2})$ for player 2 of strategy n with respect to the profile y for the other players is $Ly_3(n) > L(r+1)y_3(o)$,

thus it is strictly greater than the expected payoff of all the old strategies of player 2; therefore $y_2(n) = 1$ and $y_2(o) = 0$, contradicting part 1.

Suppose that $y_3(n) < (r-1)y_3(o)$. Then the expected payoff $u_2'((2{:}n); y_{-2})$ is $Ly_3(n) < L(r-1)y_3(o)$, hence it is strictly smaller than the expected payoff of the old strategies; therefore $y_2(n) = 0$ contradicting again part 1. ∎

Consider the mixed strategy profile $z = (z_1, z_2, z_3)$ for the game Γ obtained from the NE y of Γ' by dividing the probabilities of the old strategies of player 2 by $y_2(o)$ and the old strategies of player 3 by $y_3(o)$. Since y is a NE for Γ', it follows from the definition of the payoffs that the profile z must also be a NE of Γ. Recall that Γ has a unique Nash equilibrium z and it is fully mixed. Thus $y_1 = z_1$ is also unique. The first two strategies of player 1 have probabilities $y_1(1), y_1(2)$ which satisfy $y_1(1) > y_1(2)$ iff $val(C) > 0$. Consider the strategies n_1, n_2 of player 3. From the definition of the payoffs of player 3 in Γ', it follows that if $val(C) > 0$, i.e. $y_1(1) > y_1(2)$, then $y_3(n_1) = y_3(n) \geq 1 - (1/r)$ and $y_3(n_2) = 0$, whereas if $val(C) > 0$, i.e. $y_1(1) < y_1(2)$, then $y_3(n_2) = y_3(n)$ and $y_3(n_1) = 0$. Therefore, if we can distinguish the case that strategy n_1 has probability $\geq 1 - (1/r)$ from the case that it has probability 0 then we can decide whether the value of the given circuit C is positive or not.

To show the uniqueness of the NE y, it suffices to show that the quantities $y_2(o), y_2(n), y_3(o), y_3(n)$ are uniquely determined, because then the uniqueness of the probabilities of the old strategies follows from the uniqueness of the NE of Γ, and for the two new strategies n_1, n_2 of Player 3, as we already argued, one of n_1, n_2 will get probability $y_3(n)$ and the other 0, depending on whether $val(C)$ is positive or negative. Let $u_2(z)$, $u_3(z)$ be the expected payoff of players 2,3 respectively at the NE z of Γ.

The expected payoff in Γ' for player 2 of all old strategies is $y_3(o)(u_2(z) + rL)$, and the expected payoff of strategy n is $Ly_3(n)$. These two quantities must be equal (because y is a NE and these strategies are in the support), thus $y_3(o)(u_2(z) + rL) = Ly_3(n)$, which together with the equality $y_3(o) + y_3(n) = 1$ determine uniquely $y_3(o)$ and $y_3(n)$.

The expected payoff for player 3 of all old strategies in the NE y is $y_2(o)(u_3(z) - L)$. If $val(C) > 0$ then strategy n_1 of player 3 is in the support, its expected payoff is $y_1(1) - Ly_2(n)$ and must be equal to $y_2(o)(u_3(z) - L)$. The equations $y_1(1) - Ly_2(n) = y_2(o)(u_3(z) - L)$ and $y_2(o) + y_2(n) = 1$ determine uniquely $y_2(o)$ and $y_2(n)$. Similarly, if $val(C) < 0$, then n_2 is in the support, its expected payoff is $y_1(2) - Ly_2(n) = y_2(o)(u_3(z) - L)$ and again combining with the equation $y_2(o) + y_2(n) = 1$ we can determine uniquely $y_2(o)$ and $y_2(n)$.

We conclude that Γ' has a unique NE, and the probability of strategy n_1 of player 3 is $\geq 1 - (1/r)$ or 0 according as $val(C) > 0$ or not. ∎

We will show now Theorem 4 for SQRT-SUM. It is known that SQRT-SUM Turing-reduces to PosSLP [1], which implies of course that SQRT-SUM Turing-reduces to the questions about Nash equilibria. However we can give direct many-one reductions using similar techniques to PosSLP.

Proof of Theorem 4 for SQRT-SUM.

We are given positive integers $d_1, \ldots, d_m$ and k and we want to test if $\sum_i \sqrt{d_i} \leq k$. Note that we can test if $\sum_i \sqrt{d_i} = k$ in polynomial time [6]. We do this test first on our input, so we can assume without loss of generality that we do not have equality.

We will give reductions using the same structure as for PosSLP. From the given instance of the SQRT-SUM problem, we will construct a 3-player game Γ that has a unique Nash equilibrium, the NE is fully mixed, and strategies 1 and 2 of player 1 have probabilities proportional to $\sum_i \sqrt{d_i}$ and k in the NE. Both parts of the Theorem can then be derived from the game Γ in exactly the same way as shown above in the proof for PosSLP.

Let R be an integer greater than $\sum_{i=1}^{m} d_i$ and k. We construct first a circuit S with two outputs $out1$, $out2$. All the gates have values in the open interval $(0,1)$ and the two output gates have values $val(out1) = (\sum_{i=1}^{m} \sqrt{d_i})/R$ and $val(out1) = k/R$. The circuit uses (scaled) square-root gates, addition, and input assignment gates. More precisely, for each $i = 1, \ldots, m$ the circuit S has a gate $q_i := \sqrt{d_i}/R$. Then it adds the gates q_i to compute the first output gate $out1$; we can do this either using a sequence of pairwise addition gates with fan-in 2, but we can also do it directly having an addition gate with fan-in m, $out1 = q_{m+1} := \sum_{i=1}^{m} q_i$. Finally we have the second output gate $out2 := k/R$ (an assignment gate). Clearly, all gates have values in $(0,1)$.

Now we construct from the circuit S a 10-player game G as we did in the PosSLP reduction (actually we do not need all the players in this case, but for simplicity we just follow the same method). The only thing we need to show is how to implement the gadget for the scaled square root operation $q_i := \sqrt{d_i}/R$. We use the construction of Lemma 7 for the polynomial $f(x) = R^2x^2 - d_i$. Clearly $f(0) < 0$, $f(1) > 0$, and the polynomial has a unique root in the interval $(0,1)$, namely $\sqrt{d_i}/R$. For addition we use the same gadget as before, and for the assignment $out2 := k/R$ we use the gadget for the linear polynomial $x - (k/R)$. As in the PosSLP case, the constructed game G has the properties of Lemma 8: it has a unique Nash equilibrium, the NE is fully mixed, and two distinguished strategies of player 1, say strategies 1,2, have probability $(\sum_{i=1}^{m} \sqrt{d_i})/nR$ and k/nR, where n is the number of gates of the circuit S (i.e. $n = m + 2$ if we use the fan-in m addition gate).

The rest of the steps are exactly the same as in the PosSLP case. We transform the game G to a 3-player game Γ, and then reduce Γ to a 4-player game H for part 1, and to a 3-player game Γ' for part 2. Both games H, Γ' have a unique Nash equilibrium. A particular strategy in game H (strategy 1 of player 4) is played with probability either 1 or 0 in the NE depending on whether $\sum_{i=1}^{m} \sqrt{d_i} > k$ or $\sum_{i=1}^{m} \sqrt{d_i} < k$. And similarly, a particular strategy (strategy n_1 of player 3) is played with probability at least $1 - (1/2^{poly})$ or 0 in the NE according as $\sum_{i=1}^{m} \sqrt{d_i} > k$ or $\sum_{i=1}^{m} \sqrt{d_i} < k$. ∎

The proof of the theorem has several interesting consequences, which we present in the rest of this section. A first corollary of the proof is that ϵ-Nash equilibria can be very far from the actual NE in a game: an ϵ-Nash equilibrium with ϵ doubly exponentially small can be distance 1 from any (even the unique) NE in a 4-player game, and distance almost 1 in a 3-player game.

Corollary 11 *There is a constant $c > 0$, such that for every n there is a 4-player game, Γ_n, of size $O(n)$ with a unique Nash equilibrium, and with a ϵ-NE, x', where $\epsilon = 1/2^{2^{\Omega(n^c)}}$, such that x' has l_∞-distance 1 from the NE. For 3-players, the same statement holds with distance 1 replaced by $1 - 2^{-poly(n)}$.*

Proof. Let S be the circuit that computes $t = 1/2^{2^m}$ by successive squaring of $1/2$, and has two output gates $out1$ and $out2$ with values t and $2t$. The circuit S has $m + 2$ gates. Apply the reduction of Theorem 4 to S to compute a 4-player game H. The game H has size polynomial in m, and has a unique Nash equilibrium y. The NE gives probability 0 to the strategy 1 of player 4 and probability 1 to strategy 2. The expected payoff of strategy 1 of player 4 differs from the expected payoff of strategy 2 by less that t. Thus, the mixed profile x which is the same as the NE y, except that strategy 1 of player 4 gets probability 1, is an ϵ-Nash equilibrium with $\epsilon < 1/2^{2^m}$.

Similarly, we can transform S to a 3-player game Γ', which has a unique Nash equilibrium y, and the NE gives probability 0 to strategy n_1 of player 3 and probability $1 - 2^{-poly}$ to strategy n_2. The mixed profile x that gives the probability $1 - 2^{-poly}$ to n_1 instead of n_2 is an ϵ-Nash equilibrium with $\epsilon < 1/2^{2^m}$. ∎

We note that this kind of gap between ϵ-NE and actual Nash equilibria with such a small ϵ can not happen in the 2-player case.

Proposition 12 *There is a polynomial p such that, for every 2-player game Γ and every rational $\delta > 0$, if $\epsilon \leq 1/2^{p(|\Gamma|+\mathtt{size}(\delta))}$ (i.e., with ϵ having size polynomial in the size of Γ and δ), then every ϵ-Nash equilibrium of Γ is within distance δ of an actual NE of Γ.*

Proof. Recall that every feasible Linear Program with rational coefficients has an optimal rational solution whose bit complexity is bounded by a polynomial in the size of the LP. Let $q(n)$ be a polynomial that upper bounds the size of an optimal solution of every LP with $10n$ variables and constraints, whose coefficients are rationals of size n. Let $p(n) = 4q(n)$.

Let Γ be a 2-player game with rational payoffs, and $\delta > 0$ a given rational. Let S_1, S_2 be the pure strategy sets of the two players, and assume without loss of generality that they are disjoint. Let u_1, u_2 be the payoff functions of the two players. Let $\epsilon \leq 1/2^{p(|\Gamma|+\mathtt{size}(\delta))}$, and let x' be any ϵ-Nash equilibrium of Γ. We view x' as a vector indexed by $S_1 \cup S_2$, where $S_1 \cap S_2 = \emptyset$. Let $i1 \in S_1$ be a best response (pure) strategy of player 1 with respect to strategy x' for player 2, i.e., $i1$ is a strategy in S_1 that has the maximum expected payoff $u_1((1{:}i1); x'_{-1})$. Let $i2 \in S_2$ be the analogous best response strategy for player 2. Let $I \subseteq S_1 \cup S_2$ be the set of strategies (of both players) with probability at most $\sqrt{\epsilon}$ in x'. Note that all pure strategies of player 1 that are not in I have expected payoffs with respect to the profile x' for player 2 that are within $\sqrt{\epsilon}$ of each other and of the best response strategy $i1$ of player 1, because otherwise if we transfer in x' the probability from the strategy in $S_1 - I$ with the minimum expected payoff to the best response strategy $i1$, the overall expected payoff of player 1 will increase by more than ϵ. Similarly, all strategies of player 2 that are not in I have expected payoffs with respect to the profile x' for player 2 that are within $\sqrt{\epsilon}$ of each other and of the best response $i2$.

Let y be the vector obtained by rounding every entry of x' to the nearest (integral) multiple of $\delta/2$. Consider the following Linear Program with a vector of variables x indexed by $S_1 \cup S_2$ and a scalar variable z. (The entries of the vector y in the LP are fixed rationals.)

Minimize z
Subject to:
$x_i - y_i \leq \delta/2$ for all $i \in S_1 \cup S_2$
$y_i - x_i \leq \delta/2$ for all $i \in S_1 \cup S_2$
$x_i \leq z$ for all $i \in I$
$\sum_{j \in S_2} u_1(i,j)x_j \leq \sum_{j \in S_2} u_1(i1,j)x_j$ for all $i \in S_1$
$\sum_{j \in S_2} u_1(i,j)x_j \geq \sum_{j \in S_2} u_1(i1,j)x_j - z$ for all $i \in S_1 - I$
$\sum_{j \in S_1} u_2(j,i)x_j \leq \sum_{j \in S_1} u_2(j,i2)x_j$ for all $i \in S_2$
$\sum_{j \in S_1} u_1(i,j)x_j \geq \sum_{j \in S_1} u_1(j,i2)x_j - z$ for all $i \in S_2 - I$
$\sum_{i \in S_1} x_i = 1$
$\sum_{i \in S_2} x_i = 1$
$x_i \geq 0$ for all $i \in S_1 \cup S_2$
$z \geq 0$

The vector x' and $z = \sqrt{\epsilon}$ satisfy all the constraints. Since $\sqrt{\epsilon} = 1/2^{2q(n)} < 1/2^{q(n)}$, it follows that an optimal solution has $z = 0$. Let x^*, $z = 0$ be an optimal solution. Then $x^*_i = 0$ for all $i \in I$. Furthermore, all strategies of player 1 in $S_1 - I$, which includes all the strategies in the support of x^*, have the same expected payoff $\sum_{j \in S_2} u_1(i1,j)x^*_j$; and all other strategies of S_1 that are not in the support do not have a higher payoff. Similarly with player 2. Therefore x^* is a Nash equilibrium of G. From the first two sets of inequalities, the vector x^* is within $\delta/2$ of the vector y, which in turn is within $\delta/2$ of x'. Therefore, $|x' - x^*| \leq \delta$. ∎

Another important difference between the 3-player and the 2-player case is that in the 2-player case the whole crux of the problem is in determining the support of a Nash equilibrium (this is why the problem is essentially purely combinatorial): If we are told what the support of a NE is then we can can easily compute a NE in polynomial time. This is not the case for 3-player games: Even if we know that there is a unique NE and that it is fully mixed (the support is the whole set of pure strategies of all the players), it is still not easy to compute or approximate a NE.

Corollary 13 *The SQRT-SUM and PosSLP problems are P-time (many-one) reducible to the following problem: Given a 3-player game that is guaranteed to have a unique Nash equilibrium, the NE is fully mixed, and the strategy 1 of player 1 is played in the NE either with probability $< \epsilon$ or with probability $> 1 - \epsilon$ where $\epsilon = 1/2^{poly}$, distinguish between the two cases.*

Proof. We show the proof for PosSLP; the proof for SQRT-SUM is similar. Given a circuit C over basis $\{+, -, *\}$ with inputs 0,1, we construct first as before a circuit D over $\{+, *\}$ with input 1 and two output gates, and from this a circuit S over $\{+, *, /\}$ with input 1/2 and with two output gates $out1, out2$ with the properties that all gates have values in the open interval $(0, 1)$, the output gates have unequal values and $val(C) > 0$ iff $val(out1) > val(out2)$ (see Lemma 5). Recall that S computes first in some gate h_d a tiny quantity $t = 1/2^{2^d}$, where d is the depth of D. The values of $out1$ and $out2$ are equal to t times the values of the output gates of D, thus they are integer multiples of t, thus in particular $val(out1), val(out2)$, and $|val(out1) - val(out2)|$ are all at least t.

Extend S to a circuit S' that has two new output gates $out1', out2'$ such that, if $val(out1) > val(out2)$ then $val(out1')$ is almost 1 (but < 1) and $val(out2')$ is almost 0 (but > 0), and symmetrically, if $val(out1) < val(out2)$ then $val(out1')$ is almost 0 and $val(out2')$ is almost 1. Specifically, $out1'$ computes $[out1 + t^2 - \min(out1, out2)]/[|out1 - out2| + 2t^2]$, and similarly $out2'$ computes $[out2 + t^2 - \min(out1, out2)]/[|out1 - out2| + 2t^2]$. If $val(out1) > val(out2)$ then $val(out1') = [|out1 - out2| + t^2])]/[|out1 - out2| + 2t^2] > 1 - t$, and $val(out2') = t^2/[|out1 - out2| + 2t^2] < t$. If $val(out1) < val(out2)$ then $val(out1') < t$ and $val(out2') > 1 - t$. We can compute the desired values using gates for the absolute value and min operations, or derive these using a square-root gate. Since we already showed a gadget for the square root operation that has the required uniqueness and full support properties, we will use a square root gate. Note that if a, b are two numbers, then $|a-b| = \sqrt{a^2 + b^2 - 2ab}$ and $min(a, b) = (a+b)/2-|a-b|/2$. We add gates to S that compute the following quantities: $w_1 := (out1)^2$, $w_2 := (out2)^2$, $w_3 := out1 * out2$, $w_4 := w_1 + w_2 - 2w_3$, $w_5 := \sqrt{w_1}$ (thus $w_5 = |out1-out2|$), $w_6 := (out1/2)+(out2/2)$, $w_7 := w_6-(w_5/2)$ (thus $w_7 = \min(out1, out2)$), $w_8 := (h_d)^2$, $w_9 := w_5+2w_8$ (thus $w_9 = [|out1-out2|+2t^2]$), $w_{10} := out1+w_8-w_7$, $out1' := w_{10}/w_9$, $w_{11} := out2+w_8-w_7$, $out2' := w_{11}/w_9$. Some of the right-hand sides have more than two operands (for example, the equation for w_4, w_{10} etc.); we could break them down further into pairwise operations if we wished, but it is not necessary. Observe that all the gates have values in the interval $(0, 1)$. For example, for $w_9 = [|out1 - out2| + 2t^2]$ this follows from the fact that $1 > out1, out2 \geq t > 2t^2$.

Now apply the transformation of Theorem 4 to the circuit S' to obtain first a 10-player game G, and from this a 3-player game Γ, such that Γ has a unique Nash equilibrium y, the NE is fully mixed, and the probabilities in the NE of strategies 1,2 of player 1 are proportional to the values of $out1'$, $out2'$, i.e. $y_1(1) = val(out1')/K$ and $y_2(2) = val(out2')/K$ for some K (cf. Lemma 10).

From Γ construct a 3-player $\hat{\Gamma}$ as follows. Player 1 has the same strategies as in Γ. Players 2 and 3 have the strategies of Γ, which we call old strategies and in addition they each have two new strategies n_1, n_2. Denote by u_i the payoff functions in Γ, and by $\hat{u}_i$ in $\hat{\Gamma}$. Let L be a number larger than the absolute value of all the payoffs of Γ and 1, and let r be an arbitrary large integer of polynomial bit-size, e.g. $r = 2^{poly(n)}$. The payoff functions $\hat{u}_i$ are as follows. Let $s = (s_1, s_2, s_3)$ be a pure strategy profile. Player 1 gets payoff $\hat{u}_1(s) = u_1(s)$ if both s_2, s_3 are old strategies, and

$\hat{u}_1(s) = 0$ otherwise. The payoff $\hat{u}_2(s)$ of Player 2 is defined as follows. If both s_2 and s_3 are old strategies, then $\hat{u}_2(s) = rL + u_2(s)$. If both s_2 and s_3 are new strategies then $\hat{u}_2(s)$ is the sum of two terms: the first term is L, and the second term is 1 iff $s_2 = n_{s_1}$ and $s_3 \neq s_2$, and it is 0 otherwise; i.e. the second term is 1 if either $s_1 = 1, s_2 = n_1, s_3 = n_2$ or $s_1 = 2, s_2 = n_2, s_3 = n_1$. If one of s_2, s_3 is new and the other is old then $\hat{u}_2(s) = 0$. The payoff $\hat{u}_3(s)$ of Player 3 is defined as follows. If both s_2 and s_3 are old strategies, then $\hat{u}_3(s) = -L + u_3(s)$. If both s_2 and s_3 are new strategies then $\hat{u}_3(s) = -L + \chi(s_2 = s_3)$; i.e., the second term is 1 if $s_2 = s_3$, and it is 0 otherwise. If one of s_2, s_3 is new and the other is old, then $\hat{u}_3(s) = 0$.

Note that the above game $\hat{\Gamma}$ is very similar to the game Γ' that we constructed in the proof of part 2 of Theorem 4. As in that game, players 2 and 3 play a weighted matching pennies game between the new and old strategies. The difference here is that both players 2 and 3 have two new strategies, and the subgame induced by these strategies is another weighted matching pennies game whose weight is determined by the probabilities of the strategies 1 and 2 of player 1.

Let $y = (y_1, y_2, y_3)$ be a Nash equilibrium of $\hat{\Gamma}$. For $j = 2, 3$, let $y_j(o), y_j(n)$ denote respectively the total probabilities of the old and new strategies of player j; for example, $y_3(n) = y_3(n_1) + y_3(n_2)$. These quantities satisfy the same properties of Claim 1 in the proof of Theorem 4: The probabilities $y_2(o), y_2(n), y_3(o), y_3(n)$ are all positive. Furthermore, $r - 1 \leq y_3(n)/y_3(o) \leq r + 1$. Let $z = (z_1, z_2, z_3)$ be the mixed strategy profile for the game Γ obtained from the NE y of $\hat{\Gamma}$ by dividing the probabilities of the old strategies of player 2 by $y_2(o)$ and the old strategies of player 3 by $y_3(o)$. As in the case of the game Γ' in the proof of Theorem 4, the profile z must be the (unique) Nash equilibrium of Γ. Thus, $y_1 = z_1$ is unique, and $y_1(1) = val(out1')/K$, $y_1(2) = val(out2')/K$.

The subgame induced by the strategies n_1, n_2 of players 2 and 3 is a weighted matching pennies game. As in Claim 1 both strategies of both players must have nonzero probability in y (since their sums $y_2(n)$ and $y_3(n)$ are both nonzero). Therefore, the NE y is fully mixed. Moreover, the ratio of the probabilities $y_3(n_1), y_3(n_2)$ is equal to the ratio of the probabilities $y_1(1), y_1(2)$. To see this, note that the expected payoff of strategy n_1 of player 2 with respect to the profile y for the other players is $\hat{u}_2((2{:}n_1); y_{-2}) = Ly_3(n) + y_1(1)y_3(n_2)$, and the expected payoff of strategy n_2 is $\hat{u}_2((2{:}n_2); y_{-2}) = Ly_3(n) + y_1(2)y_3(n_1)$. Since both strategies are in the support, the two quantities must be equal. Thus $y_1(1)y_3(n_2) = y_1(2)y_3(n_1)$, hence $y_3(n_1)/y_3(n_2) = y_1(1)/y_1(2)$.

If the given circuit C has positive value $val(C) > 0$, then $y_1(1)/y_1(2) = val(out1')/val(out2') > (1-t)/t$, thus $y_3(n_1) > (1-t)y_3(n) \geq (1-t)(1-(1/r)) = 1 - (1/2^{poly})$. If the given circuit C has value $val(C) \leq 0$, then $y_1(1)/y_1(2) = val(out1')/val(out2') < t/(1-t)$, thus $y_3(n_1) < ty_3(n) << (1/2^{poly})$.

We already remarked that the NE y has full support. For the uniqueness, it is not hard to see that $y_2(o), y_2(n), y_3(o), y_3(n)$ are uniquely determined along the same lines as in the proof of Theorem 4. This implies then the uniqueness of the probabilities of all the strategies in the NE. The uniqueness of the probabilities of the old strategies follows from the uniqueness of the NE of Γ. For the new strategies n_1, n_2 of player 3, observe that since $y_3(n_1)/y_3(n_2) = y_1(1)/y_1(2)$, we must have $y_3(n_1) = y_3(n)y_1(1)/(y_1(1)+y_1(2))$ and $y_3(n_2) = y_3(n)y_1(2)/(y_1(1)+y_1(2))$, and thus $y_3(n_1), y_3(n_2)$ are uniquely determined from $y_3(n)$ and $y_1 = z_1$. Regarding the new strategies of player 2, we can show that $y_2(n_1) = y_2(n_2) = y_2(n)/2$: To see this, note that the expected payoffs of strategies n_1, n_2 of player 3 with respect to the profile y for the other players must be equal (because they are in the support), thus $-Ly_2(n) + y_2(n_1) = -Ly_2(n) + y_2(n_2)$, hence $y_2(n_1) = y_2(n_2)$. Thus, the game $\hat{\Gamma}$ has a unique, fully mixed Nash equilibrium. ∎

Another corollary of the proof of Theorem 4 concerns the difficulty of computing specified bits of a NE. One of the implications of the theorem (and the above Corollary) is that computing the first bit of a specified strategy probability x_1 in a (any) NE is at least as hard as SQRT-SUM and

PosSLP. In [1] it is shown that computing a specified bit of the integer computed by an arithmetic circuit is #P-hard, when the index of the desired bit is given in binary. Using our reductions from PosSLP, we can deduce a similar result for NEs. The key property is that we can ensure that certain probabilities in the game are binary shifts of the numbers computed by the arithmetic circuit.

Corollary 14 *Given a 3-player game that has a unique Nash equilibrium and given an index i in binary, it is #P-hard to compute the i-th bit of the probability of the first strategy of player 1 in the NE.*

Proof. We reduce from the problem of computing a desired bit i (specified in binary) of the integer computed by an arithmetic circuit over basis $\{+, *\}$ with input 0,1 [1]. Let D be such a circuit. We follow the same transformation steps of Theorem 4. Construct from D another circuit S such that all gates have values in $(0, 1)$ with an output gate *out* whose value is $val(out) = t \cdot val(D)$, where $t = 1/2^{2^d}$, and d is the depth of D. (We don't need two outputs here.) Add dummy gates to S so that the number n of gates is a power of 2, $n = 2^m$ for some m. Next transform the circuit S to a game G as before, but now add 6 more dummy players that have one strategy each, so that G has 16 players. Finally transform G to a 3-player game Γ as before. The game Γ has a unique Nash equilibrium y. The probability of strategy 1 of player 1 in the NE is $y_1(1) = val(out)/16n = val(D)/2^{2^d+m+4}$. Thus $y_1(1)$ is a binary shift of $val(D)$, and the i-th bit of $val(D)$ is equal to the bit of $y_1(1)$ with index $i + 2^d + m + 4$. Clearly we can specify this index in binary with polynomially many bits. ■

Finally we note that similar results follow for market equilibria. We state and show below the analogue of Corollary 13, concerning the computational hardness of estimating prices in a market with a unique equilibrium. Recall that prices are normalized to be in the unit simplex, i.e. the prices of all the commodities are nonnegative and their sum is equal to 1.

Corollary 15 *The SQRT-SUM and PosSLP problems are P-time (many-one) reducible to the following problem: Given an exchange economy where the aggregate excess demand functions for the commodities are given by explicit algebraic formulas over $\{+, -, *, /, \max\}$, with the property that the market has a unique price equilibrium, all prices are positive in the equilibrium, and commodity 1 has price either $< \epsilon$ or $> 1 - \epsilon$ where $\epsilon = 1/2^{poly}$, distinguish between the two cases.*

Proof. We reduce from the problem of Corollary 13. We are given a 3-player game G that is guaranteed to have a unique Nash equilibrium, the NE is fully mixed, the strategy 1 of player 1 is played in the NE either with probability $< \epsilon'$ or $> 1 - \epsilon'$, where $\epsilon' = 1/2^{poly}$, and we wish to distinguish between the two cases. Let n_i be the number of strategies of player i, and $n = n_1 + n_2 + n_3$. We have an economy with n commodities corresponding to the n strategies of the players. We let (i, j) denote the commodity corresponding to the j-th strategy of player i, where (1,1) is the commodity 1 in the statement of the corollary. We will define a suitable Brouwer function f from Δ_n to Δ_n, which has a unique fixed point p^* that has the desired properties, and then we will define the excess demand functions from f using Uzawa's formula.

Let $\epsilon = 2\epsilon'$, $a_1 = 1 - \epsilon'$, $a_2 = a_3 = \epsilon'/2$. Every mixed strategy profile $y = (y_{i,j} | i = 1, 2, 3; j = 1, \ldots, n_i)$ of G can be mapped 1-to-1 to the point $p = \phi(y)$ of Δ_n where $p_{i,j} = \phi(y)_{i,j} = a_i y_{i,j}$; note that the point $p = \phi(y)$ lies in the subset $S = \{p \in \Delta_n | \sum_{j=1}^{n_i} p_{i,j} = a_i; i = 1, 2, 3\}$. Every point p in Δ_n can be mapped to a mixed strategy profile $y = \pi(p)$ of G using the formula $\pi(p)_{i,j} = \frac{p_{i,j} + \frac{1}{n_i} \max(a_i - \sum_{j=1}^{n_i} p_{i,j}, 0)}{\max(a_i, \sum_{j=1}^{n_i} p_{i,j})}$, for all (i, j). It is straightforward to verify that $\sum_{j=1}^{n_i} \pi(p)_{i,j} = 1$ for all $i = 1, 2, 3$. Furthermore, if $p \in S$, then $\pi(p)_{i,j} = \frac{p_{i,j}}{a_i}$ for all i, j; that is, $\phi(\pi(p)) = p$. Let F_G

be Nash's function for the game G. Define the function $f : \Delta_n \to \Delta_n$ by the formula $f(p) = \phi(F_G(\pi(p)))$. That is, a point $p \in \Delta_n$ is first mapped to the mixed strategy profile $y = \pi(p)$ of G, then y is mapped to the new strategy profile $y' = F_G(y)$, and finally y' is mapped back to a point $p' = \phi(p)$ in $S \subset \Delta_n$.

We claim that f has a unique fixed point, specifically the point $p^* = \phi(y^*)$, where y^* is the unique Nash equilibrium of G. Clearly, $f(p^*) = \phi(F_G(\pi(p^*))) = \phi(F_G(y^*)) = \phi(y^*) = p^*$. Conversely, any fixed point p' of f must belong to S because the range of f is contained in S; if $y' = \pi(p')$ is not a NE of G, then the Nash function maps it to a different profile $y'' = F_G(y') \neq y'$, and then ϕ maps it back to a different point $p'' \neq p'$, because ϕ is 1-to-1. Thus, the only fixed point of f is $p^* = \phi(y^*)$.

Using Uzawa's formula [60], we can define a total market excess demand function $g : \Delta_n \to R^n$ from the above function f, where $g(p) = f(p) - \frac{\langle p, f(p)\rangle}{\langle p,p\rangle} p$. Then there is exactly one price equilibrium, namely the unique fixed point $p^* = \phi(y^*)$ of f. The price p_1^* of commodity 1 (i.e. (1,1)) in the price equilibrium is equal to $a_1 y_{1,1}^* = (1 - \epsilon')y_{1,1}^*$. If strategy 1 of player 1 has probability $y_{1,1}^* < \epsilon'$ in y^*, then the corresponding commodity 1 has price $p_1^* < \epsilon' < \epsilon$, and if the strategy probability is $y_{1,1}^* > 1 - \epsilon'$ then the commodity 1 price is $p_1^* > (1 - \epsilon')^2 > 1 - 2\epsilon' = 1 - \epsilon$. ∎

4 Nash Equilibria and the class FIXP

We define a class of fixed point problems, **FIXP**, for which the Nash equilibrium problem (for 3 or more players) will be shown to be complete. The class of fixed point problems induces a corresponding class for each of the associated types of questions: e.g., a class of decision problems, a class of (strong) approximation problems, etc. As usual in the definition of classes (cf. PLS, PPAD, MAXSNP, etc.), it is convenient to define a syntactic version and then close it under (polynomial-time) reductions. Recall that in the framework of search problems cast as fixed points, each instance I of a search problem is associated with a continuous function F_I from a compact convex domain D_I to itself, such that $Sol(I) = Fix(F_I)$. The basic characteristics of the class FIXP are that (i) the domain D_I is a convex polytope described by a set of linear inequalities with rational coefficients that can be computed from I in polynomial time, and (ii) the function F_I is represented by an *algebraic circuit* (or *straight line program*) C_I over the basis $\{+, -, *, /, \max, \min, \sqrt[k]{}\}$ using rational constants, that computes the function over the domain, and there is a polynomial time algorithm that computes the circuit from I.[2] Note that the circuit operates on real numbers, but the algorithms we study do not do any computation on reals; the circuit is a finite representation of the function, just like a formula, but more succinct of course. The underlying model of computation for us is still the standard discrete Turing machine.

The circuit C_I is a sequence of gates $g_1, \ldots, g_r$, where for $i = 1, \ldots, n$, $g_i := x_i$ is an input variable, for $i = n+1, \ldots, n+m$, $g_i := c_i \in \mathbb{Q}$ is a rational constant (encoded in the standard way, with numerator and denominator given in binary), and for $i > n + m$, we have gates $g_i := g_j \circ g_k$, with $j, k < i$, where the binary operator is $\circ \in \{+, -, *, /, \max, \min\}$ (infix max and min have the obvious meaning), or gate $g_i = \sqrt[k]{g_j}$ for some $j < i$, and some integer $k > 1$ given in unary. The last n gates are the output gates. The circuit C_I represents the function F_I. Thus, it is required to have the property that if we input any vector $x \in D_I$ into C_I then all the gates are well defined and the circuit outputs a vector $C_I(x) = F_I(x)$ in D_I. In particular, this means that for inputs $x \in D_I$, there is no division by 0 in the circuit (i.e. the denominators of all division gates are nonzero), and

[2] Including the gates $\sqrt[k]{}$ allows one to express readily a richer class of Brouwer functions (and is also convenient in the proofs). It will follow from the completeness of the Nash equilibrium problem for FIXP that omitting the root gates does not reduce the expressive power of FIXP.

the operands of all the root gates $\sqrt[k]{}$ with even k are nonnegative; a root gate outputs the unique nonnegative root of its operand if k is even, and the unique (real) root if k is odd.

We can define other operations from the basis. Since we have multiplication, we can easily build any integer power y^m (even with m written in binary), and thus also any rational power $y^{m/k} = (\sqrt[k]{y})^m$ (with the denominator k given in unary, and the operand has to be nonnegative for even k for this to be defined). With max and arithmetic we can define other piecewise linear functions, for example the absolute value $|y| = \max(y, -y)$. The absolute value can be obtained also as $|y| = \sqrt{y^2}$.

On the other hand, some of the operations are obviously redundant (can be defined in terms of others) and could have been omitted, but we included them explicitly for convenience. For example, since we have subtraction, min can be obtained from max and vice-versa. In fact, since we allow any rational constants, subtraction itself is not even needed. Also, we could have just included the constant 1 (and -1 if we omitted subtraction) and built the other rationals using operations. Furthermore, since we have square root, the min and max operations can be built from the others and are not needed: $\max(a, b) = (a+b)/2+|a-b|/2 = (a+b)/2+\sqrt{(a-b)^2}/2$; similarly, $\min(a, b) = (a + b)/2 - |a - b|/2 = (a + b)/2 - \sqrt{(a-b)^2}/2$.

We close the class by reductions. Recall that a reduction from problem A to problem B consists of two functions, a polynomial-time function f that maps instances I of A to instances $f(I)$ of B, and a second function g that maps solutions y of the instance $f(I)$ of B to solutions x of the instance I of A. In the case of reductions between discrete problems, where solutions are finite (this includes problems with rational-valued solutions), the second function g is any polynomial-time computable function. For problems with real-valued solutions, as mentioned in Section 2, we restrict the function g to be a separable linear transformation with rational coefficients that are computable in polynomial time from I. For example, a special case of such a reduction is when the formulation of a search problem as a fixed point problem involves the addition of more auxiliary variables besides the ones we want to compute in the search problem; in this case the function g is the projection on the coordinates of interest in problem A. For instance, in a game we may be only interested in the payoff of one player in a Nash Equilibrium or the total payoff; the formulation of the problem as a fixed point problem involves the addition of variables for the probabilities of the players' strategies.

In our definition of the class, the domains of the functions are polyhedral. However, we can include in FIXP problems with common nonlinear convex domains such as balls and ellipsoids, by modifying (enlarging) the domains and defining suitable embeddings of the domains via algebraic functions that preserve the fixed points. In general, reductions can be used to embed different domains into each other by an algebraic function in a continuous fixed point-preserving manner . For example, we show below that FIXP does not change if we restrict the domain of the function for every instance to be a unit cube or unit simplex.

Lemma 16 *The class FIXP does not change if the requirement on the domain for the instances of the problem is changed in either of the following ways.*

1. *The domain for an instance can be a ball with given rational center and diameter, or more generally an ellipsoid with given rational center and matrix, computed from the instance in polynomial time.*

2. *The domain for every instance is restricted to be a unit cube. The same holds if the domain for every instance is restricted to be a unit simplex.*

Proof. 1. Suppose that the domain for instance I is a ball $S(c, r)$ with (rational) center c and radius r in $\mathbb{R}^d$, i.e., $D_I = \{x \in \mathbb{R}^d | \sum_{i=1}^d (x_i - c_i)^2 \leq r^2\}$, and let C_I be the circuit for the function

F_I. Let $B[l,u]$ be a big enough box that contains $S(c,r)$, for example take $l = c - r \cdot 1$, $u = c + r \cdot 1$ where 1 is the all-1 vector. We change the domain to $D'_I = B[l,u]$ and modify the circuit to another circuit C'_I in such a way that the fixed points stay the same. The circuit C'_I takes as input a point $x \in B[l,u]$, computes $\theta = \max(1, \frac{\sqrt{\sum_{i=1}^{d}(x_i - c_i)^2}}{r})$, and then computes the point $y = c + (x-c)/\theta$. Note that $y \in S(c,r)$, and if $x \in S(c,r)$ then $y = x$. Then we feed y into the circuit C_I, and output $C_I(y)$. Clearly, every fixed point of C_I is also a fixed point of C'_I, and conversely every fixed point x of C'_I must be in the ball $S(c,r)$ and is a fixed point of C_I.

Suppose that D_I is an ellipsoid $E(c,A) = \{x \in \mathbb{R}^d | (x-c)^T A(x-c) \leq 1\}$, where c is a rational point and A a rational symmetric positive definite matrix, constructed from I in polynomial time. Recall that every symmetric positive definite matrix, A, has a Cholesky factorization $A = LL^T$, where L is lower triangular, non-singular, and has positive diagonal entries. The matrix L can be 'computed' from A using a polynomial number of algebraic operations, including square roots (see, e.g., [58], page 183). Note that $E(c,A) = \{x \in \mathbb{R}^d | (L^T(x-c))^T (L^T(x-c)) \leq 1\}$. We again change the domain to a big enough box $D'_I = B[l,u]$ that contains D_I. The new circuit C'_I takes as input a point $x \in B[l,u]$ and works as follows. First it computes the Cholesky factor matrix L, and its inverse L^{-1}. Then it computes the point $z = L^T(x-c)$, the scalar $\theta = \max(1, \sqrt{\sum_{i=1}^{d} z_i^2})$, the point $w = z/\theta$, and then the point $y = (L^{-1})^T w + c$. Finally it outputs $C_I(y)$. Note that y is in $D_I = E(c,A)$, because $(y-c)^T A(y-c) = w^T L^{-1} LL^T (L^{-1})^T w = w^T w \leq 1$; furthermore, if $x \in D_I$, then $\theta = 1$, $w = z$, and $y = x$. Therefore, the fixed points of C'_I are the same as the fixed points of C_I.

2. Let Π be a problem in FIXP. We reduce it to another problem Φ in FIXP that has the same instances, and whose domain for every instance is a unit cube. We use the same domain-embedding mapping as in the proof of Lemma 1. Each instance I of Π has an associated polyhedral domain D_I, and a function F_I represented by a polynomially computable circuit C_I that maps D_I to itself. Let $d = d_I$ be the number of variables of F_I. As shown in Lemma 1 we can define another function G_I that maps the unit cube $[0,1]^d$ to itself, and define a linear separable function g on $\mathbb{R}^d$ such that there is a 1-1 correspondence between the fixed points of F_I and G_I, specifically, y is a fixed point of G_I iff $x = g(y)$ is a fixed point of F_I.

The function g maps the i-th coordinate of a point y to the i-th coordinate of a point x by the expression $x_i = g_i(y) = (u_i - l_i) \cdot y_i + l_i$, where $u_i > l_i$ are respectively an upper and lower bound on the i-th coordinate of all the points in D_I. The reverse mapping g^{-1} maps x_i to $y_i = (x_i - l_i)/(u_i - l_i)$. The function G_I is defined as $g^{-1} \circ F_I \circ \pi \circ g$, where π maps the box $B[l,u]$ to D_I. We follow the definitions of these functions to construct from the instance I a circuit C'_I for the function G_I.

First we compute in polynomial time from the instance I a set of linear equations $\{ax = b_i | i = 1, \ldots, k\}$ and a set of linear inequalities $\{ax \leq b_i | i = k+1, \ldots, m\}$ that describe the domain D_I, and compute a point c in the relative interior of D_I and a matrix Q as in the proof of Lemma 1. We also compute the circuit C_I for the function F_I. The circuit C'_I for the function G_I is constructed as follows. The circuit C'_I takes as input a vector (point) y in the unit cube $[0,1]^d$, and applies first the linear transformation g to compute a point x, which will be in the box $B[l,u]$. The next part of the circuit computes $\rho(x) = c + Q(Q^T Q)^{-1} Q^T (x-c)$. Then it computes $\theta_i(x) = \max((a_i \rho(x) - a_i c)/(b_i - a_i c), 1)$ for $i = k+1, \ldots, m$, and then $\theta(x) = \max_i \theta_i(x)$. Note that $\theta(x)$ is guaranteed to be nonzero, in fact at least 1. Then the circuit computes $\pi(x) = c + (\rho(x) - c)/\theta(x)$. The vector $\pi(x)$ is fed into the inputs of the circuit C_I. Finally the circuit C'_I applies the (linear) transformation g^{-1} to the output x' of C_I to produce the output $y' = g^{-1}$ of C'_I.

The proof for the unit simplex is similar. ∎

Corresponding to the class FIXP of search problems (with real solutions) there is a class of

Decision problems, a class of Approximation problems and a class of Partial computation problems. To emphasize the distinction between them, we can attach a subscript to FIXP, denoting these classes of associated problems as $FIXP_d$, $FIXP_a$, $FIXP_{pc}$, respectively. Note that these are classes of discrete search problems (hence their complexity can be studied in the standard Turing machine model), as opposed to FIXP which is a class of, in general, real-valued search problems (whose complexity can be studied in a real computation model, e.g. [5]). We can show that all the discrete problems are in PSPACE, by appealing to PSPACE decision procedures for the *existential theory of reals* (ETR) [8, 51]. The decision problem for the ETR is the following: decide whether there exists a real vector of values x which satisfies a given boolean combination of multi-variate polynomial inequalities of the form $P(x) \geq 0$, where the polynomials $P(x)$ have rational coefficients [51].

Proposition 17 *The Partial Computation, Decision, (Strong) Approximation and Existence versions of all problems in FIXP can be solved in PSPACE.*

Proof. Let Π be a problem in FIXP. Recall the *existence problem*: Given an instance I and a tuple of rational (upper and/or lower) bounds for a subset of the components of a solution, does there exist a solution $x \in Sol(I)$ that satisfies the given bounds? As we noted, this is stronger than the simpler Decision problem that we defined for the search problem. The existence problem can be formulated in the Existential Theory of the Reals. Given an instance I of Π, construct the circuit C_I for the instance, introduce a variable for every gate, and write a set of polynomial equations and inequalities with rational coefficients that express the semantics of the gates. The constraints for the $+, -, *$ gates are obvious. The constraint for a division gate $g_i := g_j/g_k$ is $g_i g_k = g_j$. The constraint set for a max gate $g_i := \max(g_j, g_k)$ is $g_i \geq g_j$, $g_i \geq g_k$, $(g_i - g_j)(g_i - g_k) = 0$. The constraint set for a min gate is similar. The constraint set for a root gate $g_i := \sqrt[k]{g_j}$ with k odd is $(g_i)^k = g_j$, and with k even is $(g_i)^k = g_j$, $g_i \geq 0$. Add equalities that equate the outputs of the circuit with the inputs, and add the set of linear inequalities on the input variables x that define the domain D_I. Finally, add inequalities for the existence question that relate the coordinates of x with the given bounds. Call a PSPACE procedure [8, 51] that determines whether the system has a solution.

We can solve the partial computation and (strong) approximation problems by using the above existence problem in a binary search manner, coordinate by coordinate. We can actually (partially) compute up to to any precision a specific desired solution if we want, for example one that maximizes or minimizes specified components. ■

We show that Nash exactly characterizes the search problems that can be cast as fixed points of functions represented by algebraic circuits:

Theorem 18 *The Nash equilibrium problem for 3 (or more) players, is FIXP-complete. In particular, the corresponding Decision, (Strong) Approximation and Partial Computation problems are complete respectively for $FIXP_d$, $FIXP_a$, and $FIXP_{pc}$.*

Proof. It is easy to see that d-player Nash for any d is in FIXP. Let I be an instance (a game), let S_i be the set of strategies of player i, let $n_i = |S_i|$ be their number, let $n = \sum n_i$, and let u_i be the payoff function of player i. The domain of the Nash problem is the cartesian product Δ of the d unit simplexes Δ_{n_i}, $i = 1, \ldots, d$. That is, if the vector $x = (x_{ij})$ denotes a mixed strategy profile, where x_{ij} is the probability of the jth strategy of player i, then $D_I = \Delta$ is described by the set of linear equations $\sum_{j=1}^{n_i} x_{ij} = 1$ for all $i = 1, \ldots, d$, and inequalities $x_{ij} \geq 0$ for all i, j. The Nash equilibria are the fixed points of Nash's function F_I from Δ to itself. The function F_I is given by a simple

formula that uses the algebraic operations $+, -, *, /$ and max. Recall from Section 2 that the ij-th component of $F_I(x)$ is $F_I(x)_{ij} = \frac{x_{ij}+\max\{0,g_{i,j}(x)\}}{1+\sum_{j=1}^{n_i}\max\{0,g_{i,l}(x)\}}$, where $g_{i,j}(x) = u_i((i{:}j); x_{-i}) - u_i(x)$ is the net gain of player i in expected payoff if he switches to his jth pure strategy. The expected payoff $u_i(x)$ is a (multilinear) polynomial $u_i(x) = \sum_{(j_1,\ldots,j_d)\in S} u_i(j_1,\ldots,j_d)x_{1j_1}\ldots x_{dj_d}$, where the summation ranges over all pure strategy profiles in $S = S_1 \times \ldots S_d$. Similarly, $u_i((i{:}j); x_{-i})$ and $g_{i,j}(x)$ are also polynomials. Clearly, we can efficiently build an algebraic circuit (indeed an algebraic formula) that computes F_I.

We show the hardness now. Let Π be a problem in FIXP, and I an instance of Π, which corresponds to a Brouwer function from a domain D_I to itself. By Lemma 16 we can assume that the domain is a unit cube $[0,1]^n$. From I we compute an algebraic circuit C_I over the basis $\{+, -, *, /, \sqrt[k]{\ }, \max, \min\}$ with n input and output variables. Let $x_1, ..., x_n$ be the input variables and $x'_1, ..., x'_n$ the output variables. The circuit has the property that for any input vector $x \in [0,1]^n$, all the operations of the circuit are defined (i.e. there is no division by 0, and even roots are applied only to nonnegative operands) and the output is a vector $x' \in [0,1]^n$. We can assume without loss of generality that all constants of the circuit are nonnegative, since we have subtraction gates available. We will first transform the circuit in several substeps to bring it into a normal form, and then we will construct from it a game using similar gadgets as in the last section.

Step 1: Transformation of the circuit.

First we eliminate the max and min gates, using the identities $\max(a,b) = (a+b)/2 + |a-b|/2 = (a+b)/2 + \sqrt{(a-b)^2}/2$, and $\min(a,b) = (a+b)/2 - |a-b|/2 = (a+b)/2 - \sqrt{(a-b)^2}/2$. That is, each max gate $g_i := \max(g_j, g_k)$ is replaced by a sequence of gates $s_{i1} := g_j + g_k$, $s_{i2} := 2$, $s_{i3} := s_{i1}/s_{i2}$, $s_{i4} := g_j - g_k$, $s_{i5} := s_{i4} * s_{i4}$, $s_{i6} := \sqrt{s_{i5}}$, $s_{i7} := s_{i6}/s_{i2}$, $g_i := s_{i3} + s_{i7}$. A min gate is similarly replaced by an analogous sequence of gates. Let D_1 be the resulting circuit.

Next we move all the divisions to the top, that is, we transform to another circuit that has one division for each variable, which is the final output gate for the variable. We replace every gate g_i of D_1 by two new gates g'_i, g''_i which will play the role of the numerator and denominator, i.e. when we evaluate the circuit for any input in the domain, the value of g_i in the original circuit will equal the value of g'_i/g''_i in the transformed circuit. The transformation is as follows. If g_i is an assignment gate, $g_i := c$ for some constant c or $g_i := x_j$ for some input variable x_j then we set $g'_i := c$ or $g'_i := x_j$ respectively, and $g''_i := 1$. For an addition or subtraction gate $g_i := g_j \pm g_k$ we let $g'_i := (g'_j * g''_k) \pm (g'_k * g''_j)$ (this involves two multiplication gates and one addition or subtraction gate) and let $g''_i := g''_j * g''_k$. A multiplication gate $g_i := g_j * g_k$ becomes $g'_i := g'_j * g'_k$ and $g''_i := g''_j * g''_k$. A division gate $g_i := g_j/g_k$ becomes $g'_i := g'_j * g''_k$ and $g''_i := g''_j * g'_k$. A root gate $g_i := \sqrt[k]{g_j}$ with odd k becomes $g'_i := \sqrt[k]{g'_j}$ and $g''_i := \sqrt[k]{g''_j}$. A root gate $g_i := \sqrt[k]{g_j}$ with even k becomes $g'_i := \sqrt[k]{g'_j * g''_j}$ and $g''_i := \sqrt[k]{g''_j * g''_j}$; i.e., we introduce two intermediate multiplication gates that compute $g'_j * g''_j$ and $g''_j * g''_j$ and take their roots. Note that since the operand $g_j = g'_j/g''_j$ of the original gate is guaranteed to be nonnegative for any input in the domain, it follows that the operands of the new root gates are also nonnegative. Finally, for each output gate out_i of C_I that computes the output variable x'_i, we have a gate $out_i := out'_i/out''_i$.

Let D_2 be the circuit resulting from the above transformation. A straightforward induction shows that for each input vector $x \in [0,1]^n$ the following properties hold: (1) all the gates g''_i representing the denominators are nonzero, (2) all even root gates have nonnegative operands, and (3) for each gate g_i of C_I its value $val(g_i)$ is equal to the ratio $val(g'_i)/val(g''_i)$ of the values of the corresponding gates in D_1.

In the third step of the transformation, we move the subtractions right below the top, that is, for each output variable x'_i there are (at most) two subtractions, one for the numerator out'_i and

one for the denominator out''_i. After this step is finished, there may be some subtraction gates further below in the circuit, but all the gates will be guaranteed to compute nonnegative values for any input in the domain. This step is similar to the transformation of the last section, i.e. for each gate g_i of D_1, except for the top division gates, we introduce two gates g_i^+ and g_i^- for the positive and the negative part. An assignment gate $g_i := c$ or $g_i := x_j$ becomes $g_i^+ := c$ or $g_i^+ := x_j$ (recall we assumed, wlog, that all constants are nonnegative) and $g_i^- := 0$. Addition, subtraction and multiplication gates are transformed as in the previous section.

For a root gate $g_i := \sqrt[k]{g_j}$ we introduce several intermediate gates. First compute $s_j := \min(g_j^+, g_j^-)$ via a sequence of $+, -, /, \sqrt{}$ gates, corresponding to the subexpressions of the expression $(g_j^+ + g_j^-)/2 - \sqrt{((g_j^+)^2 + (g_j^-)^2) - 2 * (g_j^+ * g_j^-)}/2$. This involves two subtraction gates: one that computes the operand of the square root, $((g_j^+)^2 + (g_j^-)^2) - 2 * (g_j^+ * g_j^-)$, and a second one that does the final subtraction of the expression to produce s_j; note that both of them compute nonnegative values. Next we introduce gates $t_j^+ := g_j^+ - s_j$, $t_j^- := g_j^- - s_j$, and then let $g_i^+ := \sqrt[k]{t_j^+}$ and $g_i^- := \sqrt[k]{t_j^-}$. Note that the intermediate subtraction gates t_j^+, t_j^- always compute nonnegative numbers: if g_j has nonnegative value, i.e. $val(g_j^+) \geq val(g_j^-)$, then $val(t_j^+) = val(g_j)$, $val(t_j^-) = 0$, both root gates g_i^+ and g_i^- have nonnegative operands and are thus well-defined whether k is odd or even, and $val(g_i^+) = val(g_i)$, $val(g_i^-) = 0$. If g_j has negative value, i.e. $val(g_j^+) < val(g_j^-)$, then $val(t_j^+) = 0$, and $val(t_j^-) = -val(g_j)$; in this case, k must be odd, and we have $val(g_i^+) = 0$, $val(g_i^-) = -val(g_i)$. Thus, in both cases, the transformation produces the correct value and ensures that all the gates have nonnegative value.

When we reach the children out'_i, out''_i of the division gates at the roots (which compute the outputs x'_i), we have introduced corresponding gates for their positive and negative parts. Note that all the gates below this layer compute non-negative numbers. To ensure that all the gates in the circuit compute non-negative numbers, instead of subtracting the positive and negative parts to compute out'_i, out''_i, we compute the absolute values of the differences, and divide these at the top. That is, we have a gate $aout'_i$ that computes $|(out'_i)^+ - (out'_i)^-|$, and gate $aout''_i$ that computes $|(out''_i)^+ - (out''_i)^-|$ and change the top gates to $out_i := aout'_i/aout''_i$. Note that this does not affect the final outputs: For every input vector in the domain, each output x'_i of C_I is nonnegative, therefore dividing the absolute values yields the same result. The absolute difference $|(out'_i)^+ - (out'_i)^-|$ is computed by introducing gates for the subexpressions of the expression $\sqrt{(((out'_i)^+)^2 + ((out'_i)^-)^2) - 2 * ((out'_i)^+ * (out'_i)^-)}$; note that the subtraction gate that computes the operand of the square root produces a nonnegative value. Similarly with the other gate $aout''_i$ that computes $|(out''_i)^+ - (out''_i)^-|$.

Let D_3 be the circuit resulting after the third substep of the transformation. Clearly, for every input vector in the domain, all the gates in the circuit compute nonnegative numbers. By a straightforward induction it is easy to see that the value of every gate in D_2 below the top two levels is equal to the difference of the two corresponding gates in D_3.

In the last step we transform the circuit D_3 so that all the gates compute numbers between 0 and 1. We do this as follows. The computed numbers are at most doubly exponential in the size of the circuit, and hence in the size of the instance I. Let $d = poly(|I|)$ be such that all the computed numbers are strictly smaller than 2^{2^d}. In the next step we add at the bottom of the circuit D_3 a circuit T that computes $t = 1/2^{2^d}$ by successive squaring of $1/2$ d times, and multiply all the inputs and the constants by t before feeding them into the original circuit; i.e., an assignment gate $g_i := c$ or $g_i := x_j$ becomes $g_i := c * h_d$ or $g_i := x_j * h_d$, where h_d is the last gate of the circuit T that computes t. We modify the gates of D_3 in a similar way as in Section 3. In particular, the

addition and subtraction gates stay the same. A multiplication gate $g_i := g_j * g_k$ is changed to $g_i := (g_j * g_k)/h_d$, broken into two steps, where we first perform the multiplication and then the division. A division gate that is not an output gate, $g_i := g_j/g_k$, is changed to $g_i := (g_j * h_d)/g_k$, broken into two steps where we first perform the multiplication and then the division. The output gates are left alone. A root gate $g_i := \sqrt[k]{g_j}$ is changed to $g_i := \sqrt[k]{g_j * t^{k-1}}$, i.e. we introduce additional gates that compute the powers of $t = h_d$, e.g. $s_2 := h_d * h_d$, $s_3 := s_2 * h_d$, etc., add a gate $\hat{g}_i := g_j * s_{k-1}$, and change the gate g_i to $g_i := \sqrt[k]{\hat{g}_i}$.

Let S be the circuit that results from the last substep of the transformation. The circuit S has the property that for every input vector in the domain, all gates have nonnegative values. Furthermore, except for the top division gates that compute the outputs, every other gate of S that corresponds to a gate of D_3 has value equal to t times the value of the corresponding gate of D_3. This can be shown by a straightforward induction. This fact means in particular that all these gates have values in the interval $[0, 1]$. This obviously holds also for the gates of the initial subcircuit T that we attached in the bottom, and for the other auxiliary gates we introduced. Furthermore it holds also for the output gates $out_i := aout'_i/aout''_i$ because the new factor t in the numerator and denominator cancels out, so the output in S is the same as the output in D_3, and it is in $[0, 1]$.

To summarize, the circuit S uses (constant or input) assignment, addition, subtraction, multiplication, division, and root gates. For every input vector in $x \in [0, 1]^n$, all the gates are well-defined (no division by zero, no negative operands for even root gates), the values of all the gates are in $[0, 1]$, and the outputs of the circuit are equal to the outputs of the given circuit C_I.

Step 2: Construction of the game.

We are ready now to transform the circuit S to a game. As in the reduction of the last section, we first construct a game G with 10 players, and then transform it to a 3-player game Γ. The constructions are very similar to those in the last section.

The game G has 10 players that are paired in 5 pairs i, i', for $i = 1, \ldots, 5$. The structure of the game is the same as in the proof of Theorem 4. Players 1,2,3 are the primary players, players 4,5 are the auxiliary players, their strategies are partitioned into blocks, each primed player i' has one strategy corresponding to each block of the corresponding unprimed player i and plays matching pennies with it. Let N be the number of gates of S, and let $q_1, \ldots, q_N$ be the sequence of gates in topological order. Each primary player $i = 1, 2, 3$ has N blocks, $(i, j); j = 1, \ldots, N$, one for each gate q_j of S, and each block contains two strategies $(i, j, 0)$ and $(i, j, 1)$. An auxiliary player $l = 4, 5$ has $3N$ blocks $(i, j); i = 1, 2, 3; j = 1, \ldots, N$, one for each primary player i and each gate q_j of S; the blocks contain two strategies $(l, i, j, 0)$, $(l, i, j, 1)$, in all cases, except if $i = 1$ and q_j is a k-th root gate, in which case the block contains $k + 1$ strategies $(l, 1, j, 0)$, $(l, 1, j, 1), \ldots, (l, 1, j, k)$.

The payoff functions of the players are defined in a similar manner as in the proof of Theorem 4. In particular the payoff function u_l of each unprimed player $l = 1, \ldots, 5$ is the sum $\mu_l + \mu'_l$ of two terms, where the first term depends on the strategies of the unprimed players only, and the second term on the strategy of l and l' only, just like before. The second term μ'_l and the payoff $u_{l'}$ of the primed players is the same as in Theorem 4; namely if the strategy of l' corresponds to the block of the strategy of l in a pure strategy profile s, then $\mu'_l(s) = M$ and $u_{l'}(s) = -M$, where M is a large number (e.g. $M > 6N \max_s |\mu_l(s)|$); otherwise, $\mu'_l(s) = 0$ and $u_{l'}(s) = -0$.

The first term μ_l of the payoff of each unprimed player l depends only on the strategies of the unprimed players, and is determined by a set of gadgets G_{ij}, one for each primary player $i = 1, 2, 3$ and gate $q_j, j = 1, \ldots, N$. The gadget G_{ij} affects the payoff function of a player for a strategy profile $s = (s_1, \ldots, s_5)$ iff his strategy lies in the block (i, j). The purpose of each gadget $G_{i,j}$ is to 'compute' in the strategy $(i, j, 0)$ of player i the value of the gate q_j as a function of its operands,

in the sense that it ensures that in every Nash equilibrium, the probability of strategy $(i, j, 0)$ of player i is equal to $val(q_j)/N$. For each gate $q_j, j = 1, \ldots, N$ of S, we first 'compute' the value of the gate in the strategy $(1, j, 0)$ of player 1, by a gadget G_{1j} that depends on the operation of the gate, and then copy the value to the corresponding strategies $(2, j, 0)$, $(3, j, 0)$ of the other two primary players 2,3, using simple copying gadgets G_{2j}, G_{3j} that do not depend on the operation of the gate.

The copying gadgets G_{ij} for $i = 2, 3$, $j = 1, \ldots, N$ are exactly the same as in the proof of Theorem 4. The gadgets G_{1j} for assignment, addition, subtraction, multiplication and division gates q_j are also exactly the same. The only difference here is that the inputs to the circuit are variables x_j in addition to constants. Replace all the operands of gates that are input variables by the corresponding output gates; one can think of this as having the outputs of the circuit feed back into it, thus forming a cyclic circuit. That is, if $l' = N - n + l$ is the index of the output gate that computes the lth output x'_l, then we replace all occurrences of x_l as an operand of a gate by $q_{l'}$. For example a multiplication gate $q_j := x_l * q_m$ becomes $q_j := q_{l'} * q_m$. As before, all these gadgets are instances of the gadget G_f of Lemma 7 for a linear polynomial f.

The gadget G_{1j} for a root gate $q_j := \sqrt[k]{q_l}$ is a copy of the gadget G_f for the polynomial $f(z) = z^k + c_k$, where $c_k = -val(q_l)$. As with the other gates, we do not compute explicitly the value of gate q_l but use the value 'computed' by the probability of strategy $(2, l, 0)$. Note that $f(0) \leq 0$ and $f(1) \geq 0$, and the polynomial has a unique root in $[0, 1]$. The blocks $(1, j)$ of the auxiliary players have $k + 1$ strategies in this case.

In more detail, the first term μ_i of the payoff of each unprimed player i for a pure strategy profile $s = (s_1, \ldots, s_5)$ is defined as follows.

Payoff of auxiliary players 4,5. Similar to Theorem 4. The payoff μ_4 is as follows (μ_5 is similar). Let $s_4 = (4, k, j, b_2)$. Strategy s_4 is in block (k, j) of player 4, which corresponds to a gadget G_{kj}, for some primary player $k = 1, 2, 3$ and gate $q_j, j = 1, \ldots, N$. If s_k or s_5 is not in block (k, j) of the respective player k or 5, then $\mu_4(s) = 0$. So suppose $s_k = (k, j, b_1)$ and $s_5 = (5, k, j, b_3)$. Then $\mu_4(s) = h_2(b_1, b_2, b_3)$, where h_2 is the payoff function of player 2 in the game G_f of Lemma 7, for the polynomial f that corresponds to the gadget G_{kj}. (Actually h_2 depends only on the degree of the polynomial, and not the polynomial itself.)

Payoff of primary players 2,3. Exactly the same as in Theorem 4.

Payoff of primary player 1. Let $s_1 = (1, j, b_1)$. If gate q_j of the circuit S is an assignment, addition, subtraction, multiplication, or division gate, then $\mu_1(s)$ is exactly the same as in Theorem 4, except that as we mentioned, we use in place of the input variables the corresponding output gates. Thus, for example if $q_j := x_l * q_m$ then we treat it as if it was $q_j := q_{l'} * q_m$, which means that $\mu_1(s)$ in this case is defined as follows. If s_4 or s_5 are not in block $(1, j)$ then $\mu_1(s) = 0$. Suppose $s_4 = (4, 1, j, b_2)$, $s_5 = (5, 1, j, b_3)$. Then $\mu_1(s) = 1$ if $b_1 = 1$ and $b_2 = 0$; $\mu_1(s) = -N^2$ if $b_1 = 1$, $b_2 = 1$, and $s_2 = (2, l', 0)$ and $s_3 = (3, m, 0)$; and $\mu_1(s) = 0$ otherwise.

Suppose that q_j is a root gate, $q_j := \sqrt[k]{q_l}$. If s_4 or s_5 is not in block $(1, j)$ then $\mu_1(s) = 0$. Suppose $s_4 = (4, 1, j, b_2)$, $s_5 = (5, 1, j, b_3)$. If $b_1 = 0$ then $\mu_1(s) = 0$. Suppose that $b_1 = 1$. If $b_2 = 0$ then $\mu_1(s) = 1$; if $b_2 = k$ and $s_2 = (2, l, 0)$ then $\mu_1(s) = -N$; in all other cases $\mu_1(s) = 0$.

This concludes the definition of the game G.

Let y be a Nash equilibrium of G. By Lemma 6, each block of an unprimed player receives the same total probability in the NE y: probability $1/N$ for the primary players 1,2,3 and $1/3N$ for the auxiliary players 4,5. For each input variable x_j, let $q_{j'}$ be the gate that computes the corresponding output x'_j, and let $x^*_j = N \cdot y(1, j', 0)$ be N times the probability of the corresponding strategy $(1, j', 0)$ of player 1 in the NE y. Let x^* be the input vector $(x^*_j : j = 1, \ldots, n)$.

Claim 2 *For every Nash equilibrium y of the game G, the vector $x^* = (N \cdot y(1, j', 0) : j = 1, \ldots, n)$*

is a fixed point of the function F_I specified by the circuit C_I. Conversely, for every fixed point x^* of F_I, there is a Nash equilibrium y of G such that $x^* = (N \cdot y(1, j', 0) : j = 1, \ldots, n)$.

Proof. Let y be a Nash equilibrium of G. From the copying gadgets G_{2j}, G_{3j} we have $y(1, j, 0) = y(2, j, 0) = y(3, j, 0)$ for all $j = 1, \ldots, N$. In particular for every output gate $q_{j'}$ we have $y(1, j', 0) = y(2, j', 0) = y(3, j', 0) = x^*_j$. Consider the circuits C_I, S with the input vector x^*. By the correctness of the transformation from C_I to S, the output values are equal. We claim that for each gate q_j of S, the value of the gate, $val(q_j)$ on input x^* to the circuit is equal to $N \cdot y(1, j, 0)$ (and hence also $N \cdot y(2, j, 0)$ and $N \cdot y(3, j, 0)$). This can by shown by induction on the height of the gate, in exactly the same way as in the proof of Theorem 4 using the properties of the gadgets G_f. It follows that the value of each output gate $q_{j'}$ is equal to $N \cdot y(1, j', 0)$, i.e. to x^*_j. Thus, x^* is a fixed point of the circuit S, and hence also of the circuit C_I.

Conversely, given any fixed point x^* of C_I, and hence also of S, we can define a profile y for G as follows. Let $val(q_j)$ be the value of gate q_j of S on input x^* to the circuit. Let $y(1, j, 0) = y(2, j, 0) = y(3, j, 0) = val(q_j)/N$ and $y(1, j, 1) = y(2, j, 1) = y(3, j, 1) = (1 - val(q_j))/N$. For each gadget G_{ij}, the probabilities of the primary player correspond to a root of the polynomial f of the gadget, hence they can be extended to the auxiliary players to form a Nash equilibrium for the gadget. The probabilities can be then extended to the primed players to form a Nash equilibrium y for G. ∎

From the game G construct a 3-player game Γ using the transformation of Proposition 9. By the proposition, the Nash equilibria of Γ correspond (in general not 1-1) to the Nash equilibria of G: For every NE z of Γ, its projection z_1 on the strategies of player 1 multiplied by 10 yields a NE of G; and conversely, for every NE y of G there is a NE $z = (z_1, z_2, z_3)$ of Γ such that $z_1 = y/10$. In particular, there is a set of n pure strategies of player 1, say strategies $1, \ldots, n$ (these are the strategies corresponding to the strategies $(1, j', 0); j = 1, \ldots, n$ in the game G), such that for every NE $z = (z_1, z_2, z_3)$ of Γ, if we multiply its projection $z_1[1 \ldots n]$ on these strategies of player 1 by $10N$ we obtain a fixed point of S, and hence also of C_I and F_I; and conversely, for every fixed point x^* of F_I, there is a Nash equilibrium z of Γ such that $x^* = z_1[1 \ldots n]/10N$.

Therefore, the fixed point problem reduces to the Nash equilibrium problem in its exact version, as well as in the decision and approximation version. For the partial computation version where we want to compute a specified number of bits of the fixed point, pad the circuit S with dummy gates so that N becomes a power of 2, $N = 2^l$, and add 6 dummy players to the game G with one strategy each, before transforming it to the 3-player game Γ. Then the relation between the fixed points x^* of the function and the NE z of Γ becomes $x^* = z_1[1 \ldots n]/2^{l+4}$, thus the mapping between the two vectors is just a binary shift. ∎

Another example of a FIXP-complete problem, is the exchange equilibrium problem (defined in Section 2) for a market specified by the excess demand functions. We have a set of m agents and n commodities, and we are given the excess demand functions $g^l_i(p); l = 1, \ldots, m; i = 1, \ldots, n$ of the agents for the different commodities at prices p, and hence also the total market excess demand function $g_i(p) = \sum_l g^l_i(p); i = 1, \ldots, n$. Recall that the equilibria of the market are the fixed points of the function $F : \Delta_n \mapsto \Delta_n$, defined by the formula $F_i(p) = \frac{p_i + \max(0, g_i(p))}{1 + \sum_k \max(0, g_k(p))}$. Clearly, if the functions g^l_i, and hence also g_i, are defined by algebraic circuits (or formulas), then so is the function F and the exchange equilibrium problem is in FIXP. For the completeness, recall that given a Brouwer function $f : \Delta_n \to \Delta_n$, one can define a total market excess demand function $g : \Delta_n \to R^n$ using Uzawa's formula [60]: $g(p) = f(p) - (\langle p, f(p) \rangle / \langle p, p \rangle)p$. The price equilibria are exactly the fixed points of f. Clearly, if f is given by an algebraic circuit we can construct a circuit

for g. Furthermore, we can also reduce directly the 3-player Nash equilibrium problem to the price equilibrium problem as in the proof of Corollary 15 which gives an algebraic formula for g, rather than a circuit. Thus:

Proposition 19 *The exchange equilibrium problem with excess demand functions given by algebraic circuits (or formulas) over* $\{+, -, *, /, \max, \min, \sqrt[k]{}\}$ *is FIXP-complete.*

As we remarked after the definition of the class FIXP, the class does not change if we omit some of the operators, because they can be defined in terms of others. Some other such invariance properties do not follow from the properties of the operators but rather from the completeness of the Nash equilibrium problem. One obvious consequence of the completeness is that the class does not change if we omit the root operators, since Nash's function does not have any roots. Also, FIXP does not change if we require the function F_I to be represented by a formula rather than a circuit, since Nash's function is given by a formula. Furthermore, bounded-depth formulas are enough, if we use addition gates with unbounded fan-in.

A non-obvious consequence is that the class does not change if we omit division. To show this, we will define another function G_I from a game I, whose fixed points are also the Nash equilibria, and which can be implemented without division. Let I be a d-player game with a set S_i of pure strategies for each player, let $n_i = |S_i|$, let $n = \sum n_i$, and let u_i be the payoff function of player i. The domain of the function is the same, $D_I = \Delta$, the cartesian product of the d unit simplexes Δ_{n_i} for the d players. For each mixed strategy profile x, let $v(x)$ be a vector which gives the expected payoff of each pure strategy of each player with respect to the profile x for the other players. That is, vector x is a vector of dimension n whose entries are indexed by pairs $(i,j), i = 1,\ldots,d; j = 1,\ldots,n_i$, and $v(x)$ is also a vector of dimension n whose (i,j)-entry is $u_i((i{:}j); x_{-i})$. Let $h(x) = x + v(x)$. For each player i, consider the function $f_{i,x}(t) = \sum_{j\in S_i} \max(h_{ij}(x) - t, 0)$. Clearly, it is a continuous, piecewise linear function of t. The function is strictly decreasing as t ranges from $-\infty$ (where $f_{i,x}(t) = +\infty$) up to $\max_j h_{ij}(x)$ (where $f_{i,x}(t) = 0$). Thus, there is a unique value of t, call it t_i, where $f_{i,x}(t_i) = 1$. The function G_I is defined as follows: $G_I(x)_{ij} = \max(h_{ij}(x) - t_i, 0)$ for every $i = 1,\ldots,d; j \in S_i$. From our choice of t_i, we have $\sum_{j\in S_i} G_I(x)_{ij} = 1$ for all $i = 1,\ldots,d$, thus $G_I(x)$ is in the domain $D_I = \Delta$. In fact it can be shown that $G_I(x)$ is the point of Δ that is closest to the point $h(x)$, and that therefore the function G_I is the same as the function of [27], which is defined as the projection of $h(x)$ on Δ. With the explicit definition of G_I above, it is actually quite easy to show that the fixed points of G_I are precisely the Nash equilibria; this gives also another elementary proof of Nash's theorem.

Lemma 20 *The fixed points of the function* G_I *are precisely the Nash equilibria of the game* I*.*

Proof. If x is a fixed point of G_I, then $x_{ij} = \max(x_{ij} + v(x)_{ij} - t_i, 0)$ for all i,j. Recall that $v(x)_{ij} = u_i((i{:}j); x_{-i})$ is the expected payoff for player i of his pure strategy $(i{:}j)$ with respect to strategy x for the other players. The equation $x_{ij} = \max(x_{ij} + u_i((i;j); x_{-i}) - t_i, 0)$ implies that $u_i((i{:}j); x_{-i}) = t_i$ for all i,j such that $x_{ij} > 0$, and $u_i((i{:}j); x_{-i}) \le t_i$ for all i,j such that $x_{ij} = 0$. Consequently, x is a Nash equilibrium.

Conversely, suppose that x is a Nash equilibrium, and let $r_i = u_i(x)$ be the expected payoff of player i in x. Then $v(x)_{ij} = u_i((i{:}j); x_{-i}) = r_i$ for all i,j such that $x_{ij} > 0$, and $v(x)_{ij} = u_i((i{:}j); x_{-i}) \le r_i$ for all i,j such that $x_{ij} = 0$. Thus, $x_{ij} = \max(x_{ij} + v(x)_{ij} - r_i, 0)$ for all i,j, and hence $f_{i,x}(r_i) = \sum_{j\in S_i} \max(h_{ij}(x) - r_i, 0) = \sum_{j\in S_i} x_{ij} = 1$. Therefore, $r_i = t_i$, and $G_I(x)_{ij} = \max(x_{ij} + v(x)_{ij} - t_i, 0) = x_{ij}$ for all i,j, i.e., $G_I(x) = x$. ∎

We can implement the function G_I by a circuit without division.

Lemma 21 *We can construct in polynomial time a circuit with basis* $\{+, -, *, \max, \min\}$ *(no division) and rational constants that computes the function* G_I*.*

Proof. The circuit implements the following algorithm. Given a vector $x \in \Delta$, first compute $y = h(x) = x + v(x)$. Clearly y can be computed with $+, *$ gates. For each player i, let y_i be the corresponding subvector of y induced by the strategies of player i. Sort y_i in decreasing order, and let z_i be the resulting sorted vector, i.e. the components of $z_i = (z_{i1}, \ldots, z_{in_i})$ are the same as the components of y_i, but they are sorted: $z_{i1} \geq z_{i2} \geq \ldots \geq z_{in_i}$. The circuit uses a sorting network N_i for each i (see e.g. [37] for background on sorting networks). For each comparator gate of the network we use a max and a min gate. Then we compute t_i as $\max\{(1/l) * ((\sum_{j=1}^{l} z_{ij}) - 1) | l = 1, \cdots, n_i\}$; we will show below that indeed this expression gives the correct value of t_i. Finally, output $x'_{ij} = \max(y_{ij} - t_i, 0)$ for each $i = 1, \ldots, d; j \in S_i$.

We show now that $t_i = \max\{(1/l) * ((\sum_{j=1}^{l} z_{ij}) - 1) | l = 1, \cdots, n_i\}$. Consider the function $f_{i,x}(t) = \sum_{j \in S_i} \max(z_{ij} - t, 0)$ as t decreases from z_{i1} where the function is 0, down until the function reaches the value 1. In the first interval from z_{i1} to z_{i2} the function is $f_{i,x}(t) = z_{i1} - t$; in the second interval from z_{i2} to z_{i3} it is $f_{i,x}(t) = z_{i1} + z_{i2} - 2t$; and so forth. In general, in the l-th interval, $f_{i,x}(t) = \sum_{j=1}^{l}(z_{ij} - t) = \sum_{j=1}^{l} z_{ij} - lt$. If the function reaches the value 1 in this interval, then $t_i = ((\sum_{j=1}^{l} z_{ij}) - 1)/l$. For $k < l$, we have $\sum_{j=1}^{k}(z_{ij} - t_i) \leq \sum_{j=1}^{l}(z_{ij} - t_i) = 1$, and therefore $((\sum_{j=1}^{k} z_{ij}) - 1)/k \leq t_i$. On the other hand, if $l < n_i$, then $t_i \geq z_{l+1}$, and thus for $k > l$ we have $\sum_{j=1}^{k}(z_{ij} - t_i) = 1 - \sum_{j=l+1}^{k}(t_i - z_{ij}) \leq 1$, and thus again $((\sum_{j=1}^{k} z_{ij}) - 1)/k \leq t_i$. Therefore, $t_i = \max\{(1/l) * ((\sum_{j=1}^{l} z_{ij}) - 1) | l = 1, \cdots, n_i\}$. ∎

Theorem 22 *The class FIXP stays the same if the circuits are restricted to use only the operations* $\{+, *, \max\}$ *and rational constants.*

Proof. From the completeness of Nash and Lemmas 20 and 21, we do not need division or roots. Subtraction and min are not needed since we have -1. ∎

5 Piecewise Linear Functions and PPAD

We will define a general class of fixed point problems with piecewise linear, polynomial-time computable functions and show that (i) they have rational fixed points, and (ii) their exact fixed point computation problem is in PPAD. We will in fact show that the following are all equivalent: (a) this piecewise linear class of fixed point problems, (b) PPAD, and (c) a (piecewise) linear fragment of FIXP (which we call Linear-FIXP), having operators $\{+, \max\}$ as well as multiplication by rational constants only.

Consider a fixed point search problem Π: every instance I is associated with a continuous function F_I on a (convex compact) domain D_I. We say that Π is a **polynomial piecewise linear** problem if the following hold: The domain D_I is a polytope defined by a set of linear inequalities with rational coefficients that can be computed from I in polynomial time. The domain is divided by hyperplanes into polyhedral cells, the function F_I is linear in each cell and is of course continuous over the whole domain. The coefficients of the function in each cell and of the dividing hyperplanes are rationals of size bounded by a polynomial in $|I|$. These are not given explicitly in the input, in fact there may be exponentially many dividing hyperplanes and cells. Rather, there is an underlying (polynomial-depth) arithmetic decision tree T_I for deciding if a point x is in the domain

and determining its cell, using linear comparisons of the form $ax \leq b$ (a, b poly-size rationals), and a polynomial-time algorithm for tracing the appropriate path in the tree for a given point x until it reaches a leaf, where it outputs the appropriate linear function $F_I(x)$. Formally, there is an oracle algorithm that runs in time polynomial in $|I|$ which generates a sequence of queries of the form $ax \leq b$? adaptively (i.e., the next query depends on I and the sequence of previous answers), and at the end outputs 'No' (i.e., x is not in the domain) or outputs a (rational) matrix C and vector C' such that $F_I(x) = Cx + C'$.

We give several examples of problems that can be expressed as polynomial piecewise linear fixed point problems.

Simple Stochastic Games (SSG) [12]. This is a two-player game played on a directed graph $G = (V, E)$ whose nodes represent the positions of the game and the edges represent the possible moves. The graph has two sink nodes labeled 0 and 1, and the rest of the nodes are partitioned into three sets, V_0 (random), V_1 (max), V_2 (min). The edges out of each random node are labelled with rational probabilities that sum to 1. A token is placed initially at a node s and then moved along edges until it reaches a sink. If the token is at a node u in V_0 then the edge is chosen randomly according to the probabilities, if $u \in V_1$ then it is chosen by Player 1 who tries to maximize the probability of reaching sink 1, and if $u \in V_2$ then the edge is chosen by Player 2 who tries to minimize the probability of reaching sink 1. For every starting node s, there is a well-defined value x_s of this (zero-sum) game, which is the probability that the token reaches eventually sink 1 if both players play optimally. The task is to compute the value x_s for a specific node s or for all nodes s. In general, each player may base his decision in each step on the whole previous history, and may also randomize. However, it is known that there are optimal pure (deterministic) stationary strategies, i.e., strategies that always select the same successor from each max or min node every time it is visited. For given deterministic strategies σ_1, σ_2 for the players, the game G becomes a Markov chain G_{σ_1,σ_2} and the values can be computed by solving a linear system of equations. Since the transition probabilities are assumed to be rational (as usual) this implies in particular that the values of the game are rational of size (bit complexity) m polynomial in the input size $|G|$. The search problem of computing the vector x of values of a SSG is in PLS [61]. The decision problem $x_s \geq 1/2$? (does Player 1 win with probability at least 1/2) is in $NP \cap coNP$, and it is a well known open problem whether it is in P [12].

Let $n = |V|$ and let x be the n-vector of game values for the different starting nodes. The vector x satisfies a system of equations $x = F(x)$, which are as follows. If u is the 0 sink or the 1 sink then $x_u = 0$ or $x_u = 1$ respectively; if $u \in V_0$ then $x_u = \sum_v p_{uv} x_v$, where the sum ranges over all edges (u, v) with p_{uv} the corresponding probability; if $u \in V_1$ then $x_u = \max\{x_v | (u, v) \in E\}$; if $u \in V_2$ then $x_u = \min\{x_v | (u, v) \in E\}$. The function F maps the domain $D = [0, 1]^n$ to itself. In general there may be multiple solutions to the system $x = F(x)$ in the unit cube, however as shown by Condon [12], every game G can be polynomially reduced to a 'stopping' game G', whose system has a unique solution in the unit cube. A game G is a *stopping game* if it has the property that for every pair of deterministic strategies σ_1, σ_2 for the players, the induced Markov chain G_{σ_1,σ_2} has a path from every node to some sink. For every stopping game, the system $x = F(x)$ has a unique solution, which is of course the vector of values of the game. A simple stochastic game G can be transformed to a stopping game G' by having every non-sink node transition with a suitably small probability δ to the 0 sink, and the node follows with the remaining probability $1 - \delta$ the original transitions (technically, for each max or min node, a random node is added that makes this probabilistic transition). If δ is a sufficiently small rational, of size polynomial in $|G|$ (and the size m of the values of the game G), then the games G and G' have the same optimal (deterministic) strategies, and furthermore the vectors of optimal values of the two games are very close to each other, much closer than 2^{-2m}. (In Condon's model [12] the transition probabilities of a game are

1/2, but it is well-known that her proof works for any rational probabilities.) Hence, by solving the stopping game G' we obtain optimal strategies σ_1, σ_2 for the two players in G, and we can also obtain the optimal values for G by either solving the linear system of equations for the Markov chain G_{σ_1,σ_2}, or by rounding the values for G' to the nearest rationals with denominator at most 2^m (as is well-known, this can be done efficiently with the continued fraction method (see, e.g., [26])).

Clearly the max and min operations, and hence also F, are piecewise linear functions. In this example, the dividing hyperplanes are of the form $x_v = x_w$, for pairs of nodes v, w that are successors of the same node in V_1 or V_2. Thus, in this case there are a polynomial number of hyperplanes, and an exponential number of cells. Clearly, for a given x, we can ask queries that compare the relevant components of x, corresponding to successors of the same max or min node to determine which successor achieves the maximum or minimum, and output $F(x)$.

Linearly interpolated functions. The following discretized model of Brouwer functions F is used in [50]. The input is an integer N (in binary) and a Turing machine M, which given a "grid" point x in the unit cube $[0,1]^n$ with coordinates multiples of $1/N$ returns in polynomial time a displacement vector $\mu(x)$ such that $F(x) = x + \mu(x) \in [0,1]^n$ (the displacement is also constrained to satisfy $|\mu(x)| \leq 1/N^2$ in [50], but this restriction is actually not necessary as we'll see). The function then is extended to a piecewise linear map throughout $[0,1]^n$, by partitioning $[0,1]^n$ into $1/N$-cubelets along the grid hyperplanes, partitioning each cubelet into simplexes in a standard simplicization, and then interpolating the values at the vertices of each simplex. The cells here are the simplexes.

It is not hard to give an algorithm that determines efficiently for a given $x \in [0,1]^n$ the simplex that contains it, and then evaluate F at the vertices of the simplex and compute the linear form of the function $F(x)$ by interpolation: For given $x \in [0,1]^n$, the algorithm first does binary search on the coordinates of x to determine the $1/N$-cubelet that contains x, and then does linear tests to determine the simplex in the standard simplicization of the cubelet. Point x can be written (uniquely) as a convex combination $x = \sum_i \lambda_i v_i$ of the $n+1$ vertices v_i of the simplex, and $F(x) = \sum_i \lambda_i F(v_i)$. The vector λ is a linear function in x, i.e., it has the form $\lambda = Bx + b$, where B, b can be computed by solving the system $x = \sum_i \lambda_i v_i; 1 = \sum_i \lambda_i$ for λ. Run the Turing machine M on the vertices of the simplex to determine their values $F(v_i)$. Let V' be the matrix with columns $F(v_i)$. Then we have $F(x) = (V'B)x + V'b$.
Note: Every Brouwer function F (say on the unit cube $[0,1]^n$) can be approximated by a linearly interpolated function G by imposing a suitably fine grid. However, a fixed point of G may not give a good approximation to any fixed point of F. If F is polynomially continuous then a fixed point of G (for a suitably small but polynomially encodable grid size) is a weak ϵ-fixed point of F. But recall that in general a weak ϵ-fixed point may be very far from any actual fixed point of F (cf. 3-player Nash).

Nash for 2-player and polymatrix games. Nash's function is nonlinear even for 2 players. However, the function of Lemma 20 becomes piecewise linear in the 2-player case. Let I be a 2-player game, S_i the set of strategies of player i, $n_i = |S_i|$ their number, and u_i the payoff function of player $i = 1, 2$. A mixed strategy profile vector $x = (x_{ij} | i = 1, 2; j \in S_i)$ is mapped to $G_I(x)$ whose ij-th component is $G_I(x)_{ij} = \max(h_{ij}(x) - t_i, 0)$, where $h_{ij}(x) = x_{ij} + u_i((i{:}j); x_{-i})$, and t_i is the unique value of t such that $\sum_{j \in S_i} \max(h_{ij}(x) - t_i, 0) = 1$. In the 2-player case, every expected payoff $u_i((i{:}j); x_{-i})$ of a pure strategy $(i{:}j)$ is a linear function of x: $u_1((1{:}j); x_{-1}) = \sum_{k \in S_2} u_1(j,k) x_{2,k}$ and $u_2((2{:}j); x_{-2}) = \sum_{k \in S_1} u_2(k,j) x_{1,k}$. Thus, each $h_{ij}(x)$ is a linear function. Recall the algorithm for computing t_i from Lemma 21: We first sort the vector $y_i = (h_{ij}(x) | j \in S_i)$ into a vector z_i; this involves a sequence of comparisons of components of y_i, i.e. of linear expressions in x, after which

we know the sorted order of the components. Then we compute $t_i = \max\{(1/l)*((\sum_{j=1}^{l} z_{ij})-1) | l = 1, \cdots, n_i\}$, i.e., perform another sequence of comparisons of linear expressions to determine which l gives the maximum value, which yields a linear expression for t_i. We finally compare $h_{ij}(x) - t_i$ with 0, and output the maximum of the two for the ij-component of $G_I(x)$. Thus, the fixed point problem for G_I, which is the 2-player Nash problem, is a polynomial piecewise linear problem.

The same holds more generally for *polymatrix games*, which are multiplayer games where the payoffs are sums of the payoffs of the pairwise interactions, i.e., the payoff function u_i of each player is the sum of $d-1$ functions v_{ik} (where d is the number of players), one for each other player $k \neq i$, where the function $v_{ik}(s)$ for a pure strategy profile s depends only on the strategies of players i and k; that is, an instance of a (normal form) polymatrix game is specified by $d(d-1)$ tables $v_{ik}; 1 \leq i \neq k \leq d$. The functions $h_{ij}(x)$ are again linear in these games, and therefore $G_I(x)$ is piecewise linear.

We now analyze the class of polynomial piecewise linear functions.

Lemma 23 *The class of functions associated with a polynomial piecewise linear fixed point problem is polynomially computable and polynomially continuous.*

Proof. Let Π be a polynomial piecewise linear fixed point problem, let I be an instance of Π, and F_I the associated function. Given instance I and rational x, we first test if x satisfies the linear constraints that describe the domain D_I, and then we apply the oracle algorithm, evaluating explicitly the linear expressions that appear in the comparisons, and finally compute the value $F_I(x)$ of the function. Clearly, this takes polynomial time in $|I|$ and $\texttt{size}(x)$. Thus, Π has a polynomially computable class $\mathcal{F}_\Pi$ of functions.

The polynomial continuity of the class $\mathcal{F}_\Pi$ follows from the fact that F_I is linear in each cell and it is continuous across cells over the whole domain D_I. In more detail, let $d(n)$ be an upper bound on the number of variables for instances I of size n, and $p(n)$ an upper bound on the size of the coefficients of the function F_I in all the cells. Thus, every coefficient is the ratio of two integers with at most $p(n)$ bits. Within every cell, if two points are within distance δ, in L_∞ norm, then their images under F_I are within distance $d(n)2^{p(n)}\delta$ of each other. If two points $r, s \in D_I$ are not in the same cell, then consider the linear segment connecting r and s, which goes through a number of cells. Note that by continuity, on the common boundaries of two cells the functions of the two cells must give the same value. The function F_I increases the distance at most by a factor $d(n)2^{p(n)}$ in each cell, and hence also over the whole segment $r - s$, i.e. $|F_I(r) - F_I(s)| \leq |r - s| d(n)2^{p(n)}$. Thus, to ensure that $|F_I(r) - F_I(s)| \leq \epsilon$, it suffices to have $|r - s| \leq \delta = \epsilon / d(n)2^{p(n)}$; clearly, the size of δ is polynomial in the size of ϵ and $n = |I|$. ∎

Theorem 24 *Polynomial piecewise linear problems have rational fixed points of polynomial size in the input size, and their exact fixed point computation problem is in PPAD.*

Proof. By the previous lemma, polynomial piecewise linear problems have a polynomially continuous and polynomially computable class of functions and hence the weak approximation problem is in PPAD by Proposition 2. We will reduce the exact computation problem to the weak approximation problem. Given an instance of the problem, we pick a suitably small ϵ, compute a weak ϵ-fixed point y, form a LP involving the inequalities on the path traced by the decision algorithm on y, and from the function at y, and show that the solution to the LP gives us an exact fixed point. Thus, we can compute an exact fixed point with one application of Scarf's algorithm and Linear Programming.

In more detail, let Π be a piecewise linear problem. Let $d = d(n)$ be (an upper bound on) the number of variables for instances I of size n, and $p(n)$ a polynomial that bounds the size of the rational coefficients of the hyperplanes and the function F_I. Let m be an upper bound on the bit-size of the solution of any linear system with $d(n) + 1$ equations and coefficients that are rationals of size $p(n)$; i.e., if such a system has a solution then it has one with every component of size $\leq m$. Then m is bounded by a polynomial $q(n)$.

Given an instance I of size n, pick $\epsilon < 1/2^m$. Suppose that y is a weak ϵ-approximate fixed point in the max (L_∞) norm, i.e., $|F_I(y) - y| \leq \epsilon$. Run the decision tree algorithm on y, and let $Ay \leq b$ be the set of inequalities formed by the queries and answers; if some inequalities on the path are strict, we change them to weak inequalities, i.e., $Ay \leq b$ describes the closure of the cell. Note that we can assume without loss of generality that all points that satisfy $Ax \leq b$ are in the domain D_I since the algorithm answers 'Yes' for y on the basis of these comparisons (we could have also constructed the linear constraints for D_I and included them explicitly, but it is not necessary). Let $Cy + C'$ be the output linear function; this is the function F_I for all points that lie in the cell of y. Note that because of the continuity of F_I over the domain, the value $F_I(x)$ of the function on all the points x in the closure of the cell (including the points on the boundary) must be $Cx + C'$; i.e. $F_I(x) = Cx + C'$ for all points x that satisfy the inequalities $Ax \leq b$.

Solve the following LP with variables a vector x and a scalar z:

Minimize z
Subject to:
$Ax \leq b$
$(Cx)_i + C'_i - x_i \leq z$ for $i = 1, \ldots, d$
$x_i - (Cx)_i - C'_i \leq z$ for $i = 1, \ldots, d$

From the last two sets of inequalities, $z \geq 0$, so we don't have to include it explicitly. The vector $(x, z) = (y, \epsilon)$ is a feasible solution to the LP. The minimum value is achieved at a vertex (x^*, z^*) of the feasible set and is at most ϵ. The optimal vertex is the intersection of $d + 1$ constraints, and therefore it is a rational vector whose entries have size at most m. Since $0 \leq z^* \leq \epsilon < 1/2^m$ at the optimal vertex, it follows that $z^* = 0$. Thus, $Cx^* + C' = x^*$, and since x^* satisfies $Ax \leq b$, the point x^* is in the domain and $F_I(x^*) = Cx^* + C' = x^*$. ∎

Corollary 25 *The following problems are in PPAD:*
1. *Compute the value and optimal strategies of a simple stochastic game.*
2. *Compute an exact fixed point of a linearly interpolated function.*
3. *Compute a Nash equilibrium of a polymatrix game.*

Membership of simple stochastic games in PPAD was observed in [34], although it should be mentioned that there is an oversight in the proof there, because it uses a theorem of [50] without noticing that the theorem does not apply to the actual function, but to a modified function obtained by linear interpolation at grid points.

We describe an alternative way to show the membership of simple stochastic games in PPAD, which is also instructive as it shows the relationship between exact computation, strong, and weak approximation for this problem. Recall that every game G can be transformed to a stopping game G' by having the game stop in every step with a suitably small probability δ, so that the values of the nodes in the game G' are very close to the values in G, say within distance $1/2^{3m}$, where m is the bit-size of the values in G. Let $x = F(x)$ be the fixed point system of equations for the non-sink nodes of the original game G, obtained by substituting 0 and 1 respectively for the variable of the 0 sink and the 1 sink. Then the system for G' is $x = F'(x)$ where $F' = (1 - \delta)F$. The function F' is

a contraction mapping with respect to norm L_∞ with contraction factor $1-\delta$ (which is the reason that it has a unique fixed point). Since $size(\delta)$ is polynomial in $|G|$ (and $|G'|$), and F' is obviously polynomially computable, we know from Proposition 2 that the strong approximation problem for G' is in PPAD. Let y be a strong ϵ-approximate fixed point of F', where $\epsilon = 1/2^{3m}$. Then y is within $1/2^{3m}$ of the vector of values of G', and hence within $1/2^{3m-1}$ of the vector of values of G. Since the values of G are rationals with denominator at most 2^m, we can readily obtain the values of G from y by the continued fraction method. Also, the optimal strategies according to y, where each player picks for every node the successor with the best (maximum or minimum) y value, is also an optimal strategy in G.

We remark that although the exact computation of the values of a simple stochastic game G reduces to (strong) approximation of the values for sufficiently small ϵ, which in turn reduces to weak approximation for the associated function F_G for a suitable error δ (of size polynomial in $\texttt{size}(\epsilon)$), this holds only for exponentially small errors. For error ϵ that is constant or polynomially small (i.e., $\epsilon = 1/|G|^c$ for some constant c) it is not at all clear that such a reduction holds. In fact we can easily compute a weak ϵ-approximation for ϵ constant or $1/|G|^c$ as follows. The function F_G is a monotone mapping from the unit cube $[0,1]^n$ to itself: if $x \geq x'$ then $F_G(x) \geq F_G(x')$. If G has multiple fixed points (e.g., if it is not a stopping game), then it has a Least Fixed Point (LFP) in $[0,1]^n$, i.e., a fixed point that is at least as small in all coordinates as all the other fixed points, and this LFP is precisely the vector of values of the game. Starting with the all-0 vector, apply F_G repeatedly; the resulting sequence of vectors $0, F_G(0), F_G^2(0), \ldots, F_G^k(0), \ldots$ is non-decreasing in all coordinates and its limit as $k \to \infty$ is the least fixed point of F_G. Now, if we stop the process when the vector does not increase by more than ϵ in any coordinate from one iteration k to the next, then the obtained vector $F_G^k(0)$ is a weak ϵ-approximate fixed point of F_G. Since the sequence starts with the 0 vector and all coordinates are bounded from above by 1, this will happen after $k \leq n/\epsilon$ iterations, thus, a polynomial number of iterations if $\epsilon \geq 1/|G|^c$. Thus, we can obtain a weak ϵ-approximate fixed point for $\epsilon = 1/|G|^c$ in polynomial time. However, the computed weak approximate fixed point does not help us to estimate the actual values of the game, which is what we are actually interested in. Indeed, it is an open problem whether the values of a simple stochastic game can be approximated with any constant error, for example $1/2$, in polynomial time; this was raised by Condon in her original paper [12].

Returning to the general class of piecewise linear fixed point problems, a consequence of Theorem 24 and the fact that linear interpolated functions, or that 2-player Nash, belong to this class (both of which are complete for PPAD [50, 10]), is that this class is equivalent to PPAD. The same can be shown for the (piecewise) 'linear' part of FIXP: Let **Linear-FIXP** be the class of problems that can be expressed as (polynomially reduced to) exact fixed point computation problems for functions given by algebraic circuits using $\{+, -, \max, \min\}$ (equivalently, {+,max}) and multiplication with rational constants only; no roots, division, or multiplications of two gates/inputs are allowed.

Theorem 26 *Linear-FIXP=PPAD.*

Proof. For the Linear-FIXP $\subseteq$ PPAD direction, it is not hard to show that a circuit as above that does not use multiplication (and division) except by a constant, computes a polynomial piecewise linear function. Let I be an instance, and C_I the Linear-FIXP circuit for I. The following oracle algorithm takes a vector x, asks linear comparison queries, and outputs the linear expression for $C_I(x)$. First construct the linear inequality system that describes D_I and check that x is in D_I. If it is not, return 'No', so assume that $x \in D_I$. Process the circuit bottom-up and compute a linear expression for each gate. For a $+, -$ gate or a gate that multiplies its input by a rational

constant, an expression can be constructed readily from the expressions of the gate's inputs. For a max or min gate, ask a query that compares the linear expressions for the inputs, and accordingly propagate one of the two input expressions. Output the expressions for the output gates of C_I. Thus, the exact fixed point problem for a problem in Linear-FIXP is in PPAD by Theorem 24.

For the other direction, PPAD $\subseteq$ Linear-FIXP, by [10] it suffices to show that 2-player Nash is in Linear-FIXP. Consider the function G_I for a 2-player game I, and the circuit C_I constructed for it in Lemma 21. In the 2-player case, $h_{ij}(x) = x_{ij} + u_i((i{:}j); x_{-i})$ is a linear function of x for all $i = 1, 2; j \in S_i$, as we mentioned earlier in this section when we explained that G_I is piecewise linear. The rest of the circuit uses only $\max, \min, +, -$ and multiplication by rational constants $1/l$. Thus, 2-player Nash is in Linear-FIXP. ∎

6 Shapley Stochastic Games

Stochastic games were first introduced by Shapley in his seminal paper [56] in a more general form where the players take actions simultaneously instead of one at a time (as they do in simple stochastic games). In **Shapley's game** there is a (finite) directed graph $G = (V, E)$, each node (state) u has an associated one-shot zero-sum finite game with a reward (payoff) matrix A_u for player 1 from player 2. If the play is in state u and the players choose actions (pure strategies) i, j then Player 1 receives reward $A_u[i, j]$ from Player 2, the game stops with probability $q^u_{ij} > 0$, and it transitions to state v with probability p^{uv}_{ij}, where $q^u_{ij} + \sum_v p^{uv}_{ij} = 1$. Since there is at least a positive probability $q = \min\{q^u_{ij} | u, i, j\} > 0$ of stopping in each step, the game stops almost surely in a finite number of steps. (Another standard equivalent formulation is as a *discounted game*, where the game does not stop but future rewards are discounted by a factor $1 - q$ per step). The goal of Player 1 is to maximize (and of Player 2 to minimize) the total expected reward. Both players have optimal stationary strategies, i.e., strategies that depend on the current node but not on the history, but the optimal strategies are mixed (randomized) in this case; this is true of course even for one-shot zero-sum games, i.e., the special case when there is only one node and the game stops after the first step. A stationary mixed strategy σ_i for player i is an assignment of a probability distribution at each node to the set of actions of player i at that node. Let $r_u(\sigma_1, \sigma_2)$ denote the expected reward of player 1 under the (mixed) strategies σ_1, σ_2 if the game is started at node u. Then the min-max relation $\sup_{\sigma_1} \inf_{\sigma_2} r_u(\sigma_1, \sigma_2) = \inf_{\sigma_2} \sup_{\sigma_1} r_u(\sigma_1, \sigma_2)$ holds, where σ_1, σ_2 range over all mixed strategies of the two players; this quantity is the *value* of the game (started at u).

An instance of a Shapley stochastic game consists of the graph $G = (V, E)$, the reward matrices $A_u, u \in V$, the transition and stopping probabilities. As usual we assume that all rewards and probabilities in the input are rationals. Let $x = (x_u | u \in V)$ be the vector of game values for the different starting states u. We would like to compute the vector x of game values, and optimal strategies for the players. Unlike simple stochastic games, the values and optimal strategy probabilities in general may be irrational now, so we may not be able to compute them exactly. We would like to bound them, e.g., answer decision questions, such as, is the value $x_u \geq 1/2$?, and/or approximate the values.

The vector x of values satisfies a fixed point equation as follows. For each node u, let $B_u(x)$ be the matrix, whose rows and columns correspond to the actions of the players, and whose i, j entry is $A_u[i, j] + \sum_v p^{uv}_{ij} x_v$. Let $Val(B_u(x))$ be the value of the one-shot zero-sum game represented by the reward matrix $B_u(x)$. The optimal value vector of the stochastic game satisfies the system of equations $x = F(x)$ where $F_u(x) = Val(B_u(x)),\ u \in V$. Furthermore, there is a unique solution, because F is a Banach function (a contraction map) under the L_∞ norm with contraction factor

$1-q$. Note that F is a nonlinear function. For the domain we can pick a box $[-M,M]^n$ with M large enough ($M = \max\{|A_u[i,j]||u,i,j\}/q$ suffices).

Theorem 27 *The following hold for computing the game value in Shapley's games.*
1. Computing the game value is in FIXP; hence Shapley reduces to (3-player) Nash.
2. The (strong) approximation problem is in PPAD.
3. The decision and partial computation problems are at least as hard as SQRT-SUM.

Proof. 1. We cannot directly build an algebraic circuit for Val because we do not even know if LP can be solved in a number of algebraic operations that is polynomial in the dimension of the problem. Instead we introduce a vector y of additional variables for the probabilities of all the actions (pure strategies) of both players at all the nodes. The domain for y is a cartesian product of unit simplexes, one for each node and each player. Thus, the domain of x, y can be readily described by a set of linear equations and inequalities.

We have the following equations. For each node u, let y_u be the subvector consisting of both players' actions at node u. We have the (vector) equation $y_u = \phi_u(y_u, x)$, where ϕ_u is the Nash function (or the function G of Lemma 20) applied to the actions at u and the reward matrix $B_u(x)$, and the equation $x_u = (y_u^1)^T \cdot B_u(x) \cdot y_u^2$, where y_u^1, y_u^2 are the subvectors of y_u corresponding to the strategies of players 1 and 2. Clearly these are algebraic formulas and we can easily build a circuit for them. Obviously, the optimal strategies for the players and the optimal rewards satisfy these equations. On the other hand, consider a fixed point x, y. Since y_u is a fixed point of Nash's function for $B_u(x)$, it defines optimal mixed strategies with respect to $B_u(x)$ and thus $x_u = Val(B_u(x))$. Hence x is the vector of values of the game.

2. The mapping F is a polynomially contracting function, since the distance from 1 of the contraction factor is q, which has polynomial size. It is also polynomial-time computable: for any rational x, we can compute the payoff matrix $B_u(x)$ for each node u, and then compute the value $Val(B_u(x))$ of the corresponding one-shot zero-sum game by Linear Programming. Therefore, by Proposition 2, strong approximation reduces to weak approximation, and is in PPAD.

3. This is shown by a modification of a construction in [20] that reduces the SQRT-SUM problem to the decision problem for concurrent stochastic games with a reachability objective. We are given a set of positive integers $a_1, \ldots, a_n$ and another integer k, and would like to determine whether $\sum_{i=1}^n \sqrt{a_i} \geq k$. We can clearly assume that $a_i > 1$ for all i. Construct a graph G for the stochastic game that contains a start node s, it has n other nodes $u_1, \ldots, u_n$, and a target node t.

The one-shot games for the various nodes are as follows. At each node u_i, each player has two actions 1,2. The reward is 0 for all combinations of actions. The transition and stopping probabilities are defined as follows. Compute a rational number m_i that approximates $\sqrt{a_i}$ from above within $1/2a_i$, i.e., $0 \leq m_i - \sqrt{a_i} \leq 1/(2a_i)$. We can clearly compute efficiently such a rational approximation with standard methods. Let $c_{i,2} = (m_i^2 - a_i)/4$, $g_i = m_i - 1 - c_{i,2}$, and $c_{i,1} = g_i c_{i,3}$, where $0 < c_{i,3} < 1$ is a small-sized rational value such that $c_{i,3} < 1/2g_i$. If players 1 and 2 choose different actions at node u_i then the game stops with probability 1. If both players choose action 1, then the game moves with probability $c_{i,1}$ to t, stays with probability $c_{i,2}$ at node u_i, and stops with the remaining probability $1 - c_{i,1} - c_{i,2}$. If both players choose action 2, then the game moves with probability $c_{i,3}$ to t and stops with the remaining probability $1 - c_{i,3}$. It is easy to see that these are all legitimate probabilities. The reward at u_i is 0 in all cases.

At the start node s, the players have only one action (i.e., effectively they have no choice). At node s the game stops with probability 1/2, and transitions to node u_i with probability E/e_i for each $i = 1, \ldots, n$, where $e_i = c_{i,3}/2c_{i,2}$ and $E = 1/2(\sum_{i=1}^n \frac{1}{e_i})$. The reward at node s is 0. At the target node t, the players have also only one action. The reward is 1, and the game stops with

probability 1. This concludes the specification of the game Γ. Since there is nonzero probability of stopping at every node and for every combination of player actions, Γ is a Shapley game.

Note that a reward (of 1 unit) is only obtained at node t, and the game stops after that. Therefore, optimizing (maximizing or minimizing) the reward is equivalent to optimizing the probability of reaching the target node t. The analysis follows closely the analysis of [20]. The value of the game at t is clearly 1. Let x_i be the value of the game at node u_i. Then $x_i = Val(B_i)$, where the 2×2 matrix B_i for the one-shot zero-sum matrix game at u_i has $B_i[1,1] = c_{i,1} + c_{i,2}x_i$, $B_i[2,2] = c_{i,3}$, and $B_i[1,2] = B_i[2,1] = 0$. Note that $B_i[1,1] > 0$ and $B_i[2,2] > 0$. If the optimal strategy of player 1 at u_i is to play 1 with probability p_i and 2 with probability $1 - p_i$, then by basic facts about zero-sum matrix games we must have $0 < p_i < 1$ and $x_i = p_i(c_{i,1} + c_{i,2}x_i) = (1 - p_i)c_{i,3}$. So $p_i = c_{i,3}/(c_{i,1}+c_{i,2}x_i+c_{i,3})$, and substituting this expression for p_i in the equality $x_i = p_i(c_{i,1}+c_{i,2}x_i)$, we have: $c_{i,2}x_i^2+(g_ic_{i,3}+c_{i,3}-c_{i,2}c_{i,3})x_i-g_i(c_{i,3})^2 = 0$. Solving for x_i, and keeping only the positive root (since x_i is a probability), we can derive after some calculations that $x_i = d_i + e_i\sqrt{a_i}$, where $d_i = -(g_ic_{i,3} + c_{i,3} - c_{i,2}c_{i,3})/2c_{i,2}$ and $e_i = c_{i,3}/2c_{i,2}$ as defined above.

The value at node s is $x_s = \sum_{i=1}^{n}(E/e_i)x_i = D + E\sum_{i=1}^{n}\sqrt{a_i}$, where $D = E\sum_{i=1}^{n} d_i/e_i$. Note that D and E are rational numbers that we can easily compute from the input, and $E > 0$. Let $r = D + Ek$. Then $\sum_{i=1}^{n}\sqrt{a_i} \geq k$ iff the value x_s of the game starting at s is at least r. For the partial computation problem, just add reward $-(D + Ek)$ at the start node s. Then $\sum_{i=1}^{n}\sqrt{a_i} \geq k$ iff $x_s \geq 0$. ∎

Given part (3), the decision problem for Shapley's game, i.e., "is the value $\geq r$ for given rational r?", is not likely to be placed easily in PPAD (or NP).

7 Branching Processes, SCFGs, and 1-RMCs

Branching Processes, Stochastic Context-Free Grammars, and 1-exit Recursive Markov Chains, are three closely related probabilistic models. We will show in this section that the computation of the basic quantities of interest for these models is in FIXP.

A *(Multitype) Branching Process* (MT-BP) [39, 29] $G = (V, R)$ consists of a (finite) set $V = \{S_1, \ldots, S_k\}$ of *types*, and a (finite) set R of rules $S_i \xrightarrow{p} \alpha$, where $S_i \in V$, $p \in (0, 1]$, and α is a (finite) multi-set whose elements are in V, and such that for every type S_i, $\sum_{\langle p_j | (S_i \xrightarrow{p_j} \alpha_j) \in R\rangle} p_j = 1$. The rule $S_i \xrightarrow{p} \alpha$ specifies the probability with which an entity of type i generates the multiset α of offspring in the next generation. The MT-BP specifies a stochastic reproduction process, which starting with an initial finite set of entities of given types (we usually start with one entity of some type S_j), proceeds in discrete steps from one generation to the next, as follows. For each entity in the current set, independently and simultaneously, a rule is chosen at random (according to the rules' probabilities), whose left hand side is the type of the entity, and the entity is replaced with a new set of entities whose types are specified by the right hand side of the rule, to produce the next generation. The process continues as long as the current set of entities is not empty and terminates if and when it becomes empty. The basic quantities of interest are the *extinction probabilities*: the extinction probability $p(S_j)$ of type S_j is the probability that, starting with one entity of type S_j, the process will terminate (die out). Clearly, given these probabilities for all types, it is easy to compute the probability of extinction for any initial set of entities, by multiplying the extinction probabilities of the entities in the initial set. The multisets on the right-hand sides of the rules could either be given explicitly by listing all member types as many times as they occur in the multiset, or more succinctly, by specifying the multiplicities of the types in binary notation. As

shown in [19], the computation of the extinction probabilities in the succinct version reduces to the explicit listing version (by introducing additional types).

A *Stochastic Context-Free Grammar* (SCFG) is a tuple $G = (T, V, R, S_1)$, where T is a set of *terminal* symbols, $V = \{S_1, \ldots, S_k\}$ is a set of *variables* (or *non-terminals*), and R is a set of rules $S_i \xrightarrow{p} \alpha$, where $S_i \in V$, $p \in (0, 1]$, and $\alpha \in (V \cup T)^*$, such that for every non-terminal S_i, $\sum_{\langle p_j | (S_i \xrightarrow{p_j} \alpha_j) \in R \rangle} p_j = 1$. S_1 is specified as the starting nonterminal. A SCFG G generates a language $L(G) \subseteq T^*$ and associates a probability $p(\tau)$ to every terminal string τ in the language, according to the following stochastic process. Start with the starting nonterminal S_1, pick a rule with left hand side S_1 at random (according to the probabilities of the rules) and replace S_1 with the string on the right-hand side of the rule. In general, in each step we have a string $\sigma \in (V \cup T)^*$; take the leftmost nonterminal S_i in the string σ (if there is any), pick a random rule with left-hand side S_i (according to the probabilities of the rules) and replace this occurrence of S_i in σ by the right-hand side of the rule to obtain a new string σ'. The process stops only when (and if) the current string σ has only terminals. The probability $p(\tau)$ of a terminal string is the probability that the process terminates with the string τ. In the above definition we used leftmost derivation for concreteness, but rightmost or any other derivation rule yields the same probabilities $p(\tau)$ for the terminal strings of the language. The probability of the language $L(G)$ of the SCFG G is $p(L(G)) = \sum_{\tau \in L(G)} p(\tau)$. Note that $p(L(G))$ is the probability that the stochastic process that we described above, starting with S_1 terminates. More generally, we can define for each nonterminal $S_j \in V$ an associated probability $p(S_j)$, which is the probability that the process starting with S_j terminates.

A *Recursive Markov Chain* (RMC) consists of a collection $\{A_1, \ldots, A_k\}$ of component Markov chains that can call each other in a potentially recursive manner like recursive procedures. Each component chain A_i has a set N_i of nodes, a set B_i of *boxes* (these correspond to recursive calls), and a set E_i of edges (transitions). A subset $En_i \subseteq N_i$ of nodes is specified as the set of *entries* of A_i and another disjoint subset $Ex_i \subseteq N_i$ is specified as the set of *exits*; the entries have no incoming edges and represent nodes where execution of A_i can start, and exits have no outgoing edges and represent nodes where execution of A_i terminates. Every box $b \in B_i$ is associated with a component $A_{Y(b)}$ via a labelling function Y that maps each box to the index of a component that is called recursively when there is a transition to the box b. Each transition $(u, v) \in E_i$ goes from a vertex u that is either a (non-exit) node of N_i or a pair (b, ex) where $b \in B_i$ is a box and ex is an exit of the associated component $A_{Y(b)}$, to a vertex v that is either a (non-entry) node of N_i or a pair (b, en) where $b \in B_i$ is a box and en is an entry of the associated component $A_{Y(b)}$. The term *vertex* is used to refer collectively to the nodes and the entries (b, en) and exits (b, ex) of the boxes. Every transition (u, v) has an associated (rational) probability p_{uv}. Note that exits of components and entries of boxes do not have any outgoing transitions; every other vertex has outgoing transitions and their probabilities must sum to 1. An *1-exit Recursive Markov chain* (1-RMC) is an RMC in which every component has only 1 exit.

An RMC, started at a given initial vertex, represents a stochastic process, an infinite state Markov chain, where the state contains both the current vertex and the stack of pending recursive calls (i.e., a sequence of boxes). At each step a transition is chosen out of the current vertex according to the transition probabilities; if the transition leads to the entry (b, en) of a box, then the current component is suspended, the box b is added to the stack and a call to the corresponding component $A_{Y(b)}$ is initiated at its entry en. If the transition leads to an exit ex of the current component then the current call terminates; if the stack is empty then the whole execution terminates, otherwise the top box b is retrieved from the stack and the calling component resumes from the exit (b, ex) of the box b. We refer to [19] for more detailed descriptions and thorough treatment of the theory of RMCs. The key quantity of interest in the analysis of RMCs is the probability of

termination: for a given vertex u (of some component), what is the probability q_u that the RMC started at vertex u (with empty stack) will eventually terminate?

It was shown in [19] that MT-BPs, SCFGs, and 1-RMCs are computationally equivalent, in the sense that there are pairwise (polynomial) reductions between the computation of the extinction probabilities of MT-BPs, the probability of the language of SCFGs, and probability of termination of 1-RMCs. These probabilities are all in general irrational. The "qualitative" problem of determining whether the probabilities are 0,1, or in-between can be solved in polynomial time; i.e., for example we can determine in P-time whether the language of a SCFG has probability 1. On the other hand, the "quantitative" decision problem of determining whether a specified termination probability is at least a given rational r is SQRT-SUM and PosSLP-hard. We show in this section that the computation of these probabilities is in FIXP.

We will use 1-RMCs in our analysis below for concreteness. Let $\{A_1, \ldots, A_k\}$ be a 1-RMC. Let Q, N, E, B be respectively the set of vertices, nodes, edges, and boxes of (all the components of) the RMC A. Let $q = (q_u | u \in Q)$ be the vector of termination probabilities of all the vertices, let $x = (x_u | u \in Q)$ be a corresponding vector of variables, and let $n = |Q|$. The termination probabilities satisfy a system of fixed point equations $x = P(x)$, where $P : \mathbb{R}^n_{\geq 0} \mapsto \mathbb{R}^n_{\geq 0}$, and each coordinate function $P_u(x)$ is given by a polynomial with positive rational coefficients. Specifically, there are three types of equations (polynomials), depending on the type of vertex u. If u is an exit then $P_u(x) = 1$. If u is a non-entry node or a box-exit then $P_u(x) = \sum p_{uv} x_v$ where the summation ranges over all transitions (u, v) out of u. If u is a box-entry, i.e., $u = (b, en)$ for some box b, and the exit of the box is $v = (b, ex)$, then $P_u(x) = x_{en} x_v$. The mapping P is a nonlinear, monotone operator, and has a *Least Fixed Point* (LFP): a point $x^* \in [0, 1]^n$ such that $x^* = P(x^*)$ and $x^* \leq x'$ for all other fixed points $x' \in \mathbb{R}^n_{\geq 0}$. The LFP is precisely the vector q of termination probabilities [19].

The function P can clearly be implemented by an algebraic circuit (in fact, simple formulas), and it maps the unit cube to itself. The problem is that in general there are multiple fixed points in the unit cube, and to place the problem in FIXP we have to 'eliminate' them, by modifying the domain and the function in such a way that the desired LFP is left as the unique fixed point in the domain.

Theorem 28 *Computing the termination probabilities for 1-RMCs, MT-BPs, and SCFGs, is in FIXP. Thus the decision/approximation problem reduces to decision/approximation for (3 player) Nash equilibria.*

Proof. We will prove that the domain and the polynomial map $P(x)$ for 1-RMCs (equivalently, MT-BPs, SCFGs) can be transformed using min-max algebraic operations in such way that $x = P(x)$ has a unique solution which is the LFP, x^*. The result follows immediately.

Given a 1-RMC A, we construct the system $x = P(x)$ associated with it. We first find and eliminate the nodes with termination probability 0 or 1, i.e., those entries i such that $x^*_i = 0$ and $x^*_i = 1$. This can be done in P-time ([19]), yielding a reduced system $x = F(x)$ on the remaining variables. From the theory of branching processes and 1-RMCs [29, 19], we know that the reduced system (i) has a unique fixed point in the semi-open unit cube $B[0, 1) = \{x \in \mathbb{R}^n | 0 \leq x < 1\}$, which is the LFP x^*, and (ii) for any point $x \in B[0, 1)$, the sequence $F^k(x), k = 1, 2, \ldots$ converges to x^*. The problem is that Brouwer's theorem and the definition of FIXP require the domain to be a closed (convex) set and $B[0, 1)$ is not closed. We will modify the function so that the domain becomes the closed cube, while avoiding the boundary fixed points.

We need the following crucial lemma:

Lemma 29 *The LFP, x^*, for the reduced n-dimensional system $x = F(x)$ for a 1-RMC (without 0 and 1 nodes) has the property that for all i, $x_i^* \leq (1-\epsilon)$, where $\epsilon = 1/2n2^{|A|^c}$ for a constant c, where $|A|$ denotes the encoding size of the 1-RMC.*

Proof. The proof requires concepts and results from [19]. Construct the dependency graph G of $x = F(x)$: the graph has one node for each variable, and has an edge from x_i to x_j if x_j appears in $F_i(x)$. We will argue that every bottom strongly connected component (SCC) of G contains a variable x_j such that its value in the LFP x^* is $x_j^* \leq 1 - 1/2^{poly(|A|)}$, and then we will use this fact to show that this holds for all variables.

Consider any bottom SCC C in the dependency graph G. We can confine $x = F(x)$ to C to get a subsystem $x_C = R(x_C)$, where x_C is the subvector of x consisting of the variables in C. Let $B(x_C)$ denote the Jacobian matrix (matrix of partial derivatives) of $R(x_C)$, and let $B(1)$ be the matrix evaluated at the all 1 vector (i.e., $B(1)$ is the so-called moment matrix of $R(x_C)$). Let $r = \rho(B(1))$ be the spectral radius of $B(1)$. By a result in [19], the fact that the value in the LFP of the variables of C is strictly less than 1 implies that $r = \rho(B(1)) > 1$. Moreover, there is a nonzero vector $u \geq 0$ such that $B(1)u = ru$ (u is an eigenvector associated with the eigenvalue r).

Let I' be the set of indices i such that $u_i > 0$, and let $B'(1)$ be the corresponding square submatrix of $B(1)$ indexed by rows and columns in I'. Since $r > 1$, we can argue (as in Lemma 31 of [19]), that there is a rational vector $y' \geq 0$ indexed by I' such that $B'(1)y' \geq y' + 1$, and all entries of y' have size $m = poly(|A|)$, polynomial in the size of A. To see this, note that if we scale u so that its smallest nonzero entry is at least $1/(r-1)$, then its subvector on I', $u[I']$, satisfies $B'(1)u[I'] = ru[I'] \geq u[I'] + 1$; thus, the LP $B'(1)y' \geq y' + 1; y' \geq 0$ has a solution, and therefore it has a (nonnegative) rational solution $y' \neq 0$ all of whose coordinates are of size m polynomial in n and the size of the entries of $B'(1)$. The entries of $B'(1)$ are 0,1, and transition probabilities of the RMC A. Therefore m is polynomial in $|A|$. Thus, every nonzero coordinate of y' is between $1/2^m$ and 2^m, where $m = poly(|A|)$.

Let z be a vector, indexed by the variables in C, that is $1 - 1/2n2^m$ in all coordinates, and let $B(z)$ be the Jacobian $B(x_C)$ evaluated at z. The matrix $B(z)$ is the same as $B(1)$ except that some entries decreased by $\leq 1/2n2^m$. Let y be the vector indexed by the variables in C, with $y_i = y'_i$ for $i \in I'$ and $y_i = 0$ for $i \notin I'$. Consider $B(z)y$. For an index $i \notin I'$, we know that $(B(1)u)_i = ru_i = 0$, hence $B_{ij}(1) = 0$ for all $j \in I'$, and since $B(z) \leq B(1)$, the same holds for $B(z)$. Therefore, $(B(z)y)_i = 0$ for all $i \notin I'$. For $i \in I'$, we have $(B(z)y)_i \geq (B(1)y)_i - 1/2$ because every entry of $B(z)$ differs from the corresponding entry of $B(1)$ by at most $1/2n2^m$, there are n entries in each row, and all coordinates of y are at most 2^m. Thus, $(B(z)y)_i \geq y_i + 1/2 \geq ay_i$ for all $i \in I'$, where $a = 1 + 1/2^{m+1}$. Therefore, $B(z)y \geq ay$, which implies that the spectral radius of $B(z)$ satisfies $\rho(B(z)) \geq a > 1$ (by applying basic facts from Perron-Frobenius theory, e.g., Theorem 8.3.2 in [32]).

We show now that C contains a variable x_j such that $x_j^* < z_j = 1 - 1/(2n2^m)$. As shown in [19], the spectral radius of the Jacobian evaluated at the LFP, $B(x_C^*)$, satisfies $\rho(B(x_C^*)) \leq 1$. Since $\rho(B(x))$ is a monotonic function of x, and $\rho(B(z)) > 1$, it must be the case that $z_j > x_j^*$ for some coordinate j. Therefore, $x_j^* < 1 - 1/(2n2^m)$.

Thus, every bottom SCC contains a variable x_j such that $x_j^* < 1 - 1/(2n2^m)$. Every node x_i of the dependency graph G can reach some bottom SCC, hence every node x_i has a path, of length at most n, to some variable x_j with $x_j^* < 1 - 1/(2n2^m)$. If p is the smallest transition probability of the 1-RMC, then the path from x_i to x_j implies that $x_i^* \leq 1 - p^n(1/(2n2^m)) = 1 - (p^n/(2n2^m))$. Let ϵ in the statement of the lemma be $\epsilon = (p^n/(2n2^m))$. Then $x_i^* \leq 1 - \epsilon$ for all i. ∎

Now, define a function $F' : [0,1]^n \mapsto [0,1]^n$ as follows: $F'(x) = F(\min(x, 1-\epsilon))$. That

is, trim if necessary every component x_i of x to $1-\epsilon$, i.e., set $y_i = \min(x_i, 1-\epsilon)$, and then output $F(y)$. Clearly F' is also continuous and monotone, and the original LFP x^* of F is also a fixed point of F'. We claim there are no other fixed points. Suppose z is another fixed point, $F'(z) = z$, and let $z' = \min(z, 1-\epsilon)$. Now $z' \leq z = F(z')$, by definition of $F'(z)$. Thus we have $z' \leq F^k(z') \leq \lim_{k\to\infty} F^k(z')$, by monotonicity of $F(x)$. But we know that for every vector z', $0 \leq z' < 1$, $\lim_{k\to\infty} F^k(z')$ converges to the LFP, x^* (this follows from, e.g., Theorem 7.2 in [29]) so $z = x^*$. Thus $F' : [0,1]^n \mapsto [0,1]^n$ is a FIXP function with a unique fixed point which is the LFP, x^*. ∎

The decision problem for 1-exit RMCs, equivalently MT-BPs and SCFGs, (is the termination probability $\geq r$? for a given rational r) is SQRT-SUM- and PosSLP-hard [19]. We do not know if this holds for the approximation problem. RMCs with multiple exits are more powerful that 1-exit RMCs, and appear to be computationally harder to analyze in some respects. Intuitively, the difference in modeling power is that in a multi-exit RMC, when a recursive call terminates it can pass back a finite amount of information to the calling procedure, whereas in a 1-exit RMC it does not pass back any information, besides the fact that it terminated. It is known that the qualitative problem for multi-exit RMCs, e.g., determining whether it terminates with probability 1, is SQRT-SUM and PosSLP-hard, and hence seems less likely to be (or at least much harder to show) in P. Furthermore, the problem of approximating the termination probabilities within any constant < 1 is also SQRT-SUM- and PosSLP-hard; this was stated in the conference version of the present paper [21], and the proof is given in the full version of [19]. We remark that the weak ϵ-approximation problem for these models (including multi-exit RMCs) can be solved in polynomial time for any $\epsilon > 0$ that is constant or even an inverse polynomial ($1/n^c$ for some c), in the same way as we described for simple stochastic games, i.e., by computing $F^k(0)$ for $k = 1, 2, \ldots$ until a weak ϵ-fixed point is reached. However, such a point does not provide any useful estimation of the termination probabilities.

8 Conclusions

We have studied the complexity of the Nash equilibrium problem for games (with any number of players) and, more generally, fixed point computational problems. We have shown that any nontrivial approximation of (actual) Nash equilibria is at least as hard as longstanding open problems, e.g., the square root sum problem, and a circuit decision problem (PosSLP) that characterizes the power of polynomial time in the unit-cost exact rational arithmetic RAM model with arbitrary precision. These problems are not known to be in NP, nor even in the polynomial-time hierarchy (the best upper bound for them is the Counting Hierarchy). We especially view the reduction from PosSLP as a strong indication that the (approximate) computation of actual Nash equilibria is not in P, nor even in NP (and thus not in PPAD either). We showed similar results for market equilibria: given an exchange economy with a unique equilibrium, it is SQRT-SUM-hard and PosSLP-hard to predict with any nontrivial accuracy the prices of the commodities in the equilibrium.

We defined a class, FIXP, that captures search problems that can be formulated as fixed point problems for algebraically defined Brouwer functions with basis $\{+, -, *, /, \max, \min\}$. We showed that the 3-player Nash equilibrium problem is FIXP-complete. In other words, Nash equilibria for 3 (or more) players encapsulate fixed points of all algebraic functions. We showed that the piecewise linear fragment of FIXP corresponds to fixed points for (polynomial) piecewise linear functions and coincides with PPAD. We also showed that the class FIXP is quite robust in several respects: with respect to the set of algebraic operators allowed, the domain of the Brouwer functions, and whether algebraic circuits or formulas are used to define the class.

We discussed several important probabilistic and game theoretic models, for which the basic computational problems can be formulated in a fixed point framework: simple stochastic games (SSGs), Shapley's original stochastic games, branching processes, stochastic context-free grammars, and 1-recursive Markov chains. The first one (SSGs) has a piecewise linear fixed point function, while all the rest have nonlinear functions and irrational solutions in general. In these problems we want to compute a specific fixed point of a (simple algebraic) function associated with a given instance. In these types of problems, where we seek a specific solution, the domain (and/or the function) have to be defined so that the desired solution is the unique fixed point in the domain. The fact that these (open) problems are in FIXP means that 3-player Nash is at least as hard as them. In the case of SSGs the problem is in PPAD (and thus reduces to 2-player Nash), and the same holds for approximation of the value of Shapley's game, but not for the decision problem, which we showed is square-root-sum-hard.

There are a number of open problems raised by this work. Besides the concrete problems we have discussed, whose complexity status is not known, it would be interesting and important to gain a better understanding of the relationship between FIXP (and the associated discrete problems) and other complexity classes. First, regarding the relationship with PPAD, we know clearly PPAD $\subseteq$ FIXP. Is the containment strict? The class FIXP represents fixed point problems of general (non-linear) algebraic functions, while PPAD represents piecewise linear functions, with 3-player Nash and 2-player Nash being respectively the prototypical complete problems for the classes. In terms of approximation, FIXP represents strong approximation (i.e., approximate computation of an actual fixed point) and PPAD represents weak approximation (getting a point that is almost fixed). The intuition is that the former tasks are strictly harder, so we conjecture that the containment is strict.

A related set of questions regarding FIXP is to characterize what minimal sets of algebraic operators suffice to capture it, and also what operators we can add without increasing its power. We have shown that algebraic circuits with the operators $\{+, *, \max\}$ suffice to capture FIXP. On the other hand, adding all the operators $\{+, -, *, /, \max, \min, \sqrt[k]{}\}$ does not change the power of FIXP. What other operators can we add without increasing the power of FIXP? What if instead we restrict the set of operators to only $\{+, *\}$, i.e., to standard algebraic circuits that only define polynomial functions? Is the $\{+, *\}$ class strictly weaker than FIXP? On the other hand, does the $\{+, *\}$ class contain Linear-FIXP = PPAD (i.e., $\{+, \max\}$ and mulitiplication by rational constants only)? Note that the $\{+, *\}$ class is of interest because such functions are differentiable (as opposed to only piecewise-differentiable in the case of $\{+, *, \max\}$, and only piecewise-linear in the case of $\{+, \max\}$). Thus, e.g., standard root finding methods for differentiable functions, like Newton's method, are more directly applicable to the $\{+, *\}$ class.

The discrete computational problems associated with FIXP are somewhere between P and PSPACE. Can the upper bound be improved? We cannot expect dramatic improvement (say to NP) without also improving the status of the square-root sum and PosSLP problems, but, still, short of that, can we place the discrete FIXP problems say in the Counting Hierarchy (which would mean that probably they are not PSPACE-complete)?

On the lower bound side, could the problems be NP-hard? Would this contradict some accepted conjecture on classical complexity classes? We now give some evidence that an NP-hardness result would also have some interesting consequences, in particular on the relationship between standard Turing machine classes and the real-model classes (and may not be easy to show). It is well known that if any (discrete) total NP search problem were NP-hard, this would imply that NP=co-NP [33, 47]. The discrete versions of FIXP problems however are not known to be (and quite possibly are not) in NP, as we indicated above. The exact computation problems are of course real-valued search problems and hence, strictly speaking, their complexity can be studied only in a real model

of computation. As such, the exact computation problems of FIXP are total real-valued search problems whose complexity lies somewhere between the Blum-Shub-Smale classes $\mathrm{P}_{\mathbb{R}}$ and (total search) $\mathrm{NP}_{\mathbb{R}}$, restricted to rational inputs and constants [5],[3] in a way similar to how PPAD lies between P and (total search) NP. The class of discrete problems solvable in $\mathrm{P}_{\mathbb{R}}$ (with rational inputs and constants) corresponds to P-time in the unit cost rational arithmetic model; it clearly contains P and, as mentioned, it is not known whether this containment is strict. Similarly, we do not know whether the class of discrete problems in $\mathrm{NP}_{\mathbb{R}}$ strictly contains NP. Indeed, equality does not seem to violate any standard complexity assumptions (but of course such an equality would have major consequences, for example it would immediately place the square-root-sum and PosSLP problems in NP). A complete problem for the former class (discrete $\mathrm{NP}_{\mathbb{R}}$) is the decision problem for the *existential theory of reals* (ETR): decide whether there exists a real vector of values x which satisfies a given boolean combination of multi-variate polynomial inequalities of the form $P(x) \geq 0$, where the polynomials $P(x)$ have rational coefficients [5]. This problem is clearly NP-hard, e.g., because one can trivially encode 0-1 integer linear programming in ETR by adding the equalities $x_i = x_i^2$ to ensure every variable x_i is either 0 or 1. On the other hand, ETR is not known to be coNP-hard. Returning to FIXP, by analogy to the results that show total NP search problems can't be NP-hard unless NP=coNP [33, 47] and using essentially the same arguments, we can show the following: *if any FIXP search problem is NP-hard (even via any real-valued search problem reduction computable in $P_{\mathbb{R}}$ restricted to rational constants), then deciding the existential theory of reals (ETR) is coNP-hard, and thus coNP $\subseteq$ $NP_{\mathbb{R}}$.* Whether ETR is coNP-hard is an open problem, and neither answer appears to contradict any accepted complexity conjecture.

Acknowledgement. Research supported by NSF Grant CCF-0728736. It is a pleasure to thank Peter Bro Miltersen for an extended email discussion about PPAD and Nash Equilibria.

References

[1] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. *SIAM J. Computing* 38(5), pp. 1987–2006, 2009.

[2] R. M. Anderson. "Almost" implies "Near". *Trans. Am. Math. Soc.*, 296, pp. 229-237, 1986.

[3] R. A. Becker and S. K. Chakrabarti. Satisficing behavior, Brouwer's fixpoint theorem and Nash equilibrium. *Economic Theory*, 26:63–83, 2005.

[4] A. Bertoni, G. Mauri, N. Sabadini, A characterization of the class of functions computable in polynomial time on random access machines. *Proc. ACM Symp. Th. of Comp*, pp. 168-176, 1981.

[5] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.

[6] A. Borodin, R. Fagin, J. Hopcroft, M. Tompa. Decreasing the Nesting Depth of Expressions Involving Square Roots. *J. Symb. Comput.*, 1(2), pp. 169-188, 1985.

[7] V. Bubelis. On equilibria in finite games. *Intl. J. Game Theory*, 8(2), 65-79, 1979.

[8] J. Canny. Some algebraic and geometric computations in PSPACE. In *STOC*, pp. 460–467, 1988.

[9] X. Chen, X. Deng, and S. H. Teng. Settling the complexity of computing two-player Nash equilibria. *J. of ACM* 56(3), 2009.

[10] X. Chen and X. Deng. Settling the complexity of two-player Nash equilibrium. *FOCS*, pp. 261–272, 2006.

[3] The Blum-Shub-Smale definition of the classes $\mathrm{P}_{\mathbb{R}}$, $\mathrm{NP}_{\mathbb{R}}$ allows for even transcendental constants, but it is natural (and, one could argue, more sensible computationally) to restrict the classes to allow only rational constants; the notations $\mathrm{P}^0_{\mathbb{R}}$, $\mathrm{NP}^0_{\mathbb{R}}$ are used in the literature for these restricted versions that allow only rational constants.

[11] X. Chen, X. Deng, and S. H. Teng. Computing Nash equilibria: approximation and smoothed complexity. In *FOCS*, pp. 603–612, 2006.

[12] A. Condon. The complexity of stochastic games. *Inf. & Comp.*, 96(2):203–224, 1992.

[13] C. Daskalakis, P. Goldberg, and C. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM J. Computing*, 39(1), pp. 195-259, 2009.

[14] C. Daskalakis, A. Fabrikant, and C. Papadimitriou. The game world is flat: The complexity of Nash equilibria in succinct games. *Proc. ICALP*, 2006.

[15] R. S. Datta. Universality of Nash equilibria. *Math. of Oper. Res.*, 28(3), 424-432, 2003.

[16] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic models of Proteins and Nucleic Acids*. Cambridge U. Press, 1999.

[17] B. C. Eaves, H. Scarf. The solution of systems of piecewise linear equations. *Math. of Oper. Res.*, 1(1), 1-27, 1976.

[18] J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. In *LICS*, 2004.

[19] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of non-linear equations. *Journal of ACM* 56(1), 2009.

[20] K. Etessami and M. Yannakakis. Recursive Concurrent Stochastic Games. In *Proc. ICALP*, 2006.

[21] K. Etessami and M. Yannakakis. On the Complexity of Nash Equilibria and Other Fixpoints (Extended Abstract). In *Proc. FOCS*, 2007.

[22] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In *8th ACM STOC*, pp. 10–22, 1976.

[23] J. Geanakoplos. Nash and Walras equilibrium via Brouwer. *Economic Theory*, 21:585–603, 2003.

[24] I. Gilboa and E. Zemel. Nash and correlated equilibria: some complexity considerations. *Games and Economic Behavior*, 1:80–93, 1989.

[25] P. Goldberg and C. Papadimitriou. Reducibility among equilibrium problems. In *Proc. STOC*, 2006.

[26] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 2nd edition, 1993.

[27] F. Gul, D. Pearce, E. Stacchetti. A bound on the proportion of pure strategy equilibria in generic games. *Math. of Oper. Res.* 18, pp. 548-552, 1993.

[28] P. Haccou, P. Jagers, and V. A. Vatutin. *Branching Processes: Variation, Growth, and Extinction of Populations*. Cambridge U. Press, 2005.

[29] T. E. Harris. *The Theory of Branching Processes*. Springer-Verlag, 1963.

[30] J. C. Harsanyi, R. Selten. *A General Theory of Equilibrium Selection in Games*. MIT Press, 1988.

[31] M. D. Hirsch, C. H. Papadimitriou, S. A. Vavasis. Exponential lower bounds for finding Brouwer fixed points. *J. Complexity*, 5, pp. 379-416, 1989.

[32] R. J. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge U. Press, 1985.

[33] D. S. Johnson, C. H. Papadimitriou, M. Yannakakis. How easy is local search? *J. Comp. Sys. Sci.*, 37, pp. 79-100, 1988.

[34] B. Juba. On the hardness of simple stochastic games. Master's thesis, CMU, 2005.

[35] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *STOC*, 2003.

[36] M. Kimmel and D. E. Axelrod. *Branching processes in biology*. Springer, 2002.

[37] D. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*, Addison-Wesley, 1972.

[38] K.-I. Ko. Computational complexity of fixed points and intersection points. *J. Complexity*, 11, pp. 265-292, 1995.

[39] A. N. Kolmogorov and B. A. Sevastyanov. The calculation of final probabilities for branching random processes. *Doklady*, 56:783–786, 1947. (In Russian).

[40] H. W. Kuhn. Simplicial approximation of fixed points. *Proc. Nat. Acad. Sc.* 61, pp. 1238-1242, 1968.

[41] H. W. Kuhn, J. G. MacKinnon. Sandwich methods for finding fixed points. *J. Opt. Th. Appl.*, 17, pp. 189-204, 1975.

[42] C. Lemke, J. Howson. Equilibrium points of bimatrix games. *J. SIAM*, pp. 413-423, 1964.

[43] R. Lipton and E. Markakis. Nash equilibria via polynomial equations. In *LATIN'04*, 2004.

[44] R.J. Lipton, E. Markakis, A. Mehta. Playing large games using simple strategies. *Proc. ACM Conf. Elec. Comm.*, 36-41, 2003.

[45] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[46] A. Mas-Colell, M. D. Whinston, J. R. Green. *Microeconomic Theory*. Oxford Univ. Press, 1995.

[47] N. Megiddo and C. Papadimitriou. On total functions, existence theorems and computational complexity. *Theor. Comput. Sci.*, 81(2):317–324, 1991.

[48] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54:289–295, 1951.

[49] C. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theor. Comp. Sci.*, 4:237–244, 1977.

[50] C. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

[51] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals, parts I-III. *J. Symb. Comp.*, 13(3):255–352, 1992.

[52] H. Scarf. The approximation of fixed points of a continuous mapping. *SIAM J. Appl. Math.*, 15:1328–1343, 1967.

[53] H. Scarf. *The Computation of Economic Equilibria*. Yale University Press, 1973.

[54] A. Schönhage. On the power of random access machines. *Proc. 6th Intl. Coll. Aut. Lang. Prog.*, Springer, pp. 520–529, 1979.

[55] A. Schrijver, *Theory of Linear and Integer Programming*. Wiley, 1986.

[56] L.S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci.*, 39:1095–1100, 1953.

[57] K. Sikorski. *Optimal solution of nonlinear equations*, Oxford Univ. Press, 2001.

[58] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*, Springer-Verlag, 2nd edition, 1993.

[59] P. Tiwari. A problem that is easier to solve on the unit-cost algebraic RAM. *Journal of Complexity*, pp. 393–397, 1992.

[60] H. Uzawa. Walras's existence theorem and Brouwer's fixed point theorem. *Econ. Stud. Q.*, 13:59–62, 1962.

[61] M. Yannakakis. The analysis of local search problems and their heuristics. *STACS*, pp. 298–311, 1990.