Simulating cardinal preferences in Boolean games: A proof technique

Egor Ianovski[a], Luke Ong[a]

[a] *Department of Computer Science, Wolfson Building, Parks Rd, Oxford OX1 3QD, United Kingdom*

Abstract

Boolean games are a succinct representation of strategic games with a logical flavour. While they have proved to be a popular formalism in the multiagent community, a commonly cited shortcoming is their inability to express richer utilities than success or failure. In addition to being a modelling limitation, this parsimony of preference has made proving complexity bounds difficult. We address the second of these issues by demonstrating how cardinal utilities can be simulated via expected utility. This allows us to prove that RationalNash and IrrationalNash are NEXP-hard, and to translate hardness results for IsNash and DValue into the Boolean games framework.

1. Introduction

The formalism of game theory has proven to be a useful lens with which to study various domains of computation. Whether we are talking about the interaction of threads in a program, bots on the internet or cranes in a dockyard, we are talking about strategic behaviour in a competitive setting, which can be modelled as a game. However, the focus on computation brings to the foreground issues of complexity and representation that were only of tangential interest to the founders of the field. One such issue is the representation of a finite strategic game. The standard normal form representation is impractical for all but the smallest of games, and as a result many succinct games have been studied in the literature. Among these are the Boolean games of Harrenstein et al. (2001).

A Boolean game is played between two or more players who assign truth values to propositional variables with the hope of satisfying a formula of propositional logic. The framework presents a succinct representation of a strategic game, as we need only k variables to model 2^k ways of playing the game, but retains some intuition about what players want and how they intend to get it, as opposed to more general succinct frameworks in which a player is in essence a computational black box. It is perhaps for these reasons that Boolean games have proven to be a popular model in the study of multiagent systems.

Given a truth assignment, however, a player's formula is either satisfied or not. He is either perfectly happy, or infinitely depressed. This is obviously

a modelling limitation, and many enrichments of Boolean games exist in the literature to address this, but there is also a subtler issue: it is difficult to prove theorems when we do not have any numbers to play with. Standard game theoretic constructions involve tweaking a player's preferences to prefer one outcome or another, but if the only tweaks we have are 1 and 0, there is a limit to what we can achieve.

In this paper we demonstrate a proof technique that can get around this difficulty. Classical game theory owes many of its fundamental assumptions to the work of von Neumann and Morgenstern (1947). Crucial among these is the players' attitude to uncertainty—von Neumann and Morgenstern demonstrate that any player whose preferences over lotteries are complete, transitive, continuous and independent is an expected utility maximiser. Intuitively, the player is indifferent between getting a utility of ½ for certain and a utility of 1 with a probability of ½. Since it makes no sense for a formula of propositional logic to only be half true, a Boolean game cannot award a player a utility of ½. However, there is no such difficulty with a formula being true with a probability of ½, and the expected utility in both cases is the same.

We demonstrate that given any rational v in $[0, 1]$ it is in fact possible to construct, in polynomial time, a two-player, zero-sum Boolean game such that the probability of Player One's formula being true in equilibrium is precisely v. By using these games as gadgets in a larger construction it is possible to construct a game where the player behaves *as if* his preferences allowed for gradation of opinion, even though in any given outcome of the game his utility is either 1 or 0. To demonstrate the fecundity of this approach, we obtain previously unknown complexity results for four decision problems concerning Boolean games.

1.1. Our contribution

We demonstrate how, given a rational $v \in [0, 1]$, it is possible to construct a two-player, zero-sum game with value v, in time polynomial in the bitlength of v. We then demonstrate how such games can be used to prove complexity results for Boolean games, either by mimicking standard game theoretic arguments, or directly translating proofs from other frameworks into ours. The problems we tackle using these techniques are RationalNash, IrrationalNash, IsNash and DValue.

RationalNash and IrrationalNash are the problems asking whether a game has an equilibrium that uses only rational strategy weights, and at least one irrational strategy weight respectively. These problems were first studied by Bilò and Mavronicolas (2011) in the context of games in normal form and later in the context of win-lose games in normal form (Bilò and Mavronicolas, 2012). The problems are shown to be NP-hard for games with three players. We demonstrate that in the context of Boolean games, RationalNash is NEXP-hard for games with three players, and IrrationalNash is NEXP-complete for games with two.

IsNash is the problem of deciding whether a given profile (represented by its support) is in fact an equilibrium. For games in normal form, this problem

is in P. In the context of succinct games, Schoenebeck and Vadhan (2012) show that the problem is coNP$^{\#\mathrm{P}}$-hard. In this paper we show that it is possible to replicate the proof of Schoenebeck and Vadhan in the context of Boolean games by carefully using gadget games and logical coding to mimic the arithmetic operations in the original proof, and hence show that IsNash is coNP$^{\#\mathrm{P}}$-hard for Boolean games with an unbounded number of players.

DValue is the problem of determining whether the value of a two-player zero-sum game exceeds a given threshold. It was known to be as hard as linear programming before the exact complexity of linear programming was established (Dantzig, 1951).[1] Koller et al. (1994) show that the problem remains in polynomial time even if the game is represented in the more succinct extensive form, and later Feigenbaum et al. (1995) show that the problem is EXP-complete for a succinct representation of the (already succinct) extensive form. We show that our gadgets allow us to translate this proof into the context of Boolean games, showing the problem is EXP-complete.

2. Related work

2.1. Boolean games

Boolean games were first introduced by Harrenstein et al. (2001) (see also Harrenstein, 2004) as two-player, zero-sum games that mirror the Lindenbaum algebra of propositional logic. They were reformulated by Bonzon et al. (2006) into the multiplayer, succinct games that are familiar today. The complexity of Boolean games was an area of much research activity (e.g. Dunne and van der Hoek, 2004; Bonzon et al., 2009b; Dunne and Wooldridge, 2012; Sauro and Villata, 2013), but these concerned themselves exclusively with play in pure strategies; before it was shown that ∃GuaranteeNash is NEXP-hard (Ianovski and Ong, 2014), nothing was known about the complexity of mixed equilibria. This work extends the techniques used to prove the hardness of ∃GuaranteeNash to obtain fresh complexity results.

Boolean games proved to be a popular framework for modelling aspects of multiagent systems such as incentive design (Wooldridge, 2012; Endriss et al., 2011; Levit et al., 2013; Turrini, 2013; Galafassi and Bazzan, 2013; Harrenstein et al., 2014), coalition formation (Dunne et al., 2008; Bonzon et al., 2012; Popovici and Dobre, 2012), delegation (Kraus and Wooldridge, 2012) and communication (Grant et al., 2011). However, the inability of agents in Boolean games to express preferences other than $\{0, 1\}$ was a hindrance in this setting, and a number of frameworks sought to enrich the players' preferences by description logics (Lukasiewicz and Ragone, 2009), CP-nets, (Bonzon et al., 2009c,a), modal logics (Gutierrez et al., 2013; Ågotnes et al., 2013; Harrenstein et al., 2015), fuzzy logics (Marchioni and Wooldridge, 2014), and formula weights (Bilò, 2007; Mavronicolas et al., 2007). While this line of work may

[1] Of course, we now know this to be complete for polynomial time (Khachiyan, 1980).

appear to resemble the main thrust of this paper, we stress the crucial difference: the above works extended the Boolean games framework into something different. This creates a more versatile modelling tool, but complexity results for their setting do not necessarily hold for Boolean games proper. In our case, we present purely a proof technique—the simulation of cardinal preferences in our setting allows us to prove complexity results for Boolean games, but they would not be a sensible way to model such preferences in an actual system.

2.2. Win-lose games

Boolean games belong to a natural class of games where a player's utility is restricted to 0 or 1: the win-lose games. While this may seem like a serious restriction—after all, even textbook games like Prisoner's Dilemma need a four-way distinction of preferences in order to tell their story—it appears to be that from a complexity-theoretic perspective, win-lose games are every bit as hard as the general case. Even before the complexity of FindNash was established (Daskalakis and Papadimitriou, 2005), it was already known that it would be no easier in the case of binary preferences (Abbott et al., 2005), nor does the problem become easier under an approximation scheme (Chen and Teng, 2007). In the case of decision problems, Codenotti and Štefankovič (2005) and Bilò and Mavronicolas (2012) demonstrate that a variety of natural decision problems, including the ones shown to be hard for games with unrestricted utilities by Gilboa and Zemel (1989) and Conitzer and Sandholm (2008), are NP or coNP-hard for win-lose games in normal form.

What is relevant to us is that Bilò and Mavronicolas (2012) show that the problem RationalNash is NP-hard for three-player win-lose games in normal form, and in this paper we show that the problem is NEXP-hard for three player Boolean games. The authors earlier showed that the problem IrrationalNash is NP-hard for three-player games in normal form (Bilò and Mavronicolas, 2011), but to our knowledge they were not able to replicate this result in the win-lose case. In this paper we show that this problem is NEXP-complete for two-player Boolean games; the proof also shows that the problem is NP-complete for two-player games in normal form (but does not generalise to win-lose games in normal form).

2.3. Succinct games

Succinct representations of games are as old as the discipline itself—the extensive form, or game-tree representation, is typically introduced right after the normal form in a textbook treatment—and as such we make no attempt to cover the field in depth, but only those directly relevant to Boolean games. By this we mean that we are not interested in representations that achieve succinctness by exploiting the *game-theoretic* aspects of the game (e.g. sparsity of payoffs, symmetry of strategies, weak dependency between players, etc.), but rather the *computational* aspects, i.e. the ability of the game to be represented as a small circuit, machine, or, indeed, a Boolean formula.

An extremely general form of a game is presented by Feigenbaum et al. (1995). A game consists of two players and a referee. The referee is free to ask

the players questions in any way he wishes, the answers to which he may then choose to share with the other player, or provide them with any other form of information before asking another question. When the referee is satisfied, he declares the payoffs and the game ends. There is no bound on the computational power of the players, but the referee must run in polynomial time. Among other results, the authors show that the problem DValue is EXP-complete for these games.

This game form was reformulated by Fortnow et al. (2005) into what would later be called a circuit game.[2] This framework is very similar to that of the Boolean game: each player sets the value to their set of input gates which, when evaluated on the Boolean circuit each player possesses, tells them what utility they derive from the strategy profile. The approximation complexity of zero-sum circuit games is studied by Fortnow et al. (2005), whereas Schoenebeck and Vadhan (2012) present results regarding a number of decision problems, including ∃GuaranteeNash and IsNash, for circuit games.

It is also worth mentioning the Turing machine games of Àlvarez et al. (2005b) (also Àlvarez et al., 2005a, 2011)—in which players each set some symbols in the input word of a Turing machine, which then computes the payoffs. The authors study three variants of these games; what they refer to as the *implicit* representation is, for all means and purposes, equivalent to a circuit game.

Boolean games form a very specific subclass of all these games, privileged by the fact that logical formulae tend to be more intuitively appealing than machine transitions, and hindered by decreased expressive power (a circuit can output a binary integer, a formula only one bit) and, assuming the NC hierarchy does not collapse, a loss of succinctness. As such, Boolean games inherit complexity upper bounds from, and provide lower bounds to, the cited game representations.

Our results in Section 4.2 are translations of Theorem 4.6 of Feigenbaum et al. (1995) and Theorem 5.5 of Schoenebeck and Vadhan (2012) into the setting of Boolean games.

3. Preliminaries

3.1. Strategic and Boolean games

While in the literature it is common to present Boolean games as games *sui generis*, it is profitable to view the issue through the wider lens of algorithmic game theory if one would ask and interpret questions of complexity. As such

[2]Strictly speaking, the reformulated version is *not* equivalent to the game of Feigenbaum et al. (1995): the games of Fortnow et al. (2005) are a succinct representation of the normal form, and the games of Feigenbaum et al. (1995) of the extensive form. As such, the one offers exponential succinctness, the other potentially double-exponential succinctness. However, it turns out that the complexity of computing the value is the same for games in normal and extensive forms (Koller et al., 1994), so this does not affect the complexity of the results studied by the mentioned authors.

our first definition pertains to a more well-known class of games, typically the first one would encounter in an undergraduate textbook.

Definition 3.1. A *strategic game* is a triple $(N, \{ S_1, \ldots, S_n \}, \{ u_1, \ldots, u_n \})$. N is a finite set of *players*, of cardinality n. S_i is a finite set of Player i's *pure strategies*. An n-tuple of pure strategies, i.e. a member of $\mathcal{S} = S_1 \times \cdots \times S_n$ is called a *pure-strategy profile*. The function $u_i : \mathcal{S} \to \mathbb{R}$ is Player i's *utility function*. If every utility function maps to $\{ 0, 1 \}$, then the game is called a *win-lose game*.

A strategic game is called *zero-sum* just if there exists a $c \in \mathbb{R}$ such that for every $\boldsymbol{s} \in \mathcal{S}$:[3]

$$\sum_{i \in N} u_i(\boldsymbol{s}) = c.$$

■

Example 3.2. Battle of the Sexes is played by two players coordinating on a venue for a date. The choices are boxing and ballet. Player One prefers ballet and Player Two prefers boxing, but both players prefer successful coordination to choosing different venues.

This game could be represented by giving the players the strategies $S_1 = \{ Box_1, Bal_1 \}$ and $S_2 = \{ Box_2, Bal_2 \}$. Their utility functions would be given by setting $u_1(Box_1, Bal_2) = u_1(Bal_1, Box_2) = 0$, $u_1(Bal_1, Bal_2) = 2$ and $u_1(Box_1, Box_2) = 1$ for Player One, $u_2(Box_1, Bal_2) = u_2(Bal_1, Box_2) = 0$, $u_2(Bal_1, Bal_2) = 1$ and $u_2(Box_1, Box_2) = 2$ for Player Two.

Matching Pennies is played by having two players reveal a coin heads or tails up. Player Two wins if the coins have the same orientation, Player One if the orientation differs.

In the given formalism this game is $S_1 = \{ H_1, T_1 \}$, $S_2 = \{ H_2, T_2 \}$, $u_1(H_1, T_2) = u_1(T_1, H_2) = 1$, $u_1(H_1, H_2) = u_1(T_1, T_2) = 0$, $u_2(H_1, H_2) = u_2(T_1, T_2) = 1$ and $u_2(H_1, T_2) = u_2(T_1, H_2) = 0$.

Matching Pennies is a win-lose game, while Battle of the Sexes requires a tripartite distinction of preferences.

This explicit representation of a strategic game, obtained by listing the graph of the utility functions, is often referred to as the *normal form* and in the two-player case it can be conveniently visualised as a table:

	Box_2	Bal_2
Box_1	$(1, 2)$	$(0, 0)$
Bal_1	$(0, 0)$	$(2, 1)$

Battle of the Sexes

	H_2	T_2
H_1	$(0, 1)$	$(1, 0)$
T_1	$(1, 0)$	$(0, 1)$

Matching Pennies

■

[3] This is of course more correctly called a constant-sum game, but to us the difference is immaterial.

In the above we draw a distinction between a strategic game and its representation via the normal form. This is by no means standard in the literature (though essentially the same approach is taken by Schoenebeck and Vadhan (2012) and Àlvarez et al. (2005b)); the terms *strategic game*, *normal-form game* and, in the two-player case, *bimatrix game* tend to be used interchangeably. We however argue that the distinction is important for algorithmic enquiry. The definition of a strategic game relies on the concept of a set, which is an abstract, mathematical notion. One cannot feed a set into an algorithm; a concrete data structure needs to be used, be it a list (ordered or unordered), a hash table, a bitmap, or something else. Naturally, the specific choice of data structure can have consequences for the complexity of the problems studied. Given that we are working with polynomial time reductions in this paper, we do not need to go into such fine details, and a list-based normal form will work as well as anything else, but it is important not to conflate the normal form with the abstract game it is meant to represent.

Games are intuitively models of behaviour, and as such are associated with normative or positive solution concepts, intended to represent the probable, desirable, or rational outcomes of a game. The most common of such concepts for a strategic game is the Nash equilibrium, which is a strategy profile from which no player would opt to deviate unilaterally. In other words, the utility the player is currently getting is at least as high as in any alternative profile that differs only by that player's own choice of strategy.

Definition 3.3. A pure-strategy profile $\boldsymbol{s}$ is called a *pure-strategy equilibrium* just if for each $i \in N$, for all $s' \in S_i$:

$$u_i(\boldsymbol{s}) \geq u_i(\boldsymbol{s_{-i}}(s')),$$

where $\boldsymbol{s_{-i}}(s')$ denotes the profile obtained by replacing Player i's strategy in $\boldsymbol{s}$ with s'. ■

Example 3.4. Battle of the Sexes has two pure-strategy equilibria: either both players go to ballet or to boxing. One player will be getting a utility of 2 the other of 1, but even the player that is getting a utility of 1 would rather stay at the venue than deviate and get a utility of 0.

Matching Pennies has no pure-strategy equilibria. No matter what profile we choose, one of the players would rather switch the orientation of their coin.

In the tabular representation the pure-strategy equilibria are precisely those cells where Player One's utility is maximal in the column, and Player Two's is maximal in the row, as the reader can verify above. ■

The fact that even games as simple as Matching Pennies can fail to have a pure-strategy equilibrium motivates us to consider broader solution concepts. It helps to consider how one would actually play a game like Matching Pennies (e.g. Rock-Paper-Scissors) were one to find oneself stuck at a horrendous dinner party. Displaying a strong preference for either heads or tails would allow the opponent to take advantage by playing the appropriate counter. On the other

hand, picking one of rock, paper and scissors at random will give the opponent no opening to exploit, and is the optimal way to play.[4] This is the concept of a mixed strategy.

Definition 3.5. Let $\mathcal{P}(S_i)$ denote the space of probability distributions over S_i. A *mixed strategy* for Player i is a member of $\mathcal{P}(S_i)$. The weight assigned to a pure strategy s by a mixed strategy σ, or $P(s \mid \sigma)$, is called the *strategy weight* of s.

An n-tuple of mixed strategies, $\boldsymbol{\sigma} \in \mathcal{P}(\mathcal{S})$, is called a *mixed-strategy profile*. We extend Player i's utility function to the space of mixed-strategy profiles on the principle of expected utility. That is:

$$u_i(\boldsymbol{\sigma}) = \sum_{\boldsymbol{s} \in \mathcal{S}} u_i(\boldsymbol{s}) P(\boldsymbol{s} \mid \boldsymbol{\sigma}).$$

A mixed-strategy profile is called a *mixed-strategy equilibrium* just if for all $s' \in S_i$:

$$u_i(\boldsymbol{\sigma}) \geq u_i(\boldsymbol{\sigma_{-i}}(s')).$$

The *support* of a mixed strategy σ is the set of pure strategies that do not receive a weight of zero in σ. ■

Clearly a pure equilibrium is just a specific type of mixed equilibrium, so when we say "equilibrium" without qualifications we mean a mixed equilibrium.

The reader will also note that in the definition above we define a mixed-strategy equilibrium as a profile that is robust to any deviation with a *pure* strategy, and do not consider deviations with mixed strategies. There is no generality lost here—if we are considering a unilateral deviation by Player i we can fix the strategy choice of other players, leaving Player i with a linear function to maximise. Such a function will attain its maxima at the extreme points, which are precisely the pure strategies.

Example 3.6. Matching Pennies has a unique equilibrium where Player One randomises equally between H_1 and T_1, and Player Two randomises equally between H_2 and T_2. The payoff for either player is $1/2$. This is also the payoff Player One would get by playing H_1, as Player Two plays T_2 with probability $1/2$, or for playing T_1, as Player Two plays H_2 with probability $1/2$. Player One is indifferent between any deviation, and, mutatis mutandis, so is Player Two.

Battle of the Sexes has an additional equilibrium where Player One plays Bal_1 with probability $2/3$ and Box_1 with probability $1/3$, whilst Player Two plays Bal_2 with probability $1/3$ and Box_2 with probability $2/3$. The utility of either player is $2/3$—the utility Player One would get by playing Bal_1 with probability

[4] Between human players, of course, the game of Rock-Paper-Scissors contains a very strong psychological factor. Combined with the fact that humans are typically very bad at randomising, this allows for the existence of the "skilled" Rock-Paper-Scissors player who can consistently win more games than our model would predict. However, it remains true that even such a player would be stumped if pitted against a computer opponent playing pseudo-randomly.

1 (1/3 chance of getting 2), or Box_1 with probability 1 (2/3 chance of getting 1). ■

Two foundational theorems of game theory are of interest to us here. The first, due to Nash, tells us that mixed equilibria exist.

Theorem 3.7 (Nash, 1951)**.** *Every strategic game has an equilibrium in mixed strategies.*

The second (albeit first chronologically), due to von Neumann, tells us that equilibria in two-player zero-sum games have a special property.

Theorem 3.8 (von Neumann, 1928)**.** *Given a two-player zero-sum game there exists a v such that Player One gets a utility of v in every equilibrium. This v is called the* value *of the game.*

The normal form representation studied above has many pedagogical benefits—being able to depict the entire strategy space on a single diagram fosters an intuitive concept of strategy, best response and equilibrium. However, the representation size does not scale well: the graph of Player i's utility function is of order $O(|\mathcal{S}|^n)$—not only is this exponential in the number of players, but the polynomial dependence on the number of strategies alone is untenable in all but the most simple of games (consider checkers). As a result many concise representations of games have been studied in the literature. In this work, we are interested in one in particular: Boolean games.

A Boolean game operates by partitioning a set of propositional variables among a set of players. A player may assign *true* or *false* values to the variables under his control in the hope of satisfying his *goal formula*, a formula of propositional logic. As his goal formula may depend on variables outside of his control he cannot achieve this unilaterally, and has to reason strategically to choose the best possible truth assignment.

Definition 3.9 (Harrenstein et al., 2001; Bonzon et al., 2006)**.** A *Boolean game* is a triple $(N, \{\Phi_1, \ldots, \Phi_n\}, \{\gamma_1, \ldots, \gamma_n\})$. The Φ_i are pairwise disjoint sets of propositional variables and each γ_i is a formula of propositional logic defined over $\biguplus_{i \in N} \Phi_i$. These components form a representation of a win-lose strategic game—Player i's set of pure strategies is the truth assignments to Φ_i, i.e. $S_i = 2^{\Phi_i}$, and Player i's utility function is:

$$u_i(\boldsymbol{\nu}) = \begin{cases} 1 \text{ if } \boldsymbol{\nu} \vDash \gamma_i, \\ 0 \text{ otherwise.} \end{cases}$$

■

Example 3.10. Matching Pennies can be represented as a Boolean game in

the following fashion:

$$\begin{aligned}\Phi_1 &= \{\, p \,\},\\ \Phi_2 &= \{\, q \,\},\\ \gamma_1 &= \neg(p \leftrightarrow q),\\ \gamma_2 &= p \leftrightarrow q.\end{aligned}$$

Battle of the Sexes cannot be represented as a Boolean game because the payoffs are not all 0 and 1. ■

A Boolean game can potentially achieve exponential succinctness—after all, we need only k variables to represent 2^k strategies. However this comes at the cost of some of the lucidity of a game in normal form—even determining whether a player ought to bother playing the game (whether γ_i is satisfiable) is NP-complete.

It is important to recall that propositional logic is expressively complete: every function that maps a string of bits to a single bit can be expressed as a Boolean formula. This allows us to classify exactly the class of games that have a Boolean representation:

Fact 3.11. *A strategic game has a Boolean representation if and only if:*

1. *Every utility function maps to* $\{\, 0, 1 \,\}$. *I.e., the game is win-lose.*
2. *The cardinality of every player's strategy set is a power of two.*

Proof. If the players have $2^{k_1}, \ldots, 2^{k_n}$ strategies respectively then we could label each of Player i's strategies with a truth assignment to the variables $\{\, p^i_{k_1}, \ldots, p^i_{k_i} \,\}$. These are the variables Player i will control in the Boolean game.

Player i's utility function u_i thus maps n truth assignments to $\{\, 0, 1 \,\}$. By the completeness of propositional logic, there exists a formula γ_i that is satisfied by a truth assignment $\nu_1, \ldots, \nu_n$ if and only if $u_i(\nu_1, \ldots, \nu_n) = 1$. □

Given a G that satisfies the hypotheses above, we do not need to construct a Boolean game representing G directly. We can simply assume one exists. We have no guarantee that this representation will be concise, however, and in the worst case the size of the Boolean game representing G will be exponential in the number of variables. For this reason, when dealing with families of games of arbitrary size we will still have to give the construction of the Boolean game explicitly and verify that it satisfies the required space constraints. However, if G refers to a game of fixed size then for our purposes it is sufficient to know that a Boolean representation exists; we do not need to construct it explicitly because even in the worst case $O(2^c) = O(c)$.

3.2. Decision problems and rational equilibria

It now pays to introduce a specific kind of equilibrium, that is of natural interest from a computational perspective.

Definition 3.12. An equilibrium $\boldsymbol{\sigma}$ is *rational* if every strategy weight in $\boldsymbol{\sigma}$ is rational. It is *irrational* if at least one strategy weight is not. ■

Rational numbers are convenient as they allow us to reason algorithmically about a wide range of problems without delving into the representational issues concerning algebraic or computable reals. It is fortuitous, then, that a consequence of the linear programming characterisation of two-player games establishes that these have rational equilibria.

Proposition 3.13 (Cottle and Dantzig, 1968)**.** *Every two-player strategic game has a rational equilibrium, and the size of this equilibrium is polynomial in the size of the normal form.*

Proof. The argument below is based on the support enumeration algorithm (Kaplan and Dickhaut, 1991; von Stengel, 2007).

Let $X \subseteq S_1$ and $Y \subseteq S_2$. We will show that we can reduce the problem of verifying whether the game has an equilibrium with support X for Player One and Y for Player Two to an instance of linear programming.

A necessary condition for equilibrium is that a player be indifferent between every strategy in his support. For Player One this condition translates to:

$$\begin{aligned} \sum_{s_j \in Y} a_{i,j} y_j &= \alpha, \text{for } s_i \in X, \\ \sum_{i \in Y} y_i &= 1, \\ \boldsymbol{y} &\geq 0. \end{aligned}$$

α is the (as yet unknown) utility Player One derives from the potential equilibrium, and the y_i variables are the strategy weights of Player Two.

For Player Two the same condition is the following:

$$\begin{aligned} \sum_{s_i \in X} b_{i,j} x_i &= \beta, \text{for } s_j \in Y, \\ \sum_{i \in X} x_i &= 1, \\ \boldsymbol{x} &\geq 0. \end{aligned}$$

The variables x_i being the strategy weights of Player One.

However, this is not sufficient – a player may well be indifferent between the strategies in his support, but that does not mean he does not have a profitable deviation with a strategy outside of it. We need to check that α and β are at

least as big as what the players can get with any pure strategy:

$$\begin{aligned}
\sum_{s_j \in Y} a_{1,j} y_j \leq & \alpha, \\
& \vdots \\
\sum_{s_j \in Y} a_{k,j} y_j \leq & \alpha, \\
\sum_{s_i \in X} b_{i,1} x_i \leq & \beta, \\
& \vdots \\
\sum_{s_i \in X} b_{i,q} x_i \leq & \beta.
\end{aligned}$$

This gives us a linear programming instance (with a trivial objective function, over the variables $\boldsymbol{x}, \boldsymbol{y}, \alpha, \beta$), and we can produce a solution, or verify that none exists, in polynomial time. Since the equilibrium is the output of a polynomial time algorithm, it must be polynomial in size. □

Unfortunately, the fun stops at $n = 2$.

Proposition 3.14 (Nash, 1951)**.** *There exist three-player games with rational payoffs that have only irrational equilibria. This is true even of win-lose games (Bilò and Mavronicolas, 2012).*

In a concurrent work our focus is on two-player games precisely because we wish to avoid the complications brought about by irrational equilibria, and thereby establish tight complexity results for two-player games. In this paper, our focus is different: rational equilibria are not a means to an end, and irrational equilibria are not obstacles. Rather, both are subjects of our complexity investigation.

The decision problems of interest to us in this work are displayed in Figure 3.1. ∃GuaranteeNash will serve as the basis of our reductions. In the case of games in normal form, ∃GuaranteeNash is NP-complete (Gilboa and Zemel, 1989). In the setting of Boolean games, the multiplayer variant has been shown to be NEXP-hard[5] by Ianovski and Ong (2014), and the two-player variant NEXP-complete in a concurrent work.

Theorem 3.15. ∃GuaranteeNash *for two-player Boolean games is* NEXP*-complete.*[6]

[5]The class of problems solveable in nondeterministic exponential time, $\bigcup_{k \in \mathbb{N}} \text{NTIME}(2^{n^k})$.

[6]Strictly speaking, the definition of ∃GuaranteeNash here is slightly different (the original used a vector of payoffs for every player, rather than v for Player One), but the theorem still holds.

∃GuaranteeNash	
Input:	A representation of a game G and a rational v.
Output:	YES if there exists an equilibrium $\boldsymbol{\sigma}$ of G such that $u_1(\boldsymbol{\sigma}) \geq v$.

RationalNash	
Input:	A representation of a game G.
Output:	YES if G has a rational equilibrium, NO otherwise.

IrrationalNash	
Input:	A representation of a game G.
Output:	YES if G has an irrational equilibrium, NO otherwise.

IsNash	
Input:	A representation of a game G and (the non-zero entries of) a strategy profile $\boldsymbol{\sigma}$.
Output:	YES if $\boldsymbol{\sigma}$ is an equilibrium of G, NO otherwise.

DValue	
Input:	A representation of a two-player, zero-sum game G and some $v \in \mathbb{Q}$.
Output:	YES if the value of G is at least v, NO otherwise.

Figure 3.1: Decision problems.

Our main results are for RationalNash and IrrationalNash. Given Proposition 3.13, RationalNash trivialises for the two-player case; our result is thus about the first non-trivial case, that of three players. Bilò and Mavronicolas (2011) claim that IrrationalNash also trivialises in the case of two players, but that is only true of non-degenerate games.[7] Seeing how the problem of determining whether a game is degenerate appears to be a difficult one (Campbell and Knight, 2015), we do not feel justified in restricting our attention to non-degenerate games only. Our result concerns two players, and is based on the following observation.

Fact 3.16. *A two-player game has an irrational equilibrium if and only if it has an infinite number of equilibria.*

Proof. This is a consequence of the linear programming characterisation of two-player games. A linear system has either zero, one, or a continuum of equilibria. If the linear system corresponding to the support pair (X, Y) (as constructed in Proposition 3.13) has a unique solution, the equilibrium must be rational. If it has an infinite number of solutions, there is a continuum of equilibria, and hence an irrational equilibrium. □

3.3. Encoding integers in logic

The goal of this section is to introduce shorthand and notation that will allow us to discuss integers and arithmetic in the language of propositional logic. This will ultimately allow us to interpret a player's strategy as a choice of integers, and a goal formula as an arithmetical constraint.

Our constructions will often involve sequences of propositional variables, so we will use the notation $\overline{p_m}$ to mean $p_1, \ldots, p_m$. Sequences differ from sets in that they have an order, which allows us to interpret a truth assignment to a sequence as an integer. We do this in the standard way—a truth assignment that sets p_i to *true* defines a binary integer where the ith most significant bit is 1.

Definition 3.17. Let $\overline{p_m}$ be a sequence of m propositional variables, and ν a truth assignment to $\overline{p_m}$. We use $[\![\overline{p_m}]\!]_\nu$ to denote the numeric value associated with ν via its assignment to $\overline{p_m}$. That is:[8]

$$[\![\overline{p_m}]\!]_\nu = \sum_{i=1}^{m} 2^{m-i}(\nu \vDash p_i).$$

If ν is clear from context, we just write $[\![\overline{p_m}]\!]$. ■

This is quite sufficient to interpret any pure strategy in a Boolean game as a choice of integer. To test whether such an integer satisfies an arithmetical

[7] A game is non-degenerate if for every mixed strategy with a support of size k, there are at most k best pure-strategy responses.

[8] $(\nu \vDash p_i)$ is 1 if $\nu \vDash p_i$, 0 otherwise.

constraint we need to introduce formulae that verify operations of (finite) arithmetic. Throughout the following arguments the reader should bear in mind that what we are trying to establish is that these formulae can be constructed in time polynomial in the bit length of the integers under consideration—that such formulae exist at all follows trivially from the expressive completeness of propositional logic.

The following are exercises in propositional logic, and we relegate the proofs to the appendix.

Lemma 3.18. *Let* $\boldsymbol{Equal}(\overline{p_m};\overline{q_m})$ *denote a term that is true if and only if* $[\![\overline{p_m}]\!] = [\![\overline{q_m}]\!]$*. I.e.,* $\nu \vDash \boldsymbol{Equal}(\overline{p_m};\overline{q_m})$ *if and only if* $[\![\overline{p_m}]\!]_\nu = [\![\overline{q_m}]\!]_\nu$*.*

We can construct $\boldsymbol{Equal}(\overline{p_m};\overline{q_m})$ *in time linear in* m*.*

Lemma 3.19. *Let* $\boldsymbol{Succ}(\overline{p_m};\overline{q_m})$ *denote a term that is true if and only if* $[\![\overline{p_m}]\!] + 1 = [\![\overline{q_m}]\!]$*.*

We can construct $\boldsymbol{Succ}(\overline{p_m};\overline{q_m})$ *in time quadratic in* m*.*

Lemma 3.20. *Let* $\boldsymbol{Less}(\overline{p_m};\overline{q_m})$ *denote a term that is true under* ν *if and only if* $[\![\overline{p_m}]\!]_\nu < [\![\overline{q_m}]\!]_\nu$*.*

We can construct $\boldsymbol{Less}(\overline{p_m};\overline{q_m})$ *in time cubic in* m*.*

Corollary 3.21. *Let* $\boldsymbol{LessEq}(\overline{p_m};\overline{q_m})$ *denote a term that is true under* ν *if and only if* $[\![\overline{p_m}]\!]_\nu \leq [\![\overline{q_m}]\!]_\nu$*.*

We can construct $\boldsymbol{LessEq}(\overline{p_m};\overline{q_m})$ *in time cubic in* m*.*

Lemma 3.22. *Let* $\boldsymbol{Add}(\overline{p_m};\overline{q_m};\overline{r_m})$ *denote a term that is true if and only if* $[\![\overline{p_m}]\!] + [\![\overline{q_m}]\!] = [\![\overline{r_m}]\!]$*.*

$\boldsymbol{Add}(\overline{p_m};\overline{q_m};\overline{r_m})$ *can be replaced by a formula of propositional logic cubic in* m*.*

Lemma 3.23. *Let* $\boldsymbol{Sub}(\overline{p_m};\overline{q_m};\overline{r_m})$ *denote a term that is true if and only if* $[\![\overline{p_m}]\!] - [\![\overline{q_m}]\!] = [\![\overline{r_m}]\!]$*.*

$\boldsymbol{Sub}(\overline{p_m};\overline{q_m};\overline{r_m})$ *can be replaced by a formula of propositional logic cubic in* m*.*

At times in lieu of testing $[\![\overline{p_m}]\!]$ for one of these three relations against $[\![\overline{q_m}]\!]$, we may wish to test, for example, whether $[\![\overline{p_m}]\!] < 3$. In other words, we would like to have access to constants as well as variables, and that is the purpose of the following definition:

Definition 3.24. Let j be a binary integer of length m. We use $\ulcorner \boldsymbol{j} \urcorner$ to denote a sequence of the logical constants $\top$ and $\bot$, the ith element of which is $\top$ if and only if the ith most significant bit of j is 1.

The intended use of $\ulcorner \boldsymbol{j} \urcorner$ is as an argument to the parametrised formulae defined so far. For example, we interpret $\boldsymbol{Less}(\overline{p_m};\ulcorner \boldsymbol{j} \urcorner)$ in the sense of Lemma 3.20, except every instance of q_i is replaced with $\top$ or $\bot$ depending on whether the ith bit of j is a 1 or 0. ■

Finally, we want some formulae to assert that a certain number of their arguments are true.

Lemma 3.25. *Let* $\boldsymbol{OneOf}(\overline{p_m})$ *and* $\boldsymbol{NoneOf}(\overline{p_m})$ *denote terms that are true just if exactly 1 and 0 respectively of the* p_i *variables are true. These terms can be constructed in time polynomial (quadratic and linear respectively) in* m*.*

3.4. Games of any value

The reader will note that while the payoff for Player One in any *pure* profile in a Boolean game need be 1 or 0, there is no such restriction on *mixed* profiles—after all, the unique mixed equilibrium of Matching Pennies gives either player $1/2$. This will provide us with a convenient family of gadget games to facilitate proofs.

In a win-lose game, $v \in [0,1]_{\mathbb{Q}}$ is the probability of Player One winning the game. Given $v = a/b$ we can construct a game in which Player One chooses an interval (possibly looping around the edges) of length a over $[0, b-1]_{\mathbb{N}}$, and Player Two chooses a single integer from the same range. If a and b are coprime, i.e. the fraction a/b is maximally reduced, we can show that this game has a unique equilibrium—the profile where Player One randomises equally over every interval and Player Two over every integer—the value of which is clearly $a/b = v$. The vocabulary we have developed in Section 3.3 will allow us to express this as a Boolean game.

Lemma 3.26. *For* $v \in [0,1]_{\mathbb{Q}}$ *we can construct, in time polynomial in* $|v|$*, a two-player, zero-sum Boolean game with value* v *and a unique equilibrium.*

We shall refer to this game as $\mathfrak{G}(v)$*.*

Proof. Let $v = a/b$. We insist that a, b are coprime.

Consider the game where Player One selects two numbers $c_1, c_2 \in [0, b-1]_{\mathbb{N}}$ with the property that $c_2 - c_1 \equiv a - 1 \mod b$ (the start and end points of an interval of length a). Player Two selects $d \in [0, b-1]_{\mathbb{N}}$. The game is won by Player One if $c_2 \geq d \geq c_1$ (Player Two's choice is in a non-looping interval), $d \geq c_1 > c_2$ (Player Two's choice is in a looping interval left of $b-1$) or $c_1 > c_2 \geq d$ (Player Two's choice is in a looping interval right of 0).

To give this game a Boolean rendition we need the following variables:

$$\Phi_1 = \{p_1, \ldots, p_m, q_1, \ldots, q_m, s_1, \ldots, s_m, t_1, \ldots, t_m\},$$
$$\Phi_2 = \{r_1, \ldots, r_m\}.$$

The interpretation is that $[\![\overline{p_m}]\!] = c_1, [\![\overline{q_m}]\!] = c_2$ and $[\![\overline{r_m}]\!] = d$. The s and t variables come in to play if Player One wishes to play a looping interval, in which case $[\![\overline{s_m}]\!]$ is the distance between 0 and c_2, while $[\![\overline{t_m}]\!]$ is the distance between c_1 and $b-1$. These variables are added to give us a way to check that if Player One plays a looping interval, its length is still a.

Player One's goal formula is the following:

$$\begin{aligned}\gamma_1 =&\big(\boldsymbol{Sub}(\overline{q_m},\overline{p_m},\ulcorner \boldsymbol{a-1}\urcorner)\wedge \boldsymbol{LessEq}(\overline{q_m},\ulcorner \boldsymbol{b-1}\urcorner)\\ &\wedge \boldsymbol{LessEq}(\overline{r_m},\overline{q_m})\wedge \boldsymbol{LessEq}(\overline{p_m},\overline{r_m})\wedge \boldsymbol{Equal}(\overline{s_m},\ulcorner \boldsymbol{0}\urcorner)\\ &\wedge \boldsymbol{Equal}(\overline{t_m},\ulcorner \boldsymbol{0}\urcorner)\big)\\ \vee&\Big(\boldsymbol{Add}(\overline{s_m},\overline{t_m},\ulcorner \boldsymbol{a-1}\urcorner)\wedge \boldsymbol{Sub}(\overline{q_m},\ulcorner \boldsymbol{0}\urcorner,\overline{s_m})\\ &\wedge \boldsymbol{Sub}(\ulcorner \boldsymbol{b-1}\urcorner,\overline{p_m},\overline{t_m})\wedge \big(\boldsymbol{LessEq}(\overline{r_m},\overline{q_m})\vee \boldsymbol{LessEq}(\overline{p_m},\overline{r_m})\big)\Big)\\ \vee&\boldsymbol{Less}(\ulcorner \boldsymbol{b-1}\urcorner,\overline{r_m}).\end{aligned}$$

The first disjunct handles the case where the interval is non-looping: $[\![\overline{q_m}]\!] - [\![\overline{p_m}]\!] = a-1$ (because the interval is closed) and Player Two's choice falls between them. We require that $[\![\overline{s_m}]\!] = [\![\overline{t_m}]\!] = 0$ to ensure the equilibrium is unique, as otherwise Player One would have been able to assign any value to those variables without affecting his chance of winning. The second disjunct handles the looping case; here $[\![\overline{s_m}]\!]$ and $[\![\overline{t_m}]\!]$ store the length of the interval on either side of zero. The last disjunct of serves to award the game to One should Two name a d outside of $[0, b-1]_{\mathbb{N}}$.

As the game is zero-sum, γ_2 is simply $\neg\gamma_1$.

Now let us verify that the equilibrium is unique. We say the *cover weight* of a number i is the sum of the weights Player One attaches to every interval containing i. In other words, the cover weight of i is the probability that Player Two will lose if she plays i with probability 1. If w_j is the weight Player One attaches to the interval starting at j then the cover weight of i, c_i, can be expressed as follows:

$$c_i = \sum_{j\equiv i-a+1 \mod b}^{i} w_j.$$

That is, the first w_j in the sum is the interval with i at its rightmost point, and hence its leftmost point is at $i-(a-1) \mod b$.

Now observe that a best response for Player Two to any strategy of Player One is to play $\arg\min_i c_i$ with probability 1, giving Player One a utility of $\min_i c_i$; Player One's maxmin strategy is thus to maximise the smallest of the cover weights. Furthermore, we can see that the sum of all the cover weights is invariant and equal to a (as the sum of all w_i is equal to 1 and each w_i appears in a distinct intervals). It follows that the only way to maximise the smallest cover weight is to make all c_i equal.

We will demonstrate that $c_i = c_{i+1}$ implies that $w_{i-a+1 \mod b} = w_{i+1}$. As a is a generating element in the additive group of b elements (this is where coprimality comes in), this will establish that all w_i must be the same.

The argument itself is trivial. Suppose $c_i = c_{i+1}$. If we expand this we have:

$$w_{i-a+1} + w_{i-a+2} + \cdots + w_i = w_{i-a+2} + w_{i-a+3} + \cdots + w_{i+1}.$$

By subtracting $(w_{i-a+2} + \cdots + w_i)$ from both sides we have that $w_{i-a+1} = w_{i+1}$.

We have thus shown that:

a) In every equilibrium Player One must make all the cover weights equal.
b) The only way to make all the cover weights equal is to play every interval with equal weight.

This settles the argument for Player One. We turn to Player Two.

This time we define the *breadth* of the interval starting at i, b_i, to be the sum of the weights, w_j, that Player Two attaches to every number in that interval:

$$b_i = \sum_{j=i}^{i+a-1 \mod b} w_j.$$

By an argument parallel to the one above, we see that Player Two seeks to choose a strategy that will make all b_i equal. Assuming $b_i = b_{i+1}$, this leads us to:

$$w_i + w_{i+1} + \cdots + w_{i+a-1} = w_{i+1} + w_{i+2} + \cdots + w_{i+a}.$$

This allows us to invoke the same argument.

In sum, we have shown that every equilibrium involves all c_i and b_i being equal, and the only way to achieve this is by randomising equally over all intervals and integers. This equilibrium is thus unique. □

We also introduce the following shorthand:

Definition 3.27. Let $G = (\Phi_1, \ldots, \Phi_n, \gamma_1, \ldots, \gamma_n)$ be a Boolean game. We use $\gamma_i(G)$ to refer to γ_i and $\mathrm{var}_i(G)$ to refer to Φ_i. This notation will be useful in the presence of multiple games, as it will allow us to distinguish between $\gamma_i(G)$ and $\gamma_i(G')$.

If G is a game or φ is a formula, we will also write $\mathrm{var}(G)$ and $\mathrm{var}(\varphi)$ to refer to the set of all variables present in the game or formula respectively. ■

4. Results

4.1. Rational and irrational equilibria

We begin with the problem of determining whether a Boolean game has an equilibrium where all the strategy weights are rational numbers.

Theorem 4.1. RationalNash *for three-player Boolean games is* NEXP*-hard.*

For intuition, we will sketch the proof of the theorem for games in normal form; then we will show how the same argument can be replicated in the case of Boolean games.

Let G'_1 be a three-player game where every equilibrium is irrational, and G'_2 an arbitrary two-player game. Given a rational v, the problem of determining whether G'_2 has an equilibrium where Player One gets a utility of at least v (i.e., (G'_2, v) is a positive instance of ∃GuaranteeNash) is NP-complete. Furthermore, since G'_2 has two players, we know that if there exists such an equilibrium, there also exists a rational equilibrium where Player One gets a utility of at least

v. We want to build an overarching game in which the players choose which of G_1' or G_2' to play in, and they choose to play in G_1' if and only if there is no equilibrium of G_2' in which Player One gets a utility of at least v. That is, the overarching game has has a rational equilibrium if and only if (G_2', v) is a positive instance of ∃GuaranteeNash.

To do this we first perform the following tweaks to G_1' and G_2'. Transform G_1' into G_1 by letting $u_1 = u_1' + v$. That is, whatever utility Player One got in G_1' he gets v more in G_1. Transform G_2' into G_2 by, first, adding a third player with no strategies—obviously this will not affect the equilibria of the game—and then letting $u_2 = u_2' + 1/2$ and $u_3 = u_3' + 1/2$. Observe that these tweaks have the effect that Player One gets a utility of at least v in every profile in G_1, and the other two players a utility of at least $1/2$ in G_2.

In the overarching game a player's strategy is a triple, (s_1, s_2, s_3), s_1 and s_2 are strategies from G_1 and G_2 respectively, and s_2 is the choice of G_1 or G_2 to play in. If the players agree which game to play in, they get the utility from that game. If the players disagree, then the players who choose to play in G_2 get 0, Player One gets v if he chooses to play in G_1, and players Two and Three get $1/2$ if they choose to play in G_1.

As a result, if (G_2', v) is a positive instance of ∃GuaranteeNash then there is a rational equilibrium where all the players play the rational equilibrium in G_2—Player One has no incentive to deviate because he's getting a utility of at least v, and he would get v by deviating, and players Two and Three are getting a utility of at least $1/2$, and that is also their utility for deviating. If, on the other hand, there is no equilibrium in G_2 guaranteeing Player One a utility of v then Player One will always choose to play in G_1, as that would give him a utility of at least v. Given that Player One is choosing to play in G_1 then the other two players will also choose to play in G_1—they get a utility of zero for choosing G_2, and at least $1/2$ for G_1—and hence the only equilibria of the game will be the irrational equilibria of G_1.

Proof. Theorem 3 in Bilò and Mavronicolas (2012) gives an example of a three-player win-lose game, call it G_1', that has only irrational equilibria. Players One and Two have two strategies each, but Player Three has three, and as such the game as given does not have a Boolean representation. However, the game has the *positive utility property*: for any choice of strategies by two players, the third has a response that will yield him strictly positive utility. We can thus extend G_1' into G_1 that has a fourth strategy for Player Three, which operates as follows:

1. Player Three's payoff for choosing the fourth strategy is zero.
2. Player One and Two's payoff from any profile where Player Three chooses the fourth strategy are the same as if Player Three chose his first strategy instead.

Observe that in no equilibrium of G_1 would Player Three attach positive weight to his fourth strategy—the positive utility property of the game ensures that he can always do better than zero utility. Thus G_1 does not introduce any new

equilibria; they are the same as the equilibria of G_1', and ergo irrational. As every player now has either two or four strategies, G_1 has a Boolean representation (Fact 3.11).

Now let the pair (G_2, v) be an instance of $\exists$GuaranteeNash with three players. Recall that only two players are needed for NEXP-hardness (Theorem 3.15), and so we can assume that Player Three does not control any variables in G_2. This guarantees that G_2 has rational equilibria (Proposition 3.13).

Define a Boolean game as follows:

$$
\begin{aligned}
\Phi_1 &= \mathrm{var}_1(G_1) \cup \mathrm{var}_1(G_2) \cup \{\, \mathit{Choice}_1 \,\} \cup \mathrm{var}_1(\mathfrak{G}_1(v)),\\
\Phi_2 &= \mathrm{var}_2(G_1) \cup \mathrm{var}_2(G_2) \cup \{\, \mathit{Choice}_2 \,\} \cup \mathrm{var}_1(\mathfrak{G}_2(1/2)) \cup \mathrm{var}_2(\mathfrak{G}_3(1/2)),\\
\Phi_3 &= \mathrm{var}_3(G_1) \cup \mathrm{var}_3(G_2) \cup \{\, \mathit{Choice}_3 \,\} \cup \mathrm{var}_1(\mathfrak{G}_3(1/2)) \cup \mathrm{var}_2(\mathfrak{G}_1(v))\\
&\quad \cup \mathrm{var}_2(\mathfrak{G}_2(1/2)),\\
\gamma_1' &= \big(\bigwedge_{i\leq 3} \mathit{Choice}_i \wedge \gamma_1(G_2)\big) \vee \big(\bigwedge_{i\leq 3} \neg \mathit{Choice}_i \wedge (\gamma_1(G_1) \vee \gamma_1(\mathfrak{G}(v)))\big)\\
&\quad \vee \big(\neg \mathit{Choice}_1 \wedge (\mathit{Choice}_2 \vee \mathit{Choice}_3) \wedge \gamma_1(\mathfrak{G}_1(v))\big),\\
\gamma_2' &= \Big(\big(\bigwedge_{i\leq 3} \mathit{Choice}_i \wedge (\gamma_2(G_2) \vee \gamma_1(\mathfrak{G}_2(1/2)))\big) \vee \big(\bigwedge_{i\leq 3} \neg \mathit{Choice}_i \wedge \gamma_2(G_1)\big)\\
&\quad \vee \big(\neg \mathit{Choice}_2 \wedge (\mathit{Choice}_1 \vee \mathit{Choice}_3) \wedge \gamma_1(\mathfrak{G}_2(1/2))\big)\Big) \wedge \gamma_2(\mathfrak{G}_3(1/2)),\\
\gamma_3' &= \Big(\big(\bigwedge_{i\leq 3} \mathit{Choice}_i \wedge (\gamma_3(G_2) \vee \gamma_1(\mathfrak{G}_3(1/2)))\big) \vee \big(\bigwedge_{i\leq 3} \neg \mathit{Choice}_i \wedge \gamma_3(G_1)\big)\\
&\quad \vee \big(\neg \mathit{Choice}_3 \wedge (\mathit{Choice}_1 \vee \mathit{Choice}_2) \wedge \gamma_1(\mathfrak{G}_3(1/2))\big)\Big) \wedge \gamma_2(\mathfrak{G}_2(1/2))\\
&\quad \wedge \gamma_2(\mathfrak{G}_1(v)).
\end{aligned}
$$

Intuitively, what we are doing is separating the pure profiles into three categories: the players coordinate on G_1, G_2, or a miscoordination occurs.This is implemented by the choice variables in each disjunct. The purpose of the gadget games is to provide floors and ceilings on the utility a player can attain in a given scenario. In the informal sketch above, this was done simply by translating the utility functions of the players by a constant. We cannot do this in the setting of Boolean games, so we use conjunction and disjunction instead. Observe that winning $\mathfrak{G}(v)$ is a disjunct for Player One in the case of G_1. It thus acts as a floor on his utility in that state: no matter what happens in the rest of the game, he has a probability of at least v of satisfying $\gamma_1(\mathfrak{G}(v))$. In the case of miscoordination, being a conjunct, it acts as a ceiling—whatever his probability of satisfying the rest of the clause, he must also satisfy $\gamma_1(\mathfrak{G}(v))$. For Players Two and Three, we likewise add a floor of $1/2$ to G_2, and a ceiling of $1/2$ to miscoordination.

Player Two and Three also have additional conjuncts added to the end of their formula—$\gamma_2(\mathfrak{G}_3(1/2))$ and $\gamma_2(\mathfrak{G}_2(1/2)) \wedge \gamma_2(\mathfrak{G}_1(v))$ respectively. In order to satisfy their goal formulae, the players in question *must* also satisfy these conjuncts. This has the effect of them always playing the optimal strategy in

the subgames $\mathfrak{G}_1, \mathfrak{G}_2, \mathfrak{G}_3$ in any equilibrium.

Suppose $(G_2, v) \in \exists$GuaranteeNash. We claim there is a rational equilibrium where Players One through Three set $Choice_i$ to *true*, play the rational equilibrium of G_2 that satisfies the payoff constraint over the variables in $\text{var}_i(G_2)$, the equilibrium strategy over $\text{var}_i(\mathfrak{G}_j)$ and every other variable to *false*.

Player One has no incentive to deviate while $Choice_1$ is *true*: he is indifferent about what he does with $\text{var}_1(G_1)$ and $\text{var}_1(\mathfrak{G}_1(v))$ as those variables do not affect his ability to satisfy $\gamma_1(G_2)$, and he has no incentive to deviate over $\text{var}_1(G_2)$ as G_2 is in equilibrium. Should he set $Choice_1$ to *false*, then his utility will depend only on his ability to satisfy $\gamma_1(\mathfrak{G}_1(v))$. The probability of that is v, the "ceiling" which we have set earlier, and that is at most how much he is getting in the current profile. Player Two (symmetrically, Three) has no incentive to deviate over $\text{var}_1(\mathfrak{G}_2(1/2))$, $\text{var}_2(\mathfrak{G}_3(1/2)$, or $\text{var}_2(G_2)$ as those games are in equilibrium, and the variables in $\text{var}_2(G_1)$ do not affect her current utility. If she deviates by setting $Choice_2$ to *false*, then she will be getting a utility of $1/4$—the probability of satisfying $\gamma_1(\mathfrak{G}_2(1/2))$ *and* $\gamma_2(\mathfrak{G}_3(1/2))$—whereas in the current profile she is getting $(1 - 1/2 \cdot x) \cdot 1/2$, where x is her probability of losing G_2. As x is at most 1, $1/4$ is the floor on her current utility, and hence such a deviation could do no better.

Now suppose $(G_2, v) \notin \exists$GuaranteeNash. Consider an equilibrium of G where p, q and r are the probabilities of players One, Two and Three respectively opting to play in G_2, i.e. the marginal probabilities of setting $Choice_i$ to *true*. We need to prove that all three are less than one. This will establish that the players have a non-zero chance of coordinating on G_1, and hence they must play an equilibrium strategy over $\text{var}_i(G_1)$. As all such equilibria are irrational, then the equilibrium of G must also be irrational.

We start with Player One. His utility is $pqr \cdot y + (1-p)(1-(1-q)(1-r)) \cdot v + (1-p)(1-q)(1-r) \cdot x$ (payoff from G_2, from miscoordination, and from G_1), with y being strictly less than v and x being at least v. Clearly, the way to maximise this is to set p to zero.

Given that p is 0, Player Two (mutatis mutandis, Three) get a utility of 0 if they set their choice variable to *true*, and either $1/4$ (from miscoordination) or something strictly positive (from the positive utility property of G_1). Hence q and r are also 0—the only equilibria of G involve coordinating on G_1. As all equilibria of G_1 are irrational, so are the equilibria of G. □

The construction for IrrationalNash is a lot simpler, as the problem reduces to checking whether a game has an infinite number of equilibria or not.

Theorem 4.2. IrrationalNash *is* NP*-complete for two-player games in normal form, and* NEXP*-complete for two-player Boolean games.*

Proof. We first address games in normal form. For membership in NP, guess a support pair (X, Y) construct the associated linear system as in Proposition 3.13. Verify that it induces infinitely many equilibria, and invoke Fact 3.16.

For hardness, we reduce from ∃GuaranteeNash. Let (G, v) be a game and a payoff constraint. Without loss of generality, suppose G has non-negative payoffs. For the normal form case, simply extend G into a G' in the following way:

1. Duplicate each of Player One's strategies. That is, for every $s_i \in S_1$ in G, G' has s_i and s'_i, with the property that $u_j(\boldsymbol{s}_{-1}(s_i)) = u_j(\boldsymbol{s}_{-1}(s'_i))$ for all $\boldsymbol{s}$ and $j \in \{1, 2\}$.
2. Give Player One a new strategy, a, that yields him a utility of v in every profile. Give Player Two a new strategy b, with the property that $u_2(a, b) = 1$, $u_2(s \neq a, b) = -1$, and $u_2(a, s \neq b) = -1$.

If $(G, v) \in$ ∃GuaranteeNash then G' has infinitely many equilibria—any equilibrium $\boldsymbol{\sigma}$ of G satisfying the payoff constraint generates infinitely many equilibria in G', as Player One can freely distribute strategy weights between duplicate strategies. If $(G, v) \notin$ ∃GuaranteeNash then G' has a unique equilibrium: (a, b). Player One will play a with probability 1 to get a utility of v, and given that One is playing a, Two will play b for a utility of 1. It follows that G' has an irrational equilibrium if and only if G has an equilibrium whence Player One derives a utility of at least v.

For the case of Boolean games, let (G, v) be a game and payoff constraint as before. Construct $\mathfrak{G}(v)$, and let G' be the following:

$$
\begin{aligned}
\Phi_1 =& \mathrm{var}_1(G) \cup \{ \mathit{Dummy}, \mathit{Choice}^1 \} \cup \mathrm{var}_1(\mathfrak{G}(v)),\\
\Phi_2 =& \mathrm{var}_2(G) \cup \{ \mathit{Choice}^2 \} \cup \mathrm{var}_2(\mathfrak{G}(v)),\\
\gamma_1 =& (\gamma_1(G) \wedge \mathit{Choice}^1 \wedge \mathit{Choice}^2)\\
& \vee (\gamma_1(\mathfrak{G}(v)) \wedge \neg \mathit{Choice}^1 \wedge \neg \mathit{Dummy} \wedge \bigwedge_{p \in \Phi_1} \neg p),\\
\gamma_2 =& (\gamma_2(G) \wedge \mathit{Choice}^1 \wedge \mathit{Choice}^2)\\
& \vee (\gamma_2(\mathfrak{G}(v)) \wedge \neg \mathit{Choice}^1 \wedge \neg \mathit{Choice}^2 \wedge \bigwedge_{p \in \Phi_2} \neg p).
\end{aligned}
$$

The goal formulae are composed of two disjuncts. In the first the players coordinate on G and reap the outcome of that game, in the other the players play $\mathfrak{G}(v)$. Note that the goal formulae are asymmetric: Player One can unilaterally deviate from G to $\mathfrak{G}(v)$, but Player Two can only deviate with Player One.

If $(G, v) \in$ ∃GuaranteeNash then G' has infinitely many equilibria—the *Dummy* variable does not occur in γ_1, and thus has the same effect as duplicating each of Player One's strategies did—every strategy now has the additional choice of setting *Dummy* to *true* or *false*, but this does not affect the outcome of the game. If $(G, v) \notin$ ∃GuaranteeNash then the unique equilibrium involves the players setting the variables in $\mathrm{var}_1(G)$, $\mathrm{var}_2(G)$, *Dummy*, Choice^1 and Choice^2 to *false* and playing the equilibrium strategies in $\mathfrak{G}(v)$. This is because any equilibrium of G would yield Player One less than v utility, and he can unilaterally deviate to $\mathfrak{G}(v)$ where he can guarantee himself at least v. Once

in $\mathfrak{G}(v)$ the equilibrium is guaranteed as neither player can unilaterally deviate from $\mathfrak{G}(v)$ to G—both *Choice* variables appear in their left disjuncts.

G' thus has infinitely many equilibria if and only if G has an equilibrium where Player One derives at least v utility. With Fact 3.16, this proves the theorem. □

4.2. Translations

The frameworks of Schoenebeck and Vadhan (2012) and Feigenbaum et al. (1995) are similar in flavour to Boolean games, and in those frameworks many algorithmic problems of interest to the Boolean games community have been addressed. However, these results do not transfer easily to our setting as Boolean games are less expressive and, assuming the parallel hierarchy does not collapse, less succinct. However, it would be a shame to have to reinvent the wheel. In this section we will demonstrate how the proof techniques presented in this paper can be used to translate some of these proofs into the language of Boolean games.

We first consider IsNash—the complexity of determining whether a given mixed-strategy profile is an equilibrium. The reader is asked to note carefully the definition of the problem in Figure 3.1—the input to the problem consists of only the non-zero weights, i.e. the support of the strategies. This is crucial as if we ask the input to specify every zero explicitly, then the input would be of size $O(2^k)$ for a game with k variables. With an input this big, we can certainly verify that a profile is an equilibrium in polynomial time, but it also defeats the purpose of considering succinct representations to begin with.

The second point to note is that while the other problems studied are hard with two or three players, the proof of IsNash requires that the number of players is unbounded. For any fixed number of players, IsNash is coNP-complete—simply guess a pure strategy deviation.

Our hardness proof is based on Theorem 5.5 of Schoenebeck and Vadhan (2012). The difference is that the authors have access to the full computational power of circuits, whereas we must instead explicitly construct a formula of propositional logic serving the same end.

The relevant complexity class is $\text{coNP}^{\#\text{P}}$, i.e. the complement of those decision problems solved by a nondeterministic polynomial-time machine with access to an oracle to #P. The relevant canonical problem is the following:

$\text{NP}^{\#\text{P}}$TM	
Input:	A nondeterministic, polynomial-time machine M with an oracle to #P, an input word w, and a computation bound in unary k (i.e., the string 1^k).
Output:	YES if M accepts w in at most k steps, NO otherwise.

Fact 4.3. $\text{NP}^{\#\text{P}}$TM *is complete for* $\text{NP}^{\#\text{P}}$.

The proof of Schoenebeck and Vadhan (2012) relies on a convenient characterisation of $\text{NP}^{\#\text{P}}$TM in their Lemma 5.7. Roughly speaking, the lemma states that we can without loss of generality assume that a problem in $\text{NP}^{\#\text{P}}$ can be

solved with a single call to the oracle. We have need of a modified version of this result:

Lemma 4.4. *For every $L \in \mathrm{NP}^{\#\mathrm{P}}$, there is a nondeterministic polynomial-time machine M that decides L, and that makes just a single* #Sat *oracle query on any accepting computation. Moreover, the formula in this query is of the form:*

$$\boldsymbol{OneOf}(p_1, \ldots, p_n) \wedge \bigvee_{i \leq n} (\varphi_i \wedge p_i),$$

where each φ_i is a formula in 2CNF*.*

The idea of the proof is that whereas an arbitrary computation in $\mathrm{NP}^{\#\mathrm{P}}$ may submit formulae φ_1 through φ_n to the oracle, we can instead submit a single formula of the form given in the lemma statement. The number of satisfying truth assignments to φ_i can then be recovered from the number of satisfying truth assignments to the combined formula.

Proof. Let M be a nondeterministic machine with a #P oracle. Recall that #2CNFSat is #P-complete (Valiant, 1979), so we can assume that M only submits formulae in 2CNF to the oracle.

Construct an M' that guesses an accepting computation history of M including any oracle queries made, which we denote φ_1 through φ_n, and the answers to those queries. Having done this, M' constructs a φ with the property that given the number of satisfying truth assignments of φ it is possible, in polynomial time, to derive the number of satisfying truth assignments of each of the φ_i. M' will then submit φ to the oracle, verify that its previous guesses for the oracle answers were correct, and if so, accept. Clearly such an M' will accept w in a polynomial number of steps if and only if M does. It remains to show how such a φ can be constructed.

Without loss of generality assume that the variables used in every φ_i are unique to that formula. Let k_i be the number of propositional variables in φ_i. Expand each φ_i to φ_i' in the following manner:

$$\varphi_i' = \varphi_i \wedge \bigwedge_{j < k_1 + \cdots + k_{i-1}} (q_j \vee \neg q_j), \text{ for fresh } q_j.$$

Note that all these new variables are completely redundant – any assignment to them will satisfy the clauses in which they appear. As a result, if φ_i has m_i satisfying truth assignments, then φ_i' has $2^{k_1 + \cdots + k_{i-1}} m_i$. This means that the binary integer representing the number of satisfying truth assignments of φ_i' will have 0s in the $k_1 + \cdots + k_{i-1}$ least significant positions, potentially non-zero entries in the next k_i, and nothing beyond that.

Given fresh variables p_i, our desired formula is the following:

$$\varphi = \boldsymbol{OneOf}(p_1, \ldots, p_n) \wedge \bigvee_{i \leq n} (\varphi_i' \wedge p_i).$$

Since one and only one of p_i must be true, these variables do not introduce any new satisfying truth assignments and thus the number of assignments that satisfy φ is simply $\sum_{i \leq n} 2^{k_1+\cdots+k_{i-1}} m_i$. To extract m_i we need only to look at the bits between positions $k_1 + \cdots + k_{i-1}$ and $k_1 + \cdots + k_{i-1} + k_i$. □

For the coming proof we need to tweak the gadget games of Lemma 3.26. So far, the subgames $\mathfrak{G}(u)$ have allowed us to pretend we were working in a setting with finer grained preferences than Boolean games. The shortcoming of the construction, however, is that by definition $\mathfrak{G}(u)$ is completely modular and separate from the rest of the game. This, of course, is working as intended—when we come to evaluate the payoffs we want to know that the value of $\mathfrak{G}(u)$ is indeed u, regardless of what else has happened in the game. The trouble is that $\mathfrak{G}(u)$ is completely unresponsive to the passage of play; we need to know, in advance, what utility values we need and code them in explicitly. Should we want to associate a different utility to every possible strategy we would need formulae of the form $(p_1 \wedge \neg p_2 \wedge \cdots \wedge p_k) \rightarrow \gamma_1(\mathfrak{G}(u_i))$, which would not be feasible in polynomial time.

To deal with this we introduce a different type of gadget game, the value of which can be altered externally.

Definition 4.5. Fix a sequence $\overline{u_n}$. Let $\mathcal{G}(\overline{u_n})$ denote a two-player, zero-sum game with value $[\![\overline{u_n}]\!]/2^n$, with Player One controlling the variables $\overline{u_n}$. That is, the value of the game is determined *after* Player One chooses an assignment to $\overline{u_n}$.

Formally, $\mathcal{G}(\overline{u_n})$ looks as follows:

$$
\begin{aligned}
\Phi_1 =& \{p_1, \ldots, p_n, q_1, \ldots, q_n, s_1, \ldots, s_n, t_1, \ldots, t_n, u_1, \ldots, u_n\} \\
\Phi_2 =& \{r_1, \ldots, r_n\} \\
\gamma_1 =& \big(\boldsymbol{Sub}(\overline{q_n}, \overline{p_n}, \overline{u_n}) \wedge \boldsymbol{LessEq}(\overline{q_n}, \ulcorner \mathbf{2^n - 1} \urcorner) \wedge \boldsymbol{LessEq}(\overline{r_n}, \overline{q_n}) \\
& \wedge \boldsymbol{LessEq}(\overline{p_n}, \overline{r_n})\big) \vee \Big(\boldsymbol{Add}(\overline{s_n}, \overline{t_n}, \overline{u_n})) \wedge \boldsymbol{Sub}(\overline{q_n}, \ulcorner \mathbf{0} \urcorner, \overline{s_n}) \\
& \wedge \boldsymbol{Sub}(\ulcorner \mathbf{2^n - 1} \urcorner, \overline{p_n}, \overline{t_n}) \wedge \big(\boldsymbol{LessEq}(\overline{r_n}, \overline{q_n}) \vee \boldsymbol{LessEq}(\overline{p_n}, \overline{r_n})\big)\Big) \\
& \vee \boldsymbol{Less}(\ulcorner \mathbf{2^n - 1} \urcorner, \overline{r_n}).
\end{aligned}
$$

■

Example 4.6. At first glance Definition 4.5 may appear a little odd. We are essentially letting Player One rig the dice before he casts them—what behaviour can we possibly expect except that of choosing $[\![\overline{u_n}]\!] = 2^n - 1$ every single time? And if we do want a game of that form, we have already seen how to construct $\mathfrak{G}(\frac{2^n-1}{2^n})$ earlier.

The merit of this definition is that, unlike in Lemma 3.26, we do *not* require that all the variables of $\mathcal{G}(\overline{u_n})$ are fresh. We use parameter notation for $\overline{u_n}$ to indicate that these variables can occur elsewhere in the construction. Suppose,

for example, that Player One is trying to satisfy a formula of the form:

$$\gamma_1 = \varphi(\overline{v_n}) \wedge \boldsymbol{Equal}(\overline{v_n}, \overline{u_n}) \wedge \gamma_1(\mathcal{G}(\overline{u_n})).$$

Now the choice of $[\![\overline{u_n}]\!]$ will very much depend on what is going on in $\varphi(\overline{v_n})$. ■

Theorem 4.7. IsNash coNP$^{\#\mathrm{P}}$*-hard for Boolean games.*

We follow the argument of Schoenebeck and Vadhan (2012), which can be stated as follows: given a nondeterministic M with access to a #Sat oracle, a w, and 1^k we construct a game where Player One will choose between specifying a computation history of the machine (which will of course include the oracle query and answer received, if a query is made), or to abstain from the game and earn some base level of utility u. If Player One chooses to specify a computation history, the payoffs of the game will be so structured as to give Player One more than u utility if he specifies an accepting history with a valid oracle response, or at most u if either the specified history is non-accepting or the oracle response specified by Player One is incorrect.

If we manage, in polynomial time, to construct such a game and profile then we will have proved the theorem—we map $(M, w, 1^k)$ to $(G, \boldsymbol{\sigma})$ where $\boldsymbol{\sigma}$ will involve Player One abstaining. Thereby, if $(M, w, 1^k) \notin \mathrm{NP}^{\#\mathrm{P}}\mathrm{TM}$ then Player One cannot deviate to any strategy that will yield him more than u utility, so the profile will be in equilibrium and thus $(G, \boldsymbol{\sigma}) \in$ IsNash. On the other hand, if $(M, w, 1^k) \in \mathrm{NP}^{\#\mathrm{P}}\mathrm{TM}$ then Player One will be able to deviate to an accepting computation history with a valid oracle response, netting more than u utility and establishing that $(G, \boldsymbol{\sigma}) \notin$ IsNash.

Proof. Given an instance $(M, w, 1^k)$ of $\mathrm{NP}^{\#\mathrm{P}}\mathrm{TM}$, we will construct an instance $(G, \boldsymbol{\sigma})$ of IsNash.

In total G will have $4 + 6k$ players, although Player One is the only one whose preferences will matter. Player Two will play a zero-sum game of value u with Player One. The next $2k$ players will be dubbed group A, of which k are called *proposition-players* and k *clause-players*. Group B consists of the next $2k$ players with the same division into proposition and clause-players. Groups A and B will serve to punish Player One for playing an incorrect oracle response. The final $2k + 2$ players are group C, and their rôle is to enable the constructed profile $\boldsymbol{\sigma}$ to be polynomial in size, as will be made clear at the end of the proof.

To facilitate the reduction we will augment the machine with registers that deal with the query to, and the response from, the oracle. The output registers will be read-only, and the input registers write-once.

The output registers are simply k cells that will take a binary representation of the oracle's answer, with the least significant bit first, concluding with an end-of-file character $\perp$. As k is an upper bound on the size of the formula submitted to the oracle, the formula can have at most 2^k satisfying assignments and as such k cells are sufficient to store whatever output the oracle makes.

To understand the shape of the input registers, consider the form of the formula in the statement of Lemma 4.4. Such a formula is uniquely determined

by the subformulae φ_i, and each φ_i is in 2CNF. As such, each φ_i can be specified simply by listing the literals in the order in which they appear. We will thus endow the machine with $k^2 + 1$ input register cells: k sets of registers, each consisting of k cells, and a header register. The interpretation of the ith register is φ_i, and the jth entry of the ith register specifies the jth literal of φ_i. The header register will contain a special integer symbol corresponding to the amount of φ_i formulae that are present. To facilitate the encoding of literals we assume the machine alphabet contains the symbols $\{ q_1, \ldots, q_k, \neg q_1, \ldots, \neg q_k \}$ as well as the integer symbols $\{ 1, \ldots, k \}$.

The machine will write to an input register by placing the appropriate symbol at the beginning of the tape, followed by the index of the register to write to, and entering a special write state.

At this point it behoves us to summarise the assumptions we have made to convince the reader that at no point have we assumed away the issue at hand—namely, none of the restrictions we place on M are substantial enough to prevent it from representing an arbitrary polynomial-time machine.

1. We assume the machine issues at most one query to the oracle, and the query is in a very specific format. This is justified by Lemma 4.4.
2. We assume the machine is equipped with registers. As random access memory would at most allow a polynomial speed up of the machine, there is no generality lost in assuming that an arbitrary polynomial-time machine has these.
3. We assume the machine has to write the query formula, symbol by symbol, to the input registers, rather than just dumping it on the tape. This is at worst a polynomial slow down, so the machine is still polynomial time.
4. We assume the machine has additional tape symbols. This will allow at most a polynomial speed up over a machine working only in binary.

We will introduce the variables Player One controls gradually throughout the proof and give them together at the end. For now, we provide him with the variable *Abstain* and the goal formula:

$$\gamma_1 = \big(\mathit{Abstain} \wedge \gamma_1(\mathfrak{G}(u))\big) \vee \big(\neg \mathit{Abstain} \wedge \mathit{Machine} \wedge \mathit{Oracle}\big).$$

The subformula *Machine* asserts that Player One specifies a valid computation history of M on w (with the possible exception of an incorrect oracle response) that ends in an accepting state. Encoding a polynomial-time machine is trivial so we relegate the construction to the appendix.

Lemma 4.8. *There exists a formula, Machine, such that:*

1. $\nu \vDash \mathit{Machine}$ *if and only if* ν *represents an accepting run of* M *on* w *in* k *steps that is correct with respect to everything except possibly the oracle responses.*
2. $|\mathit{Machine}|$ *is polynomial in the size of* $(M, w, 1^k)$.
3. *Machine has variables* $Q_{i,j,l}$ *and* $NQ_{i,j,l}$ *representing that the* i*th register has the symbol* q_l *or* $\neg q_l$ *in cell* j *respectively. That is, the* j*th literal of* φ_i *is* q_l *or* $\neg q_l$.

4. *Machine has variables R_i to denote the value of the ith most significant bit of the oracle's response.*
5. *Machine has variables $\{ F_1, \ldots, F_k \}$ to denote the value of the header register.*

While the details of the construction are unimportant, the reader should pay attention to points 3, 4 and 5 as those variables will be used in the sequel.

The subformula *Oracle* will be designed in a manner to ensure that $\mathbb{E}[\mathit{Oracle} \mid \boldsymbol{\sigma'}] > u$ if and only if the formula specified by Player One in $\boldsymbol{\sigma'}$ does in fact have the number of satisfying assignments that Player One asserted. To do this we will implement the g function from Theorem 5.5 of Schoenebeck and Vadhan (2012), after transforming it to fit into the $[0, 1]$ range required by a Boolean game.

Of course, the difficulty here is that we do not know precisely how many satisfying assignments φ has—Player One can submit whichever formula he wants. This is where groups A and B come in. The strategies chosen by the players in a group will define, uniformly at random, a truth assignment to φ. Thus while we do not know m, the number of satisfying assignments that φ has, we do know that the assignment chosen by group A or B has precisely a $m/2^k$ chance of satisfying φ.[9] Some algebraic manipulation can then lead us to m.

To specify a truth assignment, we equip the ith proposition-player of A with a single variable A_i. The truth value of this is to be interpreted as the value assigned to q_i.[10] In $\boldsymbol{\sigma}$, each of these players sets their variable to *true* with probability $1/2$.

The clause-players in A are in charge of the p_i variables. In the case where φ has i subformulae, the ith clause-player chooses which p_j is true; the other clause-players are ignored. The ith clause-player in A controls the variables $CA_1, \ldots, CA_i$, and in $\boldsymbol{\sigma}$ he sets CA_j to *true* and every other variable to *false* with probability $1/i$. Player One's choice of F_i will tell us which clause-player to listen to.

Group B is built identically to group A.

Recall that $[\![\overline{R_k}]\!]$ is the number of satisfying assignments that Player One claims that φ possesses. Let $\varphi(A)$ be the truth value of φ if evaluated by the truth assignment specified by players A, and likewise for $\varphi(B)$. Consider the following $g([\![\overline{R_k}]\!], \varphi(A), \varphi(B))$:

[9] We are tacitly assuming that φ contains all k possible variables. This is of course impossible given the time constraints of the machine, but the assumption is harmless if we treat the unlisted variables as dummies whose truth value does not matter.

[10] The variable names p_i and q_i, as well as the subformula names φ_i, are used in the same sense as in Lemma 4.4.

$$
\begin{aligned}
g([\![\overline{R_k}]\!], \mathit{true}, \mathit{true}) =& \frac{2^{2k+1} - ([\![\overline{R_k}]\!]^2 - 2^{k+1}[\![\overline{R_k}]\!] + 2^{2k})}{2^{2k+2}},\\
g([\![\overline{R_k}]\!], \mathit{true}, \mathit{false}) =& \frac{2^{2k+1} - (-2^k[\![\overline{R_k}]\!] + [\![\overline{R_k}]\!]^2)}{2^{2k+2}},\\
=& g([\![\overline{R_k}]\!], \mathit{false}, \mathit{true}),\\
g([\![\overline{R_k}]\!], \mathit{false}, \mathit{false}) =& \frac{2^{2k+1} - [\![\overline{R_k}]\!]^2}{2^{2k+2}}.
\end{aligned}
$$

Note that this is bounded above by 1 and below by 0, so it is within the range of feasible payoffs for a Boolean game.

The expected value of g, in any $\boldsymbol{\sigma'}$ where groups A and B choose an assignment uniformly at random, is:

$$
\begin{aligned}
\mathbb{E}[g \mid \boldsymbol{\sigma'}] =& \big(\frac{m}{2^k}\big)^2 g([\![\overline{R_k}]\!], \mathit{true}, \mathit{true}) + 2\big(\frac{m}{2^k}\big)\big(1 - \frac{m}{2^k}\big) g([\![\overline{R_k}]\!], \mathit{false}, \mathit{true})\\
&+ \big(1 - \frac{m}{2^k}\big)^2 g([\![\overline{R_k}]\!], \mathit{false}, \mathit{false}),\\
=& \frac{2^{2k+1} - (m - [\![\overline{R_k}]\!])^2}{2^{2k+2}}.
\end{aligned}
$$

Clearly this function is maximised when $[\![\overline{R_k}]\!] = m$. Player One controls $\overline{R_k}$, so it remains to figure out how to incentivise him to maximise g.

The first component in implementing g is a formula that tells us whether the assignment played by players A does in fact satisfy φ. Our requirement that each φ_i is in 2CNF makes it easy to verify whether φ_i is satisfied:

$$
\begin{aligned}
&\mathit{ASatPhi}_i =\\
&\bigwedge_{j \le k/2} \bigvee_{l \le k} \bigvee_{l' \le k} \Big(\big(Q_{i,2j,l} \wedge Q_{i,2j+1,l'} \wedge (A_l \vee A_{l'})\big)\\
&\quad \vee \big(Q_{i,2j,l} \wedge NQ_{i,2j+1,l'} \wedge (A_l \vee \neg A_{l'})\big)\\
&\quad \vee \big(NQ_{i,2j,l} \wedge Q_{i,2j+1,l'} \wedge (\neg A_l \vee A_{l'})\big)\\
&\quad \vee \big(NQ_{i,2j,l} \wedge NQ_{i,2j+1,l'} \wedge (\neg A_l \vee \neg A_{l'})\big)\\
&\quad \vee \big(\boldsymbol{NoneOf}(Q_{i,2j,1}, \ldots, Q_{i,2j,k}, NQ_{i,2j,1}, \ldots, NQ_{i,2j,k})\\
&\qquad \wedge \boldsymbol{NoneOf}(Q_{i,2j+1,1}, \ldots, Q_{i,2j+1,k}, NQ_{i,2j+1,1}, \ldots, NQ_{i,2j+1,k})\big)\Big).
\end{aligned}
$$

The outer conjunction iterates over the clauses in φ_i (a conjunction because every clause in a CNF formula needs to be satisfied). The two following disjunctions iterate over the l and l' that identify which q_l and $q_{l'}$ appear in the clause under consideration (is is the job of *Machine* to ensure Player One places two and only two variables in each clause, so these disjunctions will be satisfied in at most one case).

The first four disjuncts that follow treat with the non-degenerate cases: if Player One said the jth clause contains q_l and $\neg q_{l'}$ ($Q_{i,2j,l} \wedge NQ_{i,2j+1,l'}$), then we require that the players in A set q_l to *true* or $q_{l'}$ to *false* ($A_l \vee \neg A_{l'}$). The last disjunct handles the case where φ_i does not contain a jth clause.

Recall that in verity φ is definitely not going to contain all k variables. If it actually has k', then in our present formulation $\varphi(A)$ will have $2^{k-k'}$ times more satisfying assignments than φ. To deal with this, we will add a formula requiring that all unmentioned variables need to be false.

$$NoSpurious = \bigwedge_{i \leq k} \big(\bigwedge_{j,l \leq k} \boldsymbol{NoneOf}(Q_{j,i,l}, NQ_{j,i,l}) \to \neg A_i \big).$$

Now, suppose for a moment that φ consists of j such subformulae, φ_i. We could combine the formulae $ASatPhi_i$ as follows:

$$ASatPhi[j] = \bigwedge_{i \leq j} (CA_i^j \to ASatPhi_i).$$

That is, if the jth clause-player picks p_i then we check whether φ_i is satisfied.

Of course, we do not know how many terms φ is composed of. Which is why it is kind of Player One to supply us with that information. The final formula is:

$$ASatPhi = NoSpurious \wedge \bigwedge_{i \leq k} (F_i \to ASatPhi[i]).$$

With the canaries sorted, the high level plan is to have something that looks as follows:

$$\boldsymbol{Equal}(\overline{x_n}; \text{numerator of } g(\overline{R_k}, ASatPhi, BSatPhi)) \wedge \gamma_1(\mathcal{G}(\overline{x_n})).$$

That is, Player One is to select a number that must match the numerator of g, and then play a game from Definition 4.5 with that value. This will ensure that Player One's payoff is precisely g. Now we just need a logical formula telling us what the numerator is.

It is a lot easier to verify arithmetic than to do it, so we will ask Player One to do the work for us. He will provide the values of the various terms in g as well as of g itself, and we will pass those values into logical formulae that are true if the numbers add up. To this end, we want to define $\xi_{t,t}(\overline{R_k}, \overline{G^1_{2k+2}})$ to be a formula that is true just if $[\![\overline{G^1_{2k+2}}]\!] = 2^{2k+2} \cdot g([\![\overline{R_k}]\!], true, true)$. That is, $[\![\overline{G^1_{2k+2}}]\!]$ is the numerator of $g([\![\overline{R_k}]\!], true, true)$.

Recall that this numerator looks as follows:

$$2^{2k+1} - ([\![\overline{R_k}]\!]^2 - 2^{k+1}[\![\overline{R_k}]\!] + 2^{2k}).$$

The main operation is subtraction, and we have already seen how to define $\boldsymbol{Sub}$. The minuend, 2^{2k+1}, is a constant that can be represented by a $\top$ followed by $2k+1$ $\bot$s. The trouble is with the subtrahend, but we shall cross that bridge once we reach it. For now, we will ask Player One to tell us what the subtrahend

is via an assignment to $\overline{sG^1_{2k+2}}$, and introduce a new formula, $s\xi_{t,t}$, to verify that Player One is not lying. This gives us $\xi_{t,t}$:

$$\xi_{t,t}(\overline{R_k}, \overline{G^1_{2k+2}}) = \boldsymbol{Sub}(\ulcorner \mathbf{2^{2k+1}} \urcorner; \overline{sG^1_{2k+2}}; \overline{G^1_{2k+2}}) \wedge s\xi_{t,t}(\overline{R_k}, \overline{sG^1_{2k+2}}).$$

Now we have to deal with the subtrahend, $[\![\overline{R_k}]\!]^2 - 2^{k+1}[\![\overline{R_k}]\!] + 2^{2k}$. The main operations are subtraction and addition, which we can do. The rightmost term is a constant which we can deal with, and the middle term is just $\overline{R_k}$ with $k+1$ $\bot$s after it. All we need to do now is to square $[\![\overline{R_k}]\!]$.

Recall the high-school multiplication algorithm: splitting up the multiplication of k-digit integers into k sums, of size up to $2k$. Applied to binary, computing $[\![\overline{R_k}]\!] \cdot [\![\overline{R_k}]\!]$ would involve computing these k sums, the ith being either $[\![\overline{R_k}]\!] \cdot 2^{i-1}$ or 0, depending on whether the ith most significant bit of $[\![\overline{R_k}]\!]$ is 1 or 0. If we ask Player One to provide us the summands (via $\overline{s_{2k}}$), the result of the partial sums $\sum_{j<i} [\![\overline{s^j_{2k}}]\!]$ (via $\overline{S_{2k}}$) and the squared value (via $\overline{R^2_{2k}}$), we can verify that the squared value is correct with k ***Add*** statements.

$$\begin{aligned}
&\boldsymbol{Square}(\overline{R_k}; \overline{R^2_{2k}}; \overline{s^1_{2k}}; \ldots; \overline{s^k_{2k}}; \overline{S^1_{2k}}; \ldots; \overline{S^{k+1}_{2k}}) = \\
&\quad \bigwedge_{j \le k} \left((\boldsymbol{Equal}(\overline{s^j_{2k}}; \overline{R_k} \cdot \ulcorner \mathbf{2^{j-1}} \urcorner) \wedge R_j) \vee (\boldsymbol{Equal}(\overline{s^j_{2k}}; \ulcorner \mathbf{0} \urcorner) \wedge \neg R_j) \right) \\
&\quad \wedge \bigwedge_{j \le k} \left(\boldsymbol{Add}(\overline{S^j_{2k}}; \overline{s^j_{2k}}; \overline{S^{j+1}_{2k}}) \right) \wedge \boldsymbol{Equal}(\overline{S^1_{2k}}; \ulcorner \mathbf{0} \urcorner) \wedge \boldsymbol{Equal}(\overline{S^{k+1}_{2k}}; \overline{R^2_{2k}}).
\end{aligned}$$

$\overline{R_k} \cdot \ulcorner \mathbf{2^{j-1}} \urcorner$ is shorthand for a mixed proposition/constant sequence with $j-1$ $\bot$s on the end of $\overline{R_k}$, and $k-j+1$ $\bot$s at the front to ensure the sequence is of length $2k$. This is necessary as we have only defined ***Equal*** on sequences of the same length.

This gives us what we need:

$$\begin{aligned}
&s\xi_{t,t}(\overline{R_k}, \overline{sG^1_{2k+2}}) = \\
&\quad \boldsymbol{Sub}(\overline{R^2_{2k}}; \overline{R_k} \cdot \ulcorner \mathbf{2k+1} \urcorner; \overline{aG^1_{2k+2}}) \wedge \boldsymbol{Add}(\overline{aG^1_{2k+2}}; \ulcorner \mathbf{2^{2k}} \urcorner; \overline{sG^1_{2k+2}}) \\
&\quad \wedge \boldsymbol{Square}(\overline{R_k}; \overline{R^2_{2k}}; \overline{s^1_{2k}}; \ldots; \overline{s^k_{2k}}; \overline{S^1_{2k}}; \ldots; \overline{S^{k+1}_{2k}}).
\end{aligned}$$

Armed with ***Square***, the other cases succumb to us with ease.

$$\begin{aligned}
\xi_{t,f} = \xi_{f,t} = &\boldsymbol{Sub}(\ulcorner \mathbf{2^{2k+1}} \urcorner; \overline{sG^2_{2k+2}}; \overline{G^2_{2k+2}}) \\
&\wedge \boldsymbol{Sub}(\overline{R^2_{2k}}; \overline{R_k} \cdot \ulcorner \mathbf{2^k} \urcorner; \overline{sG^2_{2k+2}}) \\
&\wedge \boldsymbol{Square}(\overline{R_k}; \overline{R^2_{2k}}; \overline{s^1_{2k}}; \ldots; \overline{s^k_{2k}}; \overline{S^1_{2k}}; \ldots; \overline{S^{k+1}_{2k}}). \\
\xi_{f,f} = &\boldsymbol{Sub}(\ulcorner \mathbf{2^{2k+1}} \urcorner; \overline{R^2_{2k}}; \overline{G^3_{2k+2}}) \\
&\wedge \boldsymbol{Square}(\overline{R_k}; \overline{R^2_{2k}}; \overline{s^1_{2k}}; \ldots; \overline{s^k_{2k}}; \overline{S^1_{2k}}; \ldots; \overline{S^{k+1}_{2k}}).
\end{aligned}$$

The sequences in the above are assumed to be padded with leading $\bot$s whenever necessary.

Now construct three games from Definition 4.5: $\mathcal{G}(\overline{G^1_{2k+2}})$, $\mathcal{G}(\overline{G^2_{2k+2}})$ and $\mathcal{G}(\overline{G^3_{2k+2}})$. Note that if Player One were to attempt to satisfy $\xi_{t,t}(\overline{R_k}, \overline{G^1_{2k+2}}) \wedge \mathcal{G}(\overline{G^1_{2k+2}})$, he would have to set $[\![\overline{G^1_{2k+2}}]\!]$ to the numerator of $g(R, \mathit{true}, \mathit{true})$, and hence his expected utility in $\mathcal{G}(\overline{G^1_{2k+2}})$ would be the numerator of $g(R, \mathit{true}, \mathit{true})$ over 2^{2k+2}, which is precisely $g(R, \mathit{true}, \mathit{true})$. And we have already seen that to maximise g, Player One would want to set $[\![\overline{R_k}]\!] = m$, the number of satisfying truth assignments of φ.

This gives us a first guess for Player One's goal. However, we warn the reader that this is not our final formulation, which is why we mark it with an apostrophe.

$$
\begin{aligned}
\gamma_1' =& (\mathit{Abstain} \wedge \gamma_1(\mathfrak{G}(u))) \vee (\neg \mathit{Abstain} \wedge \mathit{Machine} \wedge \mathit{Oracle}'),\\
\mathit{Oracle}' =& \big(\mathit{ASatPhi} \wedge \mathit{BSatPhi} \wedge \xi_{t,t}(\overline{R_k}, \overline{G^1_{2k+2}}) \wedge \gamma_1(\mathcal{G}(\overline{G^1_{2k+2}}))\big)\\
&\vee \big(\mathit{ASatPhi} \wedge \neg \mathit{BSatPhi} \wedge \xi_{t,f}(\overline{R_k}, \overline{G^2_{2k+2}}) \wedge \gamma_1(\mathcal{G}(\overline{G^2_{2k+2}}))\big)\\
&\vee \big(\neg \mathit{ASatPhi} \wedge \mathit{BSatPhi} \wedge \xi_{f,t}(\overline{R_k}, \overline{G^2_{2k+2}}) \wedge \gamma_1(\mathcal{G}(\overline{G^2_{2k+2}}))\big)\\
&\vee \big(\neg \mathit{ASatPhi} \wedge \neg \mathit{BSatPhi} \wedge \xi_{f,f}(\overline{R_k}, \overline{G^3_{2k+2}}) \wedge \gamma_1(\mathcal{G}(\overline{G^3_{2k+2}}))\big).
\end{aligned}
$$

All that remains is the value of u.

Seeing how if M accepts any word whatsoever, then Player One would be able to construe some machine history to satisfy *Machine*, Player One's choice comes down to taking u utility in the left disjunct or $\mathbb{E}[\mathit{Oracle}' \mid \boldsymbol{\sigma}] = \mathbb{E}[g \mid \boldsymbol{\sigma}] = \frac{2^{2k+1} - (m - [\![\overline{R_k}]\!])^2}{2^{2k+2}}$ utility in the right. Note that if $m = [\![\overline{R_k}]\!]$, where the expectation is maximal, this is just $1/2$. Since we want Player One to have a strict preference for the right disjunct in the situation where such a choice of $[\![\overline{R_k}]\!]$ is possible (else there would be no profitable deviation), u needs to be strictly smaller than $1/2$ but larger than any other possible value of $\mathbb{E}[g \mid \boldsymbol{\sigma}]$. The closest $\mathbb{E}[g \mid \boldsymbol{\sigma}]$ can get to $1/2$ without actually reaching it is when the difference between m and $[\![\overline{R_k}]\!]$ is unity, which gives us $\frac{2^{2k+1}-1}{2^{2k+2}}$.

Suppose we set u to $\frac{2^{2k+1}-0.5}{2^{2k+2}}$. We have then constructed a G and a $\boldsymbol{\sigma}$ satisfying the requirement that $\boldsymbol{\sigma}$ is an equilibrium if and only if M does not have an accepting computation in under k steps, and $|G|$ is polynomial in the size of $(M, w, 1^k)$. Are we done? Unfortunately not. We must not forget the size of $\boldsymbol{\sigma}$, and $\boldsymbol{\sigma}$ involves Player One playing an equilibrium strategy in $\mathfrak{G}(u)$ – this would require him to randomise over $2^{2k+1} - 1$ possible strategies.

To address this, we could try and shift the extra $\frac{1}{2^{2k+2}}$ term to the other disjunct. I.e., Player One can abstain and get $u = 1/2$, and $\mathfrak{G}(1/2)$ would only require him to randomise over two strategies, or he can get $\mathbb{E}[g \mid \boldsymbol{\sigma}] + \frac{1}{2^{2k+2}}$ in trying to satisfy *Oracle*. But that does not solve the problem: Player One may have a small strategy in $\mathfrak{G}(u)$, but Player Two would still have to play an

equilibrium strategy in all the $\mathcal{G}$ games in the *Oracle* subformula, which is also exponentially large. This is where group C comes in.

We introduce a variant of game $\mathcal{G}$:

Definition 4.9. Let $\mathcal{G}(\overline{u_n})$ be as in Definition 4.5. Define $\mathcal{G}'(\overline{u_n})$ to be where in lieu of Player Two, we have Players Two through $n+1$, each controlling just a single bit of what was earlier Player Two's choice.

Formally, $\mathcal{G}'(\overline{u_n})$ looks as follows:

$$\begin{aligned}
\Phi_1 =& \{p_1, \ldots, p_n, q_1, \ldots, q_n, s_1, \ldots, s_n, t_1, \ldots, t_n, u_1, \ldots, u_n\} \\
\Phi_{i>1} =& \{r_{i+1}\} \\
\gamma_1 =& \big(\boldsymbol{Sub}(\overline{q_i}, \overline{p_i}, \overline{u_i}) \wedge \boldsymbol{LessEq}(\overline{q_i}, \ulcorner \mathbf{2^n - 1} \urcorner) \\
& \wedge \boldsymbol{LessEq}(\overline{r_i}, \overline{q_i}) \wedge \boldsymbol{LessEq}(\overline{p_i}, \overline{r_i})\big) \\
& \vee \Big(\boldsymbol{Add}(\overline{s_i}, \overline{t_i}, \overline{u_i})) \wedge \boldsymbol{Sub}(\overline{q_i}, \ulcorner \mathbf{0} \urcorner, \overline{s_i}) \wedge \boldsymbol{Sub}(\ulcorner \mathbf{2^n - 1} \urcorner, \overline{p_i}, \overline{t_i}) \\
& \wedge \big(\boldsymbol{LessEq}(\overline{r_i}, \overline{q_i}) \vee \boldsymbol{LessEq}(\overline{p_i}, \overline{r_i})\big)\Big) \\
& \vee \boldsymbol{Less}(\ulcorner \mathbf{2^n - 1} \urcorner, \overline{r_i}).
\end{aligned}$$

We are not defining the goal formulae of the other players as they are of no importance to us at the present. ■

If the ith player in C plays r_i with probability $1/2$, then every possible integer is realised with probability $1/2^{2k+2}$, as required.

This allows us to state *Oracle* as follows:

$$\begin{aligned}
Oracle = & \big(ASatPhi \wedge BSatPhi \wedge \xi_{t,t}(\overline{R_k}, \overline{G^1_{2k+2}}) \\
& \wedge \boldsymbol{Add}(\overline{G^1_{2k+2}}; \ulcorner \mathbf{1} \urcorner; \overline{fG^1_{2k+2}}) \wedge \gamma_1(\mathcal{G}'(\overline{fG^1_{2k+2}}))\big) \\
\vee & \big(ASatPhi \wedge \neg BSatPhi \wedge \xi_{t,f}(\overline{R_k}, \overline{G^2_{2k+2}}) \\
& \wedge \boldsymbol{Add}(\overline{G^2_{2k+2}}; \ulcorner \mathbf{1} \urcorner; \overline{fG^2_{2k+2}}) \wedge \gamma_1(\mathcal{G}'(\overline{fG^2_{2k+2}}))\big) \\
\vee & \big(\neg ASatPhi \wedge BSatPhi \wedge \xi_{f,t}(\overline{R_k}, \overline{G^2_{2k+2}}) \\
& \wedge \boldsymbol{Add}(\overline{G^2_{2k+2}}; \ulcorner \mathbf{1} \urcorner; \overline{fG^2_{2k+2}}) \wedge \gamma_1(\mathcal{G}'(\overline{fG^2_{2k+2}}))\big) \\
\vee & \big(\neg ASatPhi \wedge \neg BSatPhi \wedge \xi_{f,f}(\overline{R_k}, \overline{G^3_{2k+2}}) \\
& \wedge \boldsymbol{Add}(\overline{G^3_{2k+2}}; \ulcorner \mathbf{1} \urcorner; \overline{fG^3_{2k+2}}) \wedge \gamma_1(\mathcal{G}'(\overline{fG^3_{2k+2}}))\big).
\end{aligned}$$

And the final goal formula:

$$\gamma_1 = (Abstain \wedge \gamma_1(\mathfrak{G}(1/2))) \vee (\neg Abstain \wedge Machine \wedge Oracle).$$

This formula is clearly polynomial in size. The goal formula of every other player is simply:

$$\gamma_i = \bot.$$

Their preferences do not matter, and so they can never be the cause of a deviation from $\boldsymbol{\sigma}$.

The final list of variables used is:

$$\begin{aligned}
\Phi_1 =& \text{var}(\mathit{Machine}) \cup \{\overline{Q_k}, \overline{NQ_k}, \overline{F_k}, \overline{R_k}, \overline{G^1_{2k+2}}, \overline{sG^1_{2k+2}}, \overline{R^2_{2k}},\\
& \overline{s^1_{2k}}, \ldots, \overline{s^k_{2k}}, \overline{S^1_{2k}}, \ldots, \overline{S^{k+1}_{2k}}, \overline{aG^1_{2k+2}}\}\\
& \cup \{\overline{G^2_{2k+2}}, \overline{sG^2_{2k+2}}, \overline{G^3_{2k+2}}, \overline{fG^1_{2k+2}}, \overline{fG^2_{2k+2}}, \overline{fG^3_{2k+2}}\}\\
& \cup \text{var}_1(\mathcal{G}(\overline{fG^1_{2k+2}})) \cup \text{var}_1(\mathcal{G}(\overline{fG^2_{2k+2}})) \cup \text{var}_1(\mathcal{G}(\overline{fG^3_{2k+2}}))\\
& \cup \text{var}_1(\mathfrak{G}(1/2)),\\
\Phi_2 =& \text{var}_2(\mathfrak{G}(1/2)),\\
\Phi_a =& \{A_i\} \text{ or } \{CA_i\},\\
\Phi_b =& \{B_i\} \text{ or } \{CB_i\},\\
\Phi_c =& \{r_i\}.
\end{aligned}$$

As there is a polynomial number of these, it remains to check the size of $\boldsymbol{\sigma}$.

Player One is playing *Abstain*, the equilibrium strategy in $\mathfrak{G}(1/2)$, and every other variable to *false*.[11] Player Two is playing the equilibrium strategy in $\mathfrak{G}(1/2)$. The proposition-players in A and B randomise over setting q_i to *true* and *false* equally, giving each a total of two strategies in their support. The clause-players have at most k strategies in the support of each, and the players in C have two each. This completes the proof. □

Our second translation is from Feigenbaum et al. (1995), where the authors show that deciding whether the value of a certain zero-sum game is beyond a certain threshold is EXP-complete.

Theorem 4.10. DVALUE *for Boolean games is* EXP*-complete.*

Proof. The problem is in EXP because the normal form version is in P—smply expand the Boolean game into its normal form and run the polynomial time algorithm.

For hardness, consider (M, K, w) where M is a deterministic Turing machine, K a computation bound and w an input word. Feigenbaum et al. (1995) demonstrate how to construct, in polynomial time, a game $G(M, K, w)$ and a rational v such that the value of $G(M, K, w)$ is at least v if and only if M accepts w in at most K steps. What we will show is that we can construct a Boolean game with the same value as $G(M, K, w)$, the size of which is polynomial in $|M|, |K|$ and $|w|$. This will prove the theorem.

For the sake of self-containment, we will replicate the construction of Feigenbaum et al. (1995) below. The reader well acquainted with the construction can skip straight to Lemma 4.11.

We wish to associate with M a set of Horn clauses S over a set of propositional variables P such that M accepts w in at most K steps if and only if there exists an assignment to the variables in P satisfying every clause in S.

[11] Note that Player One does not have to play the equilibrium strategy in the $\mathcal{G}'$ games as he is playing *Abstain*, and hence his assignment to those variables does not matter.

P contains propositions of the form $p[t, l, a]$ and $p[t, l, (s, a)]$. The intended interpretation of $p[t, l, a]$ is that cell l contains symbol a at computation step t. Without loss of generality, we are working with a binary alphabet, so a is 0, 1, or the blank tape symbol $\llcorner$. The intended interpretation of $p[t, l, (s, a)]$ is that, in addition to the above, the head is over cell l and in state s.

S contains three types of clauses. The first type describe the initial configuration of the machine. These consist of $p[0, 0, (q_1, w[0])]$, $p[0, i, w[i]]$ for $1 \leq i < |w|$, $p[0, i, \llcorner]$ for $i \geq |w|$, and $\neg p[0, x, y]$ for every x, y not conforming to the preceding types. The second type describe the transition rules of the machine. These take the form:

$$(p[t, l-1, \varsigma_1] \wedge p[t, l, \varsigma_2] \wedge p[t, l+1, \varsigma_3]) \rightarrow p[t+1, l, \varsigma],$$
$$(p[t, l-1, \varsigma_1] \wedge p[t, l, \varsigma_2] \wedge p[t, l+1, \varsigma_3]) \rightarrow \neg p[t+1, l, \varsigma'],$$

choosing appropriate values for $\varsigma_1, \varsigma_2, \varsigma_3$ and $\varsigma' \neq \varsigma$. The last clause is $p[K-1, 0, (q_f, 0)]$, asserting that M accepts at time K.

For convenience, we will treat every negative clause as a clause with a consequent of *false*. That is, instead of $\neg p[0, x, y]$ and $(p_1 \wedge p_2 \wedge p_3) \rightarrow \neg q$ we will have the clauses $p[0, x, y] \rightarrow \mathit{false}$ and $(p_1 \wedge p_2 \wedge p_3 \wedge q) \rightarrow \mathit{false}$. This will mean a clause can have anywhere between 0 and 4 proposition in the tail – a true initial condition, a false initial condition, a positive boundary rule, a positive rule, a negative rule.

The game $G(M, K, w)$ proceeds by letting Player One choose $r \in P$ and Player Two an element $C \in S, C = \bigwedge p_i \rightarrow q$. Letting $R \subseteq P$ be the set of variables made true in the unique run of M on w, the payoff to One is as follows:

$$H(r, C) = \begin{cases} 1 + \alpha, & r = q, \\ -1 + \alpha, & r = p_i, \\ \alpha, & \text{otherwise.} \end{cases}$$

In the above, $\alpha = \frac{j-1}{|R|}$, where $0 \leq j \leq 4$ is the number of literals in the antecedent of C. To simplify matters we assume that $|R| = 2^{2k}$, i.e. we consider the first 2^k computation steps and 2^k tape cells. This can be done by endowing the machine with a "do nothing" transition as before.

Lemma 4.11 (Feigenbaum et al., 1995)**.** *M accepts w in at most K steps if and only if the value of $G(M, K, w)$ is at least 0.*

This is where we seek to hijack the rest of their proof. If we can demonstrate that we can build a Boolean game with the same value as $G(M, K, w)$ in polynomial time, we are done. The first hurdle we face is that we clearly cannot – the payoffs of a Boolean game are restricted to $\{ 0, 1 \}$, so we cannot implement H. As such we adjust the payoffs as follows:

$$H'(r, C) = \begin{cases} 3/4 + \alpha/4, & r = q, \\ 1/4 + \alpha/4, & r = p_i, \\ 1/2 + \alpha/4, & \text{otherwise.} \end{cases}$$

H' is obtained via the affine transformation $x \mapsto {}^{x}/_{4} + {}^{1}/_{2}$, so we can conclude that a game with these payoffs has a value of at least $v = {}^{1}/_{2}$ if and only if M accepts w in at most K steps.

Note that now the payoffs are within $[\frac{1}{4} - \frac{1}{4 \cdot 2^{2k}}, \frac{3}{4} + \frac{3}{4 \cdot 2^{2k}}]$, and thus for $k \geq 1$ they are contained in $[0, 1]$, the range of feasible expected utilities in a Boolean game.

Now, suppose we can find sets of variables Φ'_1 and Φ'_2, and fifteen (polynomial size) formulae $\varphi^j_{r=q}$, $\varphi^j_{r=p_i}$, $\varphi^j_{\neq}$, $j \in \{0, 1, 2, 3, 4\}$, with the following properties:

- Every truth assignment to Φ'_1 corresponds to a choice of $r \in P$.
- Every truth assignment to Φ'_2 corresponds to a choice of $C \in S$.
- $\varphi^j_{r=q}$ is true if and only if C has j elements in the tail and r is equal to the head of C. Mutatis mutandis, for the other φ.

We claim that at that point we are done, and the following is the game we desire:

$$\begin{aligned}
\Phi_1 =& \Phi'_1 \cup \mathrm{var}_1(\mathfrak{G}({}^{3}/_{4} + {}^{\alpha}/_{4})) \cup \mathrm{var}_1(\mathfrak{G}({}^{1}/_{4} + {}^{\alpha}/_{4})) \cup \mathrm{var}_1(\mathfrak{G}({}^{1}/_{2} + {}^{\alpha}/_{4})),\\
\Phi_2 =& \Phi'_2 \cup \mathrm{var}_2(\mathfrak{G}({}^{3}/_{4} + {}^{\alpha}/_{4})) \cup \mathrm{var}_2(\mathfrak{G}({}^{1}/_{4} + {}^{\alpha}/_{4})) \cup \mathrm{var}_2(\mathfrak{G}({}^{1}/_{2} + {}^{\alpha}/_{4})),\\
\gamma_1 =& \bigvee_{j \leq 4} \left(\varphi^j_{r=q} \wedge \gamma_1(\mathfrak{G}({}^{3}/_{4} + {}^{\alpha}/_{4}))\right) \vee \bigvee_{j \leq 4} \left(\varphi^j_{r=p_i} \wedge \gamma_1(\mathfrak{G}({}^{1}/_{4} + {}^{\alpha}/_{4}))\right)\\
& \vee \bigvee_{j \leq 4} \left(\varphi^j_{\neq} \wedge \gamma_1(\mathfrak{G}({}^{1}/_{2} + {}^{\alpha}/_{4}))\right).
\end{aligned}$$

By Lemma 3.26, the subgames can be constructed in polynomial time. As such, all that remains is to provide Φ'_1, Φ'_2, and $\varphi^j_{r=q}$, $\varphi^j_{r=p_i}$, $\varphi^j_{\neq}$, for $j \in \{0, 1, 2, 3, 4\}$.

We start with Player One:

$$\Phi'_1 = \{\mathit{Zero}^1, \mathit{One}^1\} \cup \{\overline{\mathit{Time}^1_k}\} \cup \{\overline{\mathit{Tape}^1_k}\} \cup \{\overline{\mathit{State}^1_{|Q|}}\}.$$

We map a truth assignment to Φ'_1 to $r \in P$ in the following way:

- If both Zero^1 and One^1 are true, or more than one of $\{\overline{\mathit{State}^1_{|Q|}}\}$ is true, then the assignment is treated as $p[0, 0, 0]$.[12]
- If State^1_m and, without loss of generality, Zero^1 is true then the assignment is treated as $p[[\![\overline{\mathit{Time}^1_k}]\!], [\![\overline{\mathit{Tape}^1_k}]\!], (q_m, 0)]$.
- If all the state variables are false and, say, Zero^1 is true, then the assignment is treated as $p[[\![\overline{\mathit{Time}^1_k}]\!], [\![\overline{\mathit{Tape}^1_k}]\!], 0]$.

[12] In the previous section we punished a player for making an illegal move by having them lose the game. However, we cannot do this here as if both players play illegally the game would fail to be zero-sum. Instead, we pick an arbitrary legal move for them, in this case $p[0, 0, 0]$.

For Player Two we have a larger set of variables:

$$\begin{aligned}\Phi_2' =&\{pZero^2, pOne^2, Zero^2, One^2, sZero^2, sOne^2, nZero^2, nOne^2\}\\ &\cup \{\overline{Time_k^2}\} \cup \{\overline{Tape_2^k}\} \cup \{\overline{nState_{|Q|}^2}\} \cup \{\overline{pState_{|Q|}^2}\} \cup \{\overline{State_{|Q|}^2}\}\\ &\cup \{\overline{sState_{|Q|}^2}\} \cup \{Negative, Accept\}.\end{aligned}$$

We map a truth assignment to Φ_2' to $C \in S$ in the following way:

- An illegal configuration is mapped to $p[K-1, 0, (q_f, 0)]$.
- The $\overline{Time_k^2}$, $\overline{Tape_k^2}$ variables refer to the step/cell specified by the consequent – this uniquely defines the step/cell values of the tail propositions. Thus, if $[\![\overline{Time_k^2}]\!] = 0$, then the clause is treated as an initial configuration clause, $p[0, [\![\overline{Tape_k^2}]\!], w[i]]$. If $[\![\overline{Tape_k^2}]\!]$ is 0 or $2^k - 1$, then the clause is a boundary case and hence has only two propositions in the tail. If *Negative* is set to *true*, then the assignment is mapped to a negative clause. With this in mind, the contents of $\bigwedge p_i \rightarrow q$ are derived from the assignment in the natural way: recall, $nOne^2$ refers to the contents of the *n*ext computation step, or the head of the clause. $sOne^2$ and $pOne^2$ are the *s*uccessor and *p*redecessor of the central literal in the tail, and hence refer to the right and left cell.
- *Accept* is a special variable used to mark the fact that Player Two is playing the accepting clause, $p[K-1, 0, (q_f, 0)]$. If *Accept* is set to *true*, and Player Two plays $[\![\overline{Time_k^2}]\!] = K-1$, $[\![\overline{Tape_k^2}]\!] = 0$, $nZero^2$ and $nState_{accept}^2$, the assignment is treated as $p[K-1, 0, (q_f, 0)]$. [13]

At this point the reader should convince themselves that the mapping defined above does, in fact, allow Player One to specify every proposition in P and Player Two every clause in S.

Let us now turn to $\varphi_{r=q}^j$. We will deal with $j = 0$ and $j = 3$. The case of $j = 2$ is obtained from $j = 3$ by changing the appropriate cell index and $j = 1, 4$ is simply *false*, as these are negative clauses and Player One is incapable of guessing the consequent in that instance.

For $j = 0$ there are three possibilities to consider. Player Two may have correctly specified a positive initial condition, the accepting clause, or played an illegal configuration. Recall that a negative initial condition clause is treated as $q \rightarrow false$, and thus falls under $j = 1$. We also need not consider Player One playing an illegal configuration, as $p[0, 0, 0]$ cannot appear in the head of any clause, and hence cannot satisfy $\varphi_{r=q}^j$.

$$\varphi_{r=q}^0 = Init \vee Final \vee Illegal_i.$$

[13] This is technically redundant: Player Two could specify the accepting clause by playing any illegal assignment, but for the purposes of transparency we do not wish to make illegal play a necessary aspect of the game.

Init requires that $\overline{Time_k^2}$ encodes 0; the state variables are false unless $\overline{Tape_k^2}$ encodes 0, in which case only $State_1^2$ is true; and if $\overline{Tape_k^2}$ encodes j then the $nZero^2, nOne^2$ variables are played in accordance with $w[j]$. *Negative* and *Accept* are both false. Player One plays his time, tape variables such that they encode the same numbers as Player Two's, and likewise the players agree on the state and content variables.

Init can be broken down into a correctness and a matching requirement.

$$Init = Init_c \wedge MatchHead.$$

Line by line, the formula below reads: if the chosen cell is not 0, the head is not over the cell. If the chosen cell is 0, the head is over the cell and in state q_0. If the chosen cell is $i < |w|$, then Player Two sets $w[i] \in \{nZero^2 \wedge \neg nOne^2, nOne^2 \wedge \neg nZero^2\}$ to *true*, depending on the bit of w. If the chosen cell is $i \geq |w|$, then the cell is blank. As the clause is neither accepting nor negative, both those variables are set to *false*.

$$\begin{aligned} Init_c = &\big(\neg \boldsymbol{Equal}(\ulcorner \mathbf{0} \urcorner; \overline{Tape_k^2}) \rightarrow \boldsymbol{NoneOf}(\overline{nState_{|Q|}^2})\big) \\ &\wedge \Big(\boldsymbol{Equal}(\ulcorner \mathbf{0} \urcorner; \overline{Tape_k^2}) \rightarrow \big(nState_1^2 \wedge \boldsymbol{OneOf}(\overline{nState_{|Q|}^2})\big)\Big) \\ &\wedge \bigwedge_{0 \leq i < |w|} \big(\boldsymbol{Equal}(\ulcorner \boldsymbol{i} \urcorner; \overline{Tape_k^2}) \rightarrow w[i]\big) \\ &\wedge \big(\neg \boldsymbol{Less}(\overline{Tape_k^2}; \ulcorner |\boldsymbol{w}| \urcorner) \rightarrow (\neg nZero^2 \wedge \neg nOne^2)\big) \\ &\wedge \neg Accept \wedge \neg Negative. \end{aligned}$$

Note that the third line expands into $|w|$ conjuncts, so the formula is of polynomial size.

MatchHead states that Player One specifies the same proposition as is in the head of Player Two's clause. That is, the cell is the same, the computation step is the same, the tape contents are the same and the machine state is the same. This is a general term that we will reuse in other subformulae.

$$\begin{aligned} MatchHead = &\boldsymbol{Equal}(\overline{Tape_k^1}; \overline{Tape_k^2}) \wedge \boldsymbol{Equal}(\overline{Time_k^1}; \overline{Time_k^2}) \\ &\wedge (Zero^1 \leftrightarrow nZero^2) \wedge (One^1 \leftrightarrow nOne^2) \\ &\wedge \bigwedge_{1 \leq i \leq |Q|} (State_i^1 \leftrightarrow nState_i^2). \end{aligned}$$

Final likewise has a correctness and a matching requirement. The correctness requirement asks that Player Two set *Accept* to *true* and specify $p[K - 1, 0, (q_f, 0)]$. The matching requirement we can reuse from the preceding case.

$$\begin{aligned} Final = &Final_c \wedge MatchHead, \\ Final_c = &Accept \wedge \neg Negative \wedge nState_{accept}^2 \wedge \boldsymbol{OneOf}(\overline{nState_{|Q|}^2}) \\ &\wedge \boldsymbol{Equal}(\ulcorner \boldsymbol{K - 1} \urcorner; \overline{Time_k^2}) \wedge \boldsymbol{Equal}(\ulcorner \mathbf{0} \urcorner; \overline{Tape_k^2}) \\ &\wedge nZero^2 \wedge \neg nOne^2. \end{aligned}$$

$\mathit{Illegal}_i$ says that Player Two names an illegal configuration and Player One names $p[K-1,0,(q_f,0)]$.

$$\mathit{Illegal}_i = \mathit{TwoIllegal} \wedge \mathit{OneFinal}.$$

Let us list everything that could constitute an illegal assignment for Player Two:

1. The presence of both 1 and 0 in any specified cell.
2. The presence of more than one state in any cell.
3. The presence of the head in more than one cell in the tail.
4. Player Two names computation step 0, but supplies an incorrect initial configuration of the machine.
5. Player Two plays *Accept* and does not correctly describe $p[K-1,0,(q_f,0)]$.
6. Player Two names computation step ≥ 1 and supplies a clause inconsistent with the transition rules of the machine.

We will introduce a formula for each item. *TwoIllegal* will be the disjunction of these formulae.

$$\begin{aligned}
1 =& (\mathit{pZero}^2 \wedge \mathit{pOne}^2) \vee (\mathit{Zero}^2 \wedge \mathit{One}^2) \vee (\mathit{sZero}^2 \wedge \mathit{sOne}^2) \\
& \vee (\mathit{nZero}^2 \wedge \mathit{nOne}^2).
\end{aligned}$$

$$\begin{aligned}
2 =& \big(\neg \boldsymbol{OneOf}(\overline{\mathit{nState}^2_{|Q|}}) \wedge \neg \boldsymbol{NoneOf}(\overline{\mathit{nState}^2_{|Q|}})\big) \\
& \vee \big(\neg \boldsymbol{OneOf}(\overline{\mathit{pState}^2_{|Q|}}) \wedge \neg \boldsymbol{NoneOf}(\overline{\mathit{pState}^2_{|Q|}})\big) \\
& \vee \big(\neg \boldsymbol{OneOf}(\overline{\mathit{State}^2_{|Q|}}) \wedge \neg \boldsymbol{NoneOf}(\overline{\mathit{State}^2_{|Q|}})\big) \\
& \vee \big(\neg \boldsymbol{OneOf}(\overline{\mathit{pState}^2_{|Q|}}) \wedge \neg \boldsymbol{NoneOf}(\overline{\mathit{sState}^2_{|Q|}})\big).
\end{aligned}$$

$$\begin{aligned}
3 =& \big(\neg \boldsymbol{NoneOf}(\overline{\mathit{pState}^2_{|Q|}}) \wedge \neg \boldsymbol{NoneOf}(\overline{\mathit{State}^2_{|Q|}})\big) \\
& \vee \big(\neg \boldsymbol{NoneOf}(\overline{\mathit{pState}^2_{|Q|}}) \wedge \neg \boldsymbol{NoneOf}(\overline{\mathit{sState}^2_{|Q|}})\big) \\
& \vee \big(\neg \boldsymbol{NoneOf}(\overline{\mathit{State}^2_{|Q|}}) \wedge \neg \boldsymbol{NoneOf}(\overline{\mathit{sState}^2_{|Q|}})\big).
\end{aligned}$$

$$\begin{aligned}
4 =& \boldsymbol{Equal}(\ulcorner \mathbf{0} \urcorner; \overline{\mathit{Time}^2_k}) \\
& \wedge \Big(\Big(\neg \mathit{Negative} \wedge \big(\bigvee_{0 \leq i < |w|} (\boldsymbol{Equal}(\ulcorner \boldsymbol{i} \urcorner; \overline{\mathit{Tape}^2_k}) \wedge \neg w[i]) \\
& \qquad \vee (\neg \boldsymbol{Less}(\overline{\mathit{Tape}^2_k}; \ulcorner |\boldsymbol{w}| \urcorner) \wedge (\mathit{nZero}^2 \vee \mathit{nOne}^2))\big)\Big) \\
& \quad \vee \Big(\mathit{Negative} \wedge \big(\bigvee_{0 \leq i < |w|} (\boldsymbol{Equal}(\ulcorner \boldsymbol{i} \urcorner; \overline{\mathit{Tape}^2_k}) \wedge w[i]) \\
& \qquad \vee (\neg \boldsymbol{Less}(\overline{\mathit{Tape}^2_k}; \ulcorner |\boldsymbol{w}| \urcorner) \wedge (\neg \mathit{nZero}^2 \wedge \neg \mathit{nOne}^2))\big)\Big)\Big).
\end{aligned}$$

$$\begin{aligned}
5 =& \mathit{Accept} \\
& \wedge \big(\neg \boldsymbol{Equal}(\ulcorner \boldsymbol{K-1} \urcorner; \overline{\mathit{Time}^2_k}) \vee \neg \boldsymbol{Equal}(\ulcorner \mathbf{0} \urcorner; \overline{\mathit{Tape}^2_k})
\end{aligned}$$

$$\vee \neg nState^2_{accept} \vee \neg nZero^2\big).$$

The last formula we will not provide in its entirety. Its general form is a disjunction:

$$6 = \neg \bigvee_{Rule \in M} Rule.$$

That is, we check whether some rule is inconsistent with the clause. A difficulty arises because a rule of the form $(q_i, \varsigma) \to (q_j, D, \varsigma')$ manifests itself in as many as 24 different Horn clauses—boundary cases and locations of the head. Of course 24 is a constant, so as far as our proof goes there is no problem in introducing that many terms into the disjunction for every rule of the machine, but doing so could well double the size of the present document. Instead we will give a concrete example of one specific case: the rule $(q_3, 0) \to (q_4, R, 1)$ where the head is initially in the middle cell and the middle is neither 0 nor $2^k - 1$:

$$\begin{aligned}&\boldsymbol{NoneOf}(\overline{nState^2_{|Q|}}) \wedge State^2_3 \wedge \neg \boldsymbol{Equal}(\ulcorner \mathbf{0} \urcorner; \overline{Tape^2_k})\\ &\wedge \neg \boldsymbol{Equal}(\ulcorner \mathbf{2^k - 1} \urcorner; \overline{Tape^2_k}) \wedge \neg \boldsymbol{Equal}(\ulcorner \mathbf{0} \urcorner; \overline{Time^2_k}) \wedge Zero^2 \wedge nOne^2.\end{aligned}$$

We will also need to introduce "negative rules" to correspond to what the machine does not do. These will be treated in a similar way, all that needs to be mentioned is that for each $(q_i, \varsigma) \to (q_j, D, \varsigma')$ there will be only polynomially (of order $O(|Q| \cdot 2 \cdot |\Sigma|)$) many $(q_i, \varsigma) \to \neg(q'_j, D', \tau)$.

So much for $j = 0$. Let us turn to $j = 3$.

This turns out to be a lot easier as we have already done much of the grunt work. All we need is for Player Two to name a step ≥ 1, a cell ≥ 1 and $< 2^k - 1$, a correct configuration, and for Player One to guess the head.

$$\begin{aligned}\varphi^3_{r=q} =&\neg \boldsymbol{Equal}(\ulcorner \mathbf{0} \urcorner; \overline{Time^2_k}) \wedge \neg \boldsymbol{Equal}(\ulcorner \mathbf{0} \urcorner; \overline{Tape^2_k})\\ &\wedge \neg \boldsymbol{Equal}(\ulcorner \mathbf{2^k - 1} \urcorner; \overline{Tape^2_k}) \wedge \neg TwoIllegal \wedge MatchHead.\end{aligned}$$

Next up is $\varphi^j_{r=p_i}$. We will deal with $j = 4$. There is no case for $j = 0$, and $j = 1$, $j = 2$, $j = 3$ can be easily obtained from $j = 4$.

Let us start by introducing the formulae checking for Player One guessing the tail. We have already seen *MatchHead*, which is applicable in the case of $j = 4$ because we treat negative clauses as $(\bigwedge p_i \wedge q) \to false$. The others are built similarly.

$$\begin{aligned}MatchLeft =&\boldsymbol{Succ}(\overline{Tape^2_k}; \overline{Tape^1_k}) \wedge \boldsymbol{Succ}(\overline{Time^1_k}; \overline{Time^2_k})\\ &\wedge (Zero^1 \leftrightarrow pZero^2) \wedge (One^1 \leftrightarrow pOne^2)\\ &\wedge \bigwedge_{1 \leq i \leq |Q|} (State^1_i \leftrightarrow pState^2_i).\end{aligned}$$

$$MatchCentre =\boldsymbol{Equal}(\overline{Tape^1_k}; \overline{Tape^2_k}) \wedge \boldsymbol{Succ}(\overline{Time^1_k}; \overline{Time^2_k})$$

$$\wedge\,(\mathit{Zero}^1 \leftrightarrow \mathit{Zero}^2) \wedge (\mathit{One}^1 \leftrightarrow \mathit{One}^2)$$
$$\wedge \bigwedge_{1\le i\le |Q|} (\mathit{State}_i^1 \leftrightarrow \mathit{State}_i^2).$$

$$\begin{aligned}\mathit{MatchRight} =& \boldsymbol{Succ}(\overline{\mathit{Tape}_k^1}; \overline{\mathit{Tape}_k^2}) \wedge \boldsymbol{Succ}(\overline{\mathit{Time}_k^1}; \overline{\mathit{Time}_k^2})\\ &\wedge (\mathit{Zero}^1 \leftrightarrow \mathit{sZero}^2) \wedge (\mathit{One}^1 \leftrightarrow \mathit{sOne}^2)\\ &\wedge \bigwedge_{1\le i\le |Q|} (\mathit{State}_i^1 \leftrightarrow \mathit{sState}_i^2).\end{aligned}$$

Note that this is all we need to capture the case where Player One plays legally:

$$\begin{aligned}\varphi^4_{r=p_i} =& \mathit{Illegal}_{r=p_i} \vee \Big(\neg\boldsymbol{Equal}(\ulcorner\mathbf{0}\urcorner; \overline{\mathit{Time}_k^2}) \wedge \neg\boldsymbol{Equal}(\ulcorner\mathbf{0}\urcorner; \overline{\mathit{Tape}_k^2})\\ &\wedge \neg\boldsymbol{Equal}(\ulcorner\mathbf{2^k - 1}\urcorner; \overline{\mathit{Tape}_k^2}) \wedge \neg \mathit{TwoIllegal} \wedge \mathit{Negative}\\ &\wedge (\mathit{MatchHead} \vee \mathit{MatchLeft} \vee \mathit{MatchCentre} \vee \mathit{MatchRight})\Big).\end{aligned}$$

$\mathit{Illegal}_{r=p_i}$ is also relatively simple. Player One must make a violation, and Player Two needs to play a legal clause with $p[0,0,0]$ in the tail.

$$\begin{aligned}\mathit{Illegal}_{r=p_i} =& \mathit{OneIllegal} \wedge \neg \mathit{TwoIllegal}\\ &\wedge \Big(\big(\mathit{pZero}^2 \wedge \boldsymbol{NoneOf}(\overline{\mathit{pState}^2_{|Q|}})\\ &\quad \wedge \boldsymbol{Equal}(\ulcorner\mathbf{1}\urcorner; \overline{\mathit{Time}_k^2}) \wedge \boldsymbol{Equal}(\ulcorner\mathbf{1}\urcorner; \overline{\mathit{Tape}_k^2})\big)\\ &\vee \big(\mathit{Zero}^2 \wedge \boldsymbol{NoneOf}(\overline{\mathit{State}^2_{|Q|}})\\ &\quad \wedge \boldsymbol{Equal}(\ulcorner\mathbf{1}\urcorner; \overline{\mathit{Time}_k^2}) \wedge \boldsymbol{Equal}(\ulcorner\mathbf{0}\urcorner; \overline{\mathit{Tape}_k^2})\big)\Big).\end{aligned}$$

Player One does not have a lot of creativity in how to play incorrectly:

$$\begin{aligned}\mathit{OneIllegal} =& (\mathit{One}^1 \wedge \mathit{Zero}^1)\\ &\vee \big(\neg\boldsymbol{OneOf}(\overline{\mathit{State}_k^1}) \wedge \neg\boldsymbol{NoneOf}(\overline{\mathit{State}_k^1})\big).\end{aligned}$$

Finally we come to $\varphi^j_{\neq}$, where we look at $j = 3$.

There are four cases: both players play correctly and differ in the step/cell specified. Player One plays incorrectly and Player Two plays a correct clause not covering step/cell $(0,0)$. Player Two plays incorrectly and Player One plays a correct proposition not covering the step/cell $(K-1,0)$, and of course both players could play incorrectly, in which case $p[0,0,0]$ does not cover $p[K-1,0,(q_f,0)]$.

$$\begin{aligned}\varphi^3_{\neq} =& \neg\mathit{Negative} \wedge \neg\mathit{Accept} \wedge \neg\boldsymbol{Equal}(\ulcorner\mathbf{0}\urcorner; \overline{\mathit{Time}_k^2})\\ &\wedge \neg\boldsymbol{Equal}(\ulcorner\mathbf{0}\urcorner; \overline{\mathit{Tape}_k^2}) \wedge \neg\boldsymbol{Equal}(\ulcorner\mathbf{2^k - 1}\urcorner; \overline{\mathit{Tape}_k^2})\end{aligned}$$

Problem	Comments	Complexity
RationalNash	Three players	NEXP-hard
IrrationalNash	Two players	NEXP-complete
IsNash	Unbounded players	coNP$^{\#\mathrm{P}}$-hard
DValue	Two players, zero-sum	EXP-complete

Table 5.1: Complexity of problems studied.

$$\wedge \big(\mathit{BothCorrect} \vee \mathit{TwoCorrect} \vee \mathit{OneCorrect} \vee \mathit{NoneCorrect}\big).$$

We already have all the tools we need.

$$\begin{aligned}
\mathit{BothCorrect} &= \neg \mathit{OneIllegal} \wedge \neg \mathit{TwoIllegal} \\
&\quad \wedge \neg(\mathit{MatchHead} \vee \mathit{MatchLeft} \vee \mathit{MatchCentre} \vee \mathit{MatchRight}).
\end{aligned}$$

$$\begin{aligned}
\mathit{TwoCorrect} &= \mathit{OneIllegal} \wedge \neg \mathit{TwoIllegal} \\
&\quad \wedge \neg\big(\boldsymbol{Equal}(\ulcorner \mathbf{1} \urcorner; \overline{\mathit{Time}^2_k}) \wedge \boldsymbol{Equal}(\ulcorner \mathbf{0} \urcorner; \overline{\mathit{Tape}^2_k}) \wedge \mathit{Zero}^2\big) \\
&\quad \wedge \neg\big(\boldsymbol{Equal}(\ulcorner \mathbf{1} \urcorner; \overline{\mathit{Time}^2_k}) \wedge \boldsymbol{Equal}(\ulcorner \mathbf{1} \urcorner; \overline{\mathit{Tape}^2_k}) \wedge \mathit{pZero}^2\big). \\
\mathit{OneCorrect} &= \neg \mathit{OneIllegal} \wedge \mathit{TwoIllegal} \\
&\quad \wedge \neg\big(\boldsymbol{Equal}(\ulcorner \boldsymbol{K} - \mathbf{1} \urcorner; \overline{\mathit{Time}^1_k}) \wedge \boldsymbol{Equal}(\ulcorner \mathbf{0} \urcorner; \overline{\mathit{Tape}^1_k}) \\
&\qquad \wedge \mathit{Zero}^1 \wedge \mathit{State}^1_{\mathit{accept}}\big). \\
\mathit{NoneCorrect} &= \mathit{OneIllegal} \wedge \mathit{TwoIllegal}.
\end{aligned}$$

Let us recap: we have shown that we can interpret truth assignment to Φ'_1 and Φ'_2 as choices of a clause and proposition, and that we can define polynomial-size formulae that are true just if the named proposition correctly matches the head of the clause. Using these formulae we can construct a game where Player One finds himself playing $\mathfrak{G}(v)$ whenever $H'(r, C) = v$. To see that the resulting Boolean game has the same value as the game with payoffs defined by H', one need only consider the equilibrium where Player One and Two play an equilibrium of the Feigenbaum et al. game on the $\Phi'_1 \uplus \Phi'_2$ variables, and the equilibria of the $\mathfrak{G}(v)$ games on the rest. □

5. Conclusion

In this paper we have demonstrated how a simple idea and somewhat less simple encodings can be used to translate both general game theoretic reasoning (RationalNash and IrrationalNash) as well as entire proofs (IsNash and DValue) into the framework of Boolean games. At times this approach may allow researchers to piggy back on the advances of brethren fields, rather than

having to come up with entirely novel proofs for the case of Boolean games. We summarise our results in Table 5.1.

An interesting consequence of this approach is that the picture painted thus far seems to suggest that Boolean games are every bit as difficult as circuit games, despite seemingly being a lot less general. In a sense this is reminiscent of bireducibility between formulae and circuits in NP (though of course no corresponding result has been established here). It would be interesting to see whether this will continue to be the case for other classes of problems. For example, both Fortnow et al. (2005) and Schoenebeck and Vadhan (2012) present results about the approximation complexity of circuit games, whereas we have only considered deterministic approaches. The study of like problems for Boolean games is a natural direction to pursue, especially given the fact that all the decision problems studied to date turned out to be highly intractable.

Acknowledgements

Egor Ianovski is supported by a scholarship from the Oxford-Man Institute of Quantitative Finance.

AppendixA. Omitted proofs

Lemma 3.18. *Let* $\boldsymbol{Equal}(\overline{p_m};\overline{q_m})$ *denote a term that is true if and only if* $[\![\overline{p_m}]\!] = [\![\overline{q_m}]\!]$. *I.e.,* $\nu \vDash \boldsymbol{Equal}(\overline{p_m};\overline{q_m})$ *if and only if* $[\![\overline{p_m}]\!]_\nu = [\![\overline{q_m}]\!]_\nu$.

We can construct $\boldsymbol{Equal}(\overline{p_m};\overline{q_m})$ *in time linear in* m.

Proof. Two binary integers are equal if and only if they are bitwise equal. This gives us the following:

$$\boldsymbol{Equal}(\overline{p_m};\overline{q_m}) = \bigwedge_{1\leq i\leq m}(p_i \leftrightarrow q_i).$$

□

Lemma 3.19. *Let* $\boldsymbol{Succ}(\overline{p_m};\overline{q_m})$ *denote a term that is true if and only if* $[\![\overline{p_m}]\!] + 1 = [\![\overline{q_m}]\!]$.

We can construct $\boldsymbol{Succ}(\overline{p_m};\overline{q_m})$ *in time quadratic in* m.

Proof. $2^m - 1$ does not have an m bit successor, so we first require that $\overline{p_m}$ is not all 1s:

$$\boldsymbol{Succ}(\overline{p_m};\overline{q_m}) = \neg(\bigwedge_{1\leq i\leq m} p_i) \wedge Succ'.$$

$Succ'$ will do the heavy lifting.

For intuition, recall that adding a 1 to a binary integer x can have one of two outcomes: either x ends in a 0 and this 0 is replaced with a 1, or x ends with a 1 in which case that 1 is replaced with a 0 and a carry operation occurs—which is, of course, simply the act of adding a 1 to a rightshift of x.

In other words, we have a recursive operation that replaces the rightmost consecutive block of 1s in x with 0s, the subsequent 0 with a 1, and leaves the rest of the integer unchanged.

This gives us $Succ'$:

$$\begin{aligned}
&\Big(\neg p_m \to \big(q_m \wedge \boldsymbol{Equal}(\overline{p_{m-1}}; \overline{q_{m-1}})\big)\Big) \\
\wedge&\Big(\big(p_m \wedge \neg p_{m-1}\big) \to \big(\neg q_m \wedge q_{m-1} \wedge \boldsymbol{Equal}(\overline{p_{m-2}}; \overline{q_{m-2}})\big)\Big) \\
&\vdots \\
\wedge&\Big(\big(\neg p_1 \wedge \bigwedge_{1<i\leq m} p_i\big) \to \big(q_1 \wedge \bigwedge_{1<i\leq m} \neg q_i\big)\Big).
\end{aligned}$$

For the reader who abhors ellipses, we can restate this in one line:

$$Succ' = \bigwedge_{i\leq m} \Big(\big(\neg p_i \wedge \bigwedge_{j>i} p_j\big) \to \big(\boldsymbol{Equal}(\overline{p_{i-1}}; \overline{q_{i-1}}) \wedge q_i \wedge \bigwedge_{j>i} \neg q_j\big)\Big).$$

This is quadratic in the number of variables, giving us the desired result. □

Lemma 3.20. *Let $\boldsymbol{Less}(\overline{p_m}; \overline{q_m})$ denote a term that is true under ν if and only if $[\![\overline{p_m}]\!]_\nu < [\![\overline{q_m}]\!]_\nu$.*

We can construct $\boldsymbol{Less}(\overline{p_m}; \overline{q_m})$ in time cubic in m.

Proof. Let $a[i]$ be the ith most significant bit of a.

Intuitively, if $[\![\overline{p_m}]\!] < [\![\overline{q_m}]\!]$ for two big-endian binary digits then if we read the bits of the two from left to right we will see a 0 in $[\![\overline{p_m}]\!]$ before we see one in $[\![\overline{q_m}]\!]$. That is, there exists a k such that:

$$\begin{gathered}
[\![\overline{p_k}]\!] = [\![\overline{q_k}]\!], \\
\nu \not\models p_{k+1}, \quad \nu \models q_{k+1}.
\end{gathered}$$

It follows that the first bit where the two integers differ is a 1 for $[\![\overline{q_m}]\!]$ and a 0 for $[\![\overline{p_m}]\!]$. This is clearly both necessary and sufficient.

Since there are only m possible values of k, this can be replaced by a cubic size formula that looks as follows:

$$\bigvee_{1\leq k\leq m} \boldsymbol{Succ}(\overline{p_k}; \overline{q_k}).$$

□

Lemma 3.22. *Let $\boldsymbol{Add}(\overline{p_m}; \overline{q_m}; \overline{r_m})$ denote a term that is true if and only if $[\![\overline{p_m}]\!] + [\![\overline{q_m}]\!] = [\![\overline{r_m}]\!]$.*

$\boldsymbol{Add}(\overline{p_m}; \overline{q_m}; \overline{r_m})$ can be replaced by a formula of propositional logic cubic in m.

Proof. Binary addition is easy: 0 is identity, and $1 + 1$ is 0. In other words if the summands are the same the answer is 0, if they are different the answer is 1. Thus if no carry occurred at position $i+1$, then $r_i \leftrightarrow (\neg(p_i \wedge q_i))$. If a carry has occurred, we add another 1 to this result, which gives us $r_i \leftrightarrow (p_i \wedge q_i)$.

The trick is determining whether a carry has occurred – we have no memory mechanism to store this information. However, our constraint is polynomial time, and this is sufficiently generous to allow us to explicitly verify whether a carry bit has reached i by checking every $j > i$. The necessary subformula is the following:

$$\mathit{Carry}(i) = \bigvee_{j \geq i+1} \Big((p_j \wedge q_j) \wedge \bigwedge_{i+1 \leq k < j} (p_k \vee q_k) \Big).$$

In words, $\mathit{Carry}(i)$ holds just if there exists a $j > i$ such that the bits at the jth position are both 1, and every single position between j and i has at least one of the bits being 1.

Our final formula is:

$$\begin{aligned} \mathit{Add} = \bigwedge_{i \leq m} \Big[\Big(\neg \mathit{Carry}(i) \rightarrow \big(r_i \leftrightarrow (\neg(p_i \leftrightarrow q_i)) \big) \Big) \\ \wedge \Big(\mathit{Carry}(i) \rightarrow \big(r_i \leftrightarrow (p_i \leftrightarrow q_i) \big) \Big) \Big] \\ \wedge \neg \Big((p_1 \wedge q_1) \vee \big(\mathit{Carry}(1) \wedge (p_1 \vee q_1) \big) \Big). \end{aligned}$$

The last line states that integer overflow has not occurred. This is added because we are interested in exact addition, not addition modulo 2^m.

This is cubic in m, proving the lemma. □

Lemma 3.23. *Let* $\boldsymbol{Sub}(\overline{p_m}; \overline{q_m}; \overline{r_m})$ *denote a term that is true if and only if* $[\![\overline{p_m}]\!] - [\![\overline{q_m}]\!] = [\![\overline{r_m}]\!]$.

$\boldsymbol{Sub}(\overline{p_m}; \overline{q_m}; \overline{r_m})$ *can be replaced by a formula of propositional logic cubic in* m.

Proof. Consider the following identity:

$$x - y = z \iff y + z = x.$$

It follows that:

$$\mathbf{Sub}(\overline{p_m}; \overline{q_m}; \overline{r_m}) \iff \mathbf{Add}(\overline{q_m}; \overline{r_m}; \overline{p_m}).$$

□

Lemma 4.8. *There exists a formula, Machine, such that:*

1. $\nu \vDash$ *Machine if and only if* ν *represents an accepting run of* M *on* w *in* k *steps that is correct with respect to everything except possibly the oracle responses.*

2. $|Machine|$ *is polynomial in the size of* $(M, w, 1^k)$.
3. *Machine has variables* $Q_{i,j,l}$ *and* $NQ_{i,j,l}$ *representing that the* i*th register has the symbol* q_l *or* $\neg q_l$ *in cell* j *respectively. That is, the* j*th literal of* φ_i *is* q_l *or* $\neg q_l$.
4. *Machine has variables* R_i *to denote the value of the* i*th most significant bit of the oracle's response.*
5. *Machine has variables* $\{ F_1, \ldots, F_k \}$ *to denote the value of the header register.*

Proof. The proof is essentially Cook's Theorem with some register manipulation.

Observe that the computation bound, k, is given in unary. As a consequence the $k \times k$ table representing the history of the machine (see Ianovski and Ong (2014) for details) is polynomial in the size of the input. As such we have no need to mess around with cell indices, but instead can ask Player One to describe the entire table directly.

We equip Player One with k^2 $l[i, j]$ variables for every $l \in \Sigma$, denoting that the (i, j)-entry of the computation history is l. We also provide k^2 state variables for every $q \in Q$, $q[i, j]$, denoting at once the location of the head and the state that it is in. This allows us to formulate the initial configuration of the machine:[14]

$$Initial = q_0[0,0] \wedge \Big(\bigwedge_{j<|w|} w[j][0,j] \Big) \wedge \Big(\bigwedge_{j \geq |w|} \boldsymbol{NoneOf}(\{ l[0,j] : l \in \Sigma \}) \Big).$$

Now all we need is a *Rules* term which is nothing but a giant or-statement, checking that every square in the history follows from the preceding computation step via the application of an admissible rule, to establish that the machine history is correct. We shall not consider the standard transition rules and instead focus on the ones unique to the machine at hand.

Let $Write[t, i, j, l]$ be the rule that states that the machine at time t writes symbol q_l to the jth entry of the ith clause of φ. We can express it as follows:

$$\begin{aligned} Write[t,i,j,l] = &\big((q_w[t,0] \wedge p_l[t,0] \wedge i[t,1] \wedge j[t,2]) \\ &\wedge (Q_{i,j,l} \wedge q'[t+1,0]) \big) \wedge SameTape[t]. \end{aligned}$$

Recall that we have included the integers $\{ 1, \ldots, k \}$ (of which there is a polynomial number) as primitives so there is no need to mess around with encoding numbers—we simply demand that the symbol i be written on the tape to write to the ith clause. The state is q_w, the special write state, and $p_l[t, 0]$ is a propositional symbol, which is also included as a primitive. The next state, q', can be chosen arbitrarily: the machine is nondeterministic and we can assume it jumps from q' to wherever it needs to go to resume computation.

[14] $w[j][0, j]$ is to be interpreted as $l[0, j]$, $l \in \Sigma$ being the jth entry of w.

SameTape$[t]$ asserts that the tape contents are unchanged between steps t and $t+1$, and that the head does not move. This is expressed in the obvious way:

$$\begin{aligned} \mathit{SameTape}[t] = \bigwedge_{i<k, l\in\Sigma} \big(&(l[t,i] \leftrightarrow l[t+1,i]) \\ &\wedge (\boldsymbol{NoneOf}(\{\, q[t,i] : q \in Q \,\}) \leftrightarrow \boldsymbol{NoneOf}(\{\, q[t,i] : q \in Q \,\}))\big). \end{aligned}$$

There are no more than k^4 such rules, and we analogously include the rules for writing a negative literal.

Next is the oracle query. We assume this is made from state q_o, giving us the following formula:

$$\mathit{Oracle}[t,i] = q_o[t,i] \wedge q'[t+1,i] \wedge \mathit{SameTape}.$$

There are k^2 possible cases depending on where and when the query is made.

Finally, the machine needs to be able to access the oracle results somehow. There are several ways we could handle this, a simple one is to allow the machine to request that the ith bit be written to the start of the tape by writing i at the start of the tape and entering state q_r.

$$\begin{aligned} \mathit{Read}[t,i] = &\big(q_r[t,0] \wedge i[t,0]\big) \wedge \\ &\big(q'[t+1,0] \wedge (R_i \to 1[t+1,0]) \wedge (\neg R_i \to 0[t+1,0])\big) \\ &\wedge \mathit{SameTapeExceptCellZero}[t]. \end{aligned}$$

SameTapeExceptCellZero$[t]$ is exactly what it says.

We add the usual consistency constraints such as every cell can only contain one symbol, the machine can only be in one state, and so on. We do not need to worry about what happens if the machine writes an ill-configured formula into the registers, or tries to read the oracle's response before it makes the query—we can assume the machine itself would enter an abort state in such an instance. All that remains to do is verify that Player One behaves sensibly with the variables we use later in the construction.

So far we have guaranteed that if $\mathit{Write}[t,i,j,l]$ is issued, then $Q_{i,j,l}$ is true. At present nothing prevents Player One from setting to $Q_{i,j,l}$ to *true* regardless, so we have constraints of the following form for every j,l:

$$\Big(\bigwedge_{t,i<k} \neg \mathit{Write}[t,i,j,l]\Big) \to \neg Q_{i,j,l}.$$

Analogously for the negative literals.

We also need Player One to tell us how many clauses φ has. As the internal rules of the machine guarantee that there are no gaps in between the clauses, we only need to spot the highest index of an issued write command.

$$\begin{aligned} &\bigvee_{i\le k} \bigwedge_{k\ge m>i} \Big(\neg\Big(\bigvee_{t,j,l} \mathit{Write}[t,m,j,l]\Big) \wedge \Big(\bigvee_{t,j,l} \mathit{Write}[t,i,j,l]\Big) \wedge F_i\Big) \\ &\wedge \boldsymbol{OneOf}(\overline{F_i}). \end{aligned}$$

The outer or ranges over candidates for the highest i for which φ_i is defined. For this to be true it must be the case that for all $m > i$ the write command was never issued, but that it was issued at i. In this case, F_i ought to be true.

Putting these together, the lemma is proven. □

References

Abbott, T., Kane, D., Valiant, P., 2005. On the complexity of two-player win-lose games. In: Proceedings of FOCS 2005. pp. 113 – 122.

Ågotnes, T., Harrenstein, P., van der Hoek, W., Wooldridge, M., 2013. Boolean games with epistemic goals. In: LORI. pp. 1–14.

Àlvarez, C., Gabarró, J., Serna, M., 2005a. Polynomial space suffices for deciding nash equilibria properties for extensive games with large trees,. In: Deng, X., Du, D.-Z. (Eds.), Algorithms and Computation. Vol. 3827 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 634–643.
URL http://dx.doi.org/10.1007/11602613_64

Àlvarez, C., Gabarró, J., Serna, M., 2005b. Pure Nash equilibria in games with a large number of actions. In: Jędrzejowicz, J., Szepietowski, A. (Eds.), Mathematical Foundations of Computer Science 2005. Vol. 3618 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 95–106.

Àlvarez, C., Gabarró, J., Serna, M., 2011. Equilibria problems on games: Complexity versus succinctness. Journal of Computer and System Sciences 77 (6), 1172 – 1197.
URL http://www.sciencedirect.com/science/article/pii/S002200001100002X

Bilò, V., 2007. On satisfiability games and the power of congestion games. In: Kao, M.-Y., Li, X.-Y. (Eds.), Algorithmic Aspects in Information and Management. Vol. 4508 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 231–240.

Bilò, V., Mavronicolas, M., 2011. Complexity of rational and irrational nash equilibria. In: Persiano, G. (Ed.), Algorithmic Game Theory. Vol. 6982 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 200–211.
URL http://dx.doi.org/10.1007/978-3-642-24829-0_19

Bilò, V., Mavronicolas, M., 2012. The complexity of decision problems about Nash equilibria in win-lose games. In: Serna, M. (Ed.), Algorithmic Game Theory. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 37–48.

Bonzon, E., Devred, C., Lagasquie-Schiex, M.-C., 2009a. Translation of an argumentation framework into a CP-Boolean game. In: Tools with Artificial Intelligence, 2009. ICTAI '09. 21st International Conference on. pp. 522–529.

Bonzon, E., Lagasquie-Schiex, M.-C., Lang, J., 2009b. Dependencies between players in Boolean games. Int. J. Approx. Reasoning 50 (6), 899–914.

Bonzon, E., Lagasquie-Schiex, M.-C., Lang, J., 2012. Effectivity functions and efficient coalitions in Boolean games. Synthese 187 (1), 73–103.

Bonzon, E., Lagasquie-Schiex, M.-C., Lang, J., Zanuttini, B., 2006. Boolean games revisited. In: Proceedings of ECAI 2006. pp. 265–269.

Bonzon, E., Lagasquie-Schiex, M.-C., Lang, J., Zanuttini, B., 2009c. Compact preference representation and Boolean games. Autonomous Agents and Multi-Agent Systems 18 (1), 1–35.

Campbell, J., Knight, V., 2015. On testing degeneracy of bi-matrix games. URL `http://vknight.org/unpeudemath/code/2015/06/25/on\_testing\_degeneracy\_of\_games/`

Chen, X., Teng, S.-H., 2007. The approximation complexity of win-lose games. In: In Proceedings of SODA.

Codenotti, B., Štefankovič, D., 2005. On the computational complexity of nash equilibria for $(0, 1)$ bimatrix games. Information Processing Letters 94 (3), 145 – 150.
URL `http://www.sciencedirect.com/science/article/pii/S0020019005000281`

Conitzer, V., Sandholm, T., 2008. New complexity results about nash equilibria. Games and Economic Behavior 63 (2), 621–641.

Cottle, R. W., Dantzig, G. B., 1968. Complementary pivot theory of mathematical programming. Linear Algebra and its Applications 1, 103–125.

Dantzig, G. B., 1951. A proof of the equivalence of the programming problem and the game problem. In: Koopmans (Ed.), Activity analyses of production and allocation. Wiley, New York, Ch. XX.

Daskalakis, C., Papadimitriou, C. H., 2005. Three-player games are hard. Tech. Rep. TR05-139, ECCC.

Dunne, P. E., van der Hoek, W., 2004. Representation and complexity in Boolean games. In: Alferes, J. J., Leite, J. a. (Eds.), Logics in Artificial Intelligence. Vol. 3229 of Lecture Notes in Computer Science. Springer-Verlag, pp. 347–359.

Dunne, P. E., van der Hoek, W., Kraus, S., Wooldridge, M., 2008. Cooperative Boolean games. In: AAMAS'08. pp. 1015–1022.

Dunne, P. E., Wooldridge, M., 2012. Towards tractable Boolean games. In: Proceedings of AAMAS '12. pp. 939–946.

Endriss, U., Kraus, S., Lang, J., Wooldridge, M., 2011. Designing incentives for Boolean games. In: The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1. AAMAS '11. pp. 79–86.

Feigenbaum, J., Koller, D., Shor, P., 1995. A game-theoretic classification of interactive complexity classes (extended abstract). In: Proceedings of the tenth annual IEEE conference on computational complexity. pp. 227–237.

Fortnow, L., Kabanets, V., Impagliazzo, R., Umans, C., 2005. On the complexity of succinct zero-sum games. In: IEEE Conference on Computational Complexity. pp. 323–332.

Galafassi, C., Bazzan, A., 2013. Evolving mechanisms in boolean games. In: Klusch, M., Thimm, M., Paprzycki, M. (Eds.), Multiagent System Technologies. Vol. 8076 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 73–86.
URL http://dx.doi.org/10.1007/978-3-642-40776-5_9

Gilboa, I., Zemel, E., 1989. Nash and correlated equilibria: Some complexity considerations. Games and Economic Behavior 1 (1), 80–93.

Grant, J., Kraus, S., Wooldridge, M., Zuckerman, I., 2011. Manipulating Boolean games through communication. In: Proceedings of the Twenty-Second international joint conference on Artificial Intelligence. AAAI Press, pp. 210–215.

Gutierrez, J., Harrenstein, P., Wooldridge, M., 2013. Iterated Boolean games. In: Proceedings of the Twenty Third International Joint Conference on Artificial Intelligence.

Harrenstein, P., 2004. Logic in conflict. Ph.D. thesis, Utrecht University.

Harrenstein, P., Turrini, P., Wooldridge, M., 2014. Hard and soft equilibria in boolean games. In: Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems. AAMAS '14. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 845–852.
URL http://dl.acm.org/citation.cfm?id=2615731.2615867

Harrenstein, P., Turrini, P., Wooldridge, M., 2015. Electric Boolean games: Redistribution schemes for resource-bounded agents. In: In Proceedings of the Sixteenth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2015).

Harrenstein, P., van der Hoek, W., Meyer, J.-J., Witteveen, C., 2001. Boolean games. In: Proceedings of TARK 2001. pp. 287–298.

Ianovski, E., Ong, L., 2014. EGuaranteeNash for Boolean games is NEXP-hard. In: Principles of Knowledge Representation and Reasoning: Proceedings of

the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014.
URL http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/8002

Kaplan, T. R., Dickhaut, J., 1991. A Program for Finding Nash Equilibria. Working papers, University of Minnesota, Department of Economics.
URL http://ideas.repec.org/p/wop/minnec/_004.html

Khachiyan, L. G., 1980. Polynomial algorithms in linear programming. USSR Computational Mathematics and Mathematical Physics 20 (1), 53 – 72.
URL http://www.sciencedirect.com/science/article/pii/0041555380900610

Koller, D., Megiddo, N., von Stengel, B., 1994. Fast algorithms for finding randomized strategies in game trees. In: Proceedings of the 26th ACM Symposium on Theory of Computing. pp. 750–759.

Kraus, S., Wooldridge, M., 2012. Delegating decisions in strategic settings. In: ECAI 2012. pp. 468–473.

Levit, V., Grinshpoun, T., Meisels, A., Bazzan, A. L. C., 2013. Taxation search in Boolean games. In: AAMAS. pp. 183–190.

Lukasiewicz, T., Ragone, A., 2009. Combining Boolean games with the power of ontologies for automated multi-attribute negotiation in the semantic web. In: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 02. pp. 395–402.

Marchioni, E., Wooldridge, M., 2014. Łukasiewicz games. In: Proceedings of the Thirteenth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2014).

Mavronicolas, M., Monien, B., Wagner, K. W., 2007. Weighted Boolean formula games. In: Deng, X., Graham, F. (Eds.), Internet and Network Economics. Vol. 4858 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 469–481.

Nash, J., 1951. Non-cooperative games. Annals of Mathematics 54 (2), 286–295.

Popovici, M., Dobre, C., 2012. A game-theoretic approach to cooperation in multi-agent systems. In: Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics. WIMS '12. ACM, New York, NY, USA, pp. 54:1–54:4.
URL http://doi.acm.org/10.1145/2254129.2254197

Sauro, L., Villata, S., 2013. Dependency in cooperative Boolean games. Journal of Logic and Computation 23 (2), 425–444.

Schoenebeck, G. R., Vadhan, S., 2012. The computational complexity of Nash equilibria in concisely represented games. ACM Trans. Comput. Theory 4 (2), 4:1–4:50.

Turrini, P., 2013. Endogenous Boolean games. In: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence. AAAI Press, pp. 390–396.

Valiant, L. G., 1979. The complexity of enumeration and reliability problems. SIAM Journal on Computing 8 (3), 410–421.
URL http://dx.doi.org/10.1137/0208032

von Neumann, J., 1928. Zur Theorie der Gesellschaftsspiele. Mathematische Annalen 100 (1), 295–320.

von Neumann, J., Morgenstern, O., 1947. Theory of Games and Economic Behavior, 2nd Edition. Princeton University Press.

von Stengel, B., 2007. Equilibrium computation for two-player games. In: Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V. V. (Eds.), Algorithmic Game Theory. Cambridge University Press, New York, NY, USA.

Wooldridge, M., 2012. Bad equilibria (and what to do about them). In: ECAI 2012. pp. 6–11.