Theory and Decision
https://doi.org/10.1007/s11238-018-9679-3

Robust program equilibrium

Caspar Oesterheld[1,2]

Abstract
One approach to achieving cooperation in the one-shot prisoner's dilemma is Tennenholtz's (Games Econ Behav 49(2):363–373, 2004) program equilibrium, in which the players of a game submit programs instead of strategies. These programs are then allowed to read each other's source code to decide which action to take. As shown by Tennenholtz, cooperation is played in an equilibrium of this alternative game. In particular, he proposes that the two players submit the same version of the following program: cooperate if the opponent is an exact copy of this program and defect otherwise. Neither of the two players can benefit from submitting a different program. Unfortunately, this equilibrium is fragile and unlikely to be realized in practice. We thus propose a new, simple program to achieve more robust cooperative program equilibria: cooperate with some small probability ϵ and otherwise act as the opponent acts against this program. I argue that this program is similar to the tit for tat strategy for the iterated prisoner's dilemma. Both "start" by cooperating and copy their opponent's behavior from "the last round". We then generalize this approach of turning strategies for the repeated version of a game into programs for the one-shot version of a game to other two-player games. We prove that the resulting programs inherit properties of the underlying strategy. This enables them to robustly and effectively elicit the same responses as the underlying strategy for the repeated game.

Keywords Algorithmic game theory · Program equilibrium · Nash equilibrium · Repeated games

Contents

✉ Caspar Oesterheld
caspar.oesterheld@foundational-research.org; caspar.oesterheld@duke.edu

1 Foundational Research Institute, Berlin, Germany

2 Present Address: Duke University, Durham, North Carolina, USA

Published online: 23 November 2018

1 Introduction

Much has been written about rationalizing non-Nash equilibrium play in strategic-form games. Most prominently, game theorists have discussed how cooperation may be achieved in the prisoner's dilemma, where mutual cooperation is not a Nash equilibrium but Pareto-superior to mutual defection. One of the most successful approaches is the repetition of a game, and in particular the iterated prisoner's dilemma (Axelrod 2006).

Another approach is to introduce commitment mechanisms of some sort. In this paper, we will discuss one particular commitment mechanism: Tennenholtz's (2004) program equilibrium formalism (Sect. 2.2). Here, the idea is that in place of strategies, players submit programs which compute strategies and are given access to each other's source code. The programs can then encode credible commitments, such as some version of "if you cooperate, I will cooperate".

As desired, Tennenholtz (2004, Sect. 3, Theorem 1) shows that mutual cooperation is played in a program equilibrium of the prisoner's dilemma. However, Tennenholtz' equilibrium is very fragile. Essentially, it consists of two copies of a program that cooperates if it faces an exact copy of itself (cf. McAfee 1984; Howard 1988). Even small deviations from that program break the equilibrium. Thus, achieving cooperation in this way is only realistic if the players can communicate beforehand and settle on a particular outcome.

Another persuasive critique of this trivial equilibrium is that the model of two players submitting programs is only a metaphor, anyway. In real life, the programs may instead be the result of an evolutionary process (Binmore 1988 pp. 14f.) and Tennenholtz' equilibrium is a hard to obtain by such a process. Alternatively, if we view our theory as normative rather than descriptive, we may view the programs themselves as the target audience of our recommendations. This also means that these agents will already have some form of source code—e.g., one that derives and considers the program equilibria of the game—and it is out of their realm of power to change that source code to match some common standard. However, they may still decide on some procedure for thinking about this particular problem in such a way that enables cooperation with other rationally pre-programmed agents.

Noting the fragility of Tennenholtz' proposed equilibrium, it has been proposed to achieve a more robust program equilibrium by letting the programs reason about each other (van der Hoek et al. 2013; Barasz et al. 2014; Critch 2016). For example, Barasz et al. (2014, Sect. 3) propose a program FairBot—variations of which we will see in

this paper—that cooperates if Peano arithmetic can prove that the opponent cooperates against FairBot. FairBot cooperates (via Löb's theorem) more robustly against different versions of itself. These proposals are very elegant and certainly deserve further attention. However, their benefits come at the cost of being computationally expensive.

In this paper, I thus derive a class of program equilibria that I will argue to be more practical. In the case of the prisoner's dilemma, I propose a program that cooperates with a small probability and otherwise acts as the opponent acts against itself (see Algorithm 1). Doing what the opponent does—à la FairBot—incentivizes cooperation. Cooperating with a small probability allows us to avoid infinite loops that would arise if we merely predicted and copied our opponent's action (see Algorithm 2). This approach to a robust cooperation program equilibrium in the prisoner's dilemma is described in Sect. 3.

We then go on to generalize the construction exemplified in the prisoner's dilemma (see Sect. 4). In particular, we show how strategies for the repeated version of a game can be used to construct good programs for the one-shot version of that game. We show that many of the properties of the underlying strategy of the repeated game carry over to the program for the stage game. We can thus construct "good" programs and program equilibria from "good" strategies and Nash equilibria.

2 Preliminaries

2.1 Strategic-form games

For reference, we begin by introducing some basic terminology and formalism for strategic-form games. For an introduction, see, e.g., Osborne (2004). For reasons that will become apparent later on, we limit our treatment to two-player games.

A *two-player strategic game* $G = (A_1, A_2, u_1, u_2)$ consists of two countable sets of moves A_i and for both players $i \in \{1, 2\}$ a bounded utility function $u_i : A_1 \times A_2 \to \mathbb{R}$. A *(mixed) strategy* for player i is a probability distribution π_i over A_i.

Given a *strategy profile* (π_1, π_2) the probability of an *outcome* $(a_1, a_2) \in A_1 \times A_2$ is

$$P(a_1, a_2 \mid \pi_1, \pi_2) := \pi_1(a_1) \cdot \pi_2(a_2). \tag{1}$$

The expected value for player i given that strategy profile is

$$\mathbb{E}[u_i \mid \pi_1, \pi_2] := \sum_{(a_1,a_2)\in A_1\times A_2} P(a_1, a_2 \mid \pi_1, \pi_2) \cdot u_i(a_1, a_2).$$

Note that because the utility function is bounded, the sum converges absolutely, such that the order of the action pairs does not affect the sum's value.

2.2 Program equilibrium

We now introduce the concept of program equilibrium, first proposed by Tennenholtz (2004). The main idea is to replace strategies with computer programs that are given access to each other's source code.[1] The programs then give rise to strategies.

For any game G, we first need to define the set of *program profiles* $PROG(G)$ consisting of pairs of programs. The ith entry of an element of $PROG(G)$ must be a program source code p_i that, when interpreted by a function *apply*, probabilistically map program profiles[2] onto A_i.

We require that for any program profile $(p_1, p_2) \in PROG(G)$, both programs halt. Otherwise, the profile would not give rise to a well-defined strategy. Whether p_i halts depends on the program p_{-i}, it plays against, where (in accordance with convention in game theory) $- : \{1, 2\} \to \{1, 2\} : 1 \mapsto 2, 2 \mapsto 1$ and we write $-i$ instead of $-(i)$. For example, if p_i runs $apply(p_{-i}, (p_i, p_{-i}))$, i.e., simulates the opponent, then that is fine as long as p_{-i} does not also run $apply(p_i, (p_i, p_{-i}))$, which would yield an infinite loop. To avoid this mutual dependence, we will generally require that $PROG(G) = PROG_1(G) \times PROG_2(G)$, where $PROG_i(G)$ consists of programs for player i. Methods of doing this while maintaining expressive power include hierarchies of players—e.g., higher indexed players are allowed to simulate lower indexed ones but not vice versa—hierarchies of programs—programs can only call their opponents with simpler programs as input—requiring programs to have a "plan B" if termination can otherwise not be guaranteed, or allowing each player to only start strictly less than one simulation in expectation. These methods may also be combined. In this paper, we do not assume any particular definition of $PROG(G)$. However, we assume that they can perform arbitrary computations as long as these computations are guaranteed to halt regardless of the output of the parts of the code that do depend on the opponent program. We also require that $PROG(G)$ is compatible with our constructions. We will show our constructions to be so benign in terms of infinite loops that this is not too strong of an assumption.

Given a program profile (p_1, p_2), we receive a strategy profile $(apply(p_1, (p_1, p_2)), apply(p_2, (p_1, p_2)))$. For any outcome (a_1, a_2) of G, we define

$$P(a_1, a_2 \mid p_1, p_2) := P(a_1, a_2 \mid apply(p_1, (p_1, p_2)), apply(p_2, (p_1, p_2))) \tag{2}$$

and for every player $i \in \{1, 2\}$, we define

$$\mathbb{E}[u_i \mid p_1, p_2] := \sum_{(a_1, a_2) \in A_1 \times A_2} P(a_1, a_2 \mid p_1, p_2) \cdot u_i(a_1, a_2). \tag{3}$$

[1] The equilibrium in its rudimentary form had already been proposed by McAfee (1984) and Howard (1988). At least the idea of viewing players as programs with access to each other's source code has also been discussed by, e.g., Binmore (1987, Sect. 5; 1988) and Anderlini (1990).

[2] For keeping our notation simple, we will assume that our programs receive their own source code as input in addition to their opponent's. If $PROG_i(G)$ is sufficiently powerful, then by Kleene's second recursion theorem, programs could also refer to their own source code without receiving it as an input (Cutland 1980, ch. 11).

For player i, we define the (set-valued) best response function as

$$B_i(p_{-i}) = \arg\max_{p_i \in PROG_i(G)} \mathbb{E}\left[u_i \mid p_i, p_{-i}\right].$$

A program profile (p_1, p_2) is a *(weak) program equilibrium of* G if for $i \in \{1, 2\}$ it is $p_i \in B_i(p_{-i})$.

2.3 Repeated games

Our construction will involve strategies for the repeated version of a two-player game. Thus, for any game G, we define G_ϵ to be the repetition of G with a probability of $\epsilon \in (0, 1]$ of ending after each round. Both players of G_ϵ will be informed only of the last move of their opponent. This differs from the more typical assumption that players have access to the entire history of past moves. We will later see why this deviation is necessary. A strategy π_i for player i non-deterministically maps opponent moves or the information of the lack thereof onto a move

$$\pi_i : \{0\} \cup A_{-i} \rightsquigarrow A_i.$$

Thus, for $a \in A_i, b \in A_{-i}$, $\pi_i(b, a) := \pi_i(b)(a)$ denotes the probability of choosing a given that the opponent played b in the previous round and $\pi_i(0, a) := \pi_i(0)(a)$ denotes the probability of choosing a in the first round. We call a strategy π_i *stationary* if for all $a \in A_i$, $\pi_i(b, a)$ is constant with respect to $b \in \{0\} \cup A_{-i}$. If π_i is stationary, we write $\pi_i(a) := \pi_i(b, a)$. The probability that the game follows a complete history of moves $h = a_0 b_0 a_1 b_1 \cdots a_n b_n$ and then ends is

$$P(h \mid (\pi_1, \pi_2)) := \pi_1(0, a_0)\pi_2(0, b_0)\epsilon(1-\epsilon)^n \prod_{i=1}^{n} \pi_1(b_{i-1}, a_i)\pi_2(a_{i-1}, b_i). \quad (4)$$

Note that the moves in the history always come in pairs $a_i b_i$ which are chosen "simultaneously" in response to b_{i-1} and a_{i-1}, respectively. The expected value for player i given the strategy profile (π_1, π_2) is

$$\mathbb{E}[u_i \mid \pi_1, \pi_2] := \sum_{h \in (A_1 \cdot A_2)^+} P(h \mid \pi_1, \pi_2) \cdot u_i(h), \quad (5)$$

where $(A_1 \cdot A_2)^+$ is the set of all histories and

$$u_i(a_0 b_0 a_1 b_1 \cdots a_n b_n) := \sum_{i=0}^{n} u_i(a_i, b_i). \quad (6)$$

The lax unordered summation in Eq. 5 is, again, unproblematic because of the absolute convergence of the series, which is a direct consequence of the proof of Lemma 1. Note how the organization of the history into pairs of moves allows us to apply the utility function of the stage game in Eq. 6.

For player i, we define the set-valued best response function as

$$B_i(\pi_{-i}) = \arg\max_{\pi_i : \{0\} \cup A_{-i} \rightsquigarrow A_i} \mathbb{E}\left[u_i \mid \pi_i, \pi_{-i}\right].$$

Analogously, $B_i^c(\pi_{-i})$ is the set of responses to π_{-i} that are best among the computable ones, $B_i^s(\pi_{-i})$ the set of responses to π_{-i} that are best among the stationary ones, and $B_i^{s,c}(\pi_{-i})$ the set of responses to π_{-i} that are best among stationary computable strategies. A strategy profile (π_1, π_2) is a *(weak) Nash equilibrium of* G_ϵ if for $i \in \{1, 2\}$ it is $\pi_i \in B_i(\pi_{-i})$.

We now prove a few lemmas that we will need later on. First, we have suggestively called the values $P(h)$ probabilities, but we have not shown them to satisfy, say, Kolmogorov's axioms. Additivity is not an issue, because we have only defined the probability for atomic events and non-negativity is obvious from the definition. However, we will also need the fact that the numbers we have called probabilities indeed sum to 1, which requires a few lines to prove.

Lemma 1 *Let G_ϵ be a repeated game and π_1, π_2 be strategies for that game. Then*

$$\sum_{h \in (A_1 \cdot A_2)^+} P(h \mid \pi_1, \pi_2) = 1.$$

Proof

$$\begin{aligned}
\sum_h P(h \mid \pi_1, \pi_2) &\underset{\text{Eq. 4}}{=} \sum_{a_0 b_0 \cdots a_n b_n \in (A_1 \cdot A_2)^+} \pi_1(0, a_0)\pi_2(0, b_0)\epsilon(1-\epsilon)^n \\
&\quad \cdot \prod_{i=1}^{n} \pi_1(b_{i-1}, a_i)\pi_2(a_{i-1}, b_i) \\
&\underset{\substack{\text{absolute} \\ \text{convergence}}}{=} \epsilon \sum_{n=0}^{\infty} (1-\epsilon)^n \sum_{a_0 b_0 \cdots a_n b_n \in (A_1 A_2)^{n+1}} \pi_1(0, a_0)\pi_2(0, b_0) \prod_{i=1}^{n} \pi_1(b_{i-1}, a_i)\pi_2(a_{i-1}, b_i) \\
&= \epsilon \sum_{n=0}^{\infty} (1-\epsilon)^n \sum_{a_0 b_0 \in A_1 A_2} \pi_1(0, a_0)\pi_2(0, b_0) \sum_{a_1 b_1 \in A_1 A_2} \pi_1(b_0, a_1)\pi_2(a_0, b_1) \ldots \\
&\quad \cdot \sum_{a_n b_n \in A_1 A_2} \pi_1(b_{n-1}, a_n)\pi_2(a_{n-1}, b_n) \\
&= \epsilon \sum_{n=0}^{\infty} (1-\epsilon)^n \\
&\underset{\substack{\text{sum of geo-} \\ \text{metric series}}}{=} 1
\end{aligned}$$

For seeing why the second-to-last equation is true, notice that the inner-most sum is 1. Thus, the next sum is 1 as well, and so on. Since the ordering in the right-hand side of the first line is lax, and because only the second line is known to converge absolutely,

the re-ordering is best understood from right to left. The last step uses the well-known formula $\sum_{k=0}^{\infty} x^k = 1/(1-x)$ for the geometric series. □

For any game G_ϵ, $k \in \mathbb{N}_+$, $a \in A_1$, $b \in A_2$ and strategies π_1 and π_2 for G_ϵ, we define

$$P_{k,G_\epsilon}(a,b \mid \pi_1,\pi_2) := (1-\epsilon)^k \sum_{a_0b_0\cdots a_{k-1}b_{k-1}\in(A_1A_2)^k} \pi_1(b_{k-1},a)\pi_2(a_{k-1},b)\pi_1(0,a_0)\pi_2(0,b_0) \cdot \prod_{j=1}^{k-1} \pi_1(b_{j-1},a_j)\pi_2(a_{j-1},b_j). \tag{7}$$

For $k = 0$, we define

$$P_{0,G_\epsilon}(a,b \mid \pi_1,\pi_2) := \pi_1(0,a) \cdot \pi_2(0,b).$$

Intuitively speaking, $P_{k,G_\epsilon}(a,b \mid \pi_1,\pi_2)$ is the probability of reaching at least round k and that (a,b) is played in that round. With this

$$\sum_{ab\in A_1A_2} P_{k,G_\epsilon}(a,b \mid \pi_1,\pi_2)u_i(a,b)$$

should be the expected utility from the kth round (where not getting to the kth round counts as 0). This suggests a new way of calculating expected utilities on a more round-by-round basis.

Lemma 2 *Let G_ϵ be a game, and let π_1, π_2 be strategies for that game. Then*

$$\mathbb{E}[u_i \mid \pi_1,\pi_2] = \sum_{k=0}^{\infty} \sum_{ab\in A_1A_2} P_{k,G_\epsilon}(a,b \mid \pi_1,\pi_2)u_i(a,b).$$

Proof

$$\begin{aligned}
\mathbb{E}_{G_\epsilon}[u_i \mid \pi_1,\pi_2] &= \sum_{h\in(A_1A_2)^+} P(h \mid \pi_1,\pi_2)u_i(h) \\
&\underset{\text{Eqs. 4, 6}}{=} \sum_{a_0b_0\cdots a_nb_n\in(A_1A_2)^+} \pi_1(0,a_0)\pi_2(0,b_0)\epsilon(1-\epsilon)^n \left(\prod_{j=1}^{n} \pi_1(b_{j-1},a_j)\pi_2(a_{j-1},b_j)\right) \\
&\quad \cdot \sum_{k=0}^{n} u_i(a_k,b_k) = \sum_{k=0}^{\infty} \sum_{a_0b_0\cdots a_nb_n\in(A_1A_2)^{\geq k+1}} \pi_1(0,a_0)\pi_2(0,b_0)\epsilon(1-\epsilon)^n \\
&\quad \cdot \left(\prod_{j=1}^{n} \pi_1(b_{j-1},a_j)\pi_2(a_{j-1},b_j)\right) u_i(a_k,b_k) \\
&= \sum_{k=0}^{\infty} \sum_{a_0b_0\cdots a_kb_k\in(A_1A_2)^{k+1}} \sum_{a_{k+1}b_{k+1}\cdots a_nb_n\in(A_1A_2)^*} \pi_1(0,a_0)\pi_2(0,b_0)\epsilon(1-\epsilon)^n u_i(a_k,b_k)
\end{aligned}$$

$$
\begin{aligned}
&\cdot \left(\prod_{j=1}^{k} \pi_1(b_{j-1}, a_j)\pi_2(a_{j-1}, b_j) \right) \left(\prod_{j=k+1}^{n} \pi_1(b_{j-1}, a_j)\pi_2(a_{j-1}, b_j) \right) \\
&= \sum_{k=0}^{\infty} \sum_{a_0 b_0 \cdots a_k b_k \in (A_1 A_2)^{k+1}} \pi_1(0, a_0)\pi_2(0, b_0)\epsilon(1-\epsilon)^k u_i(a_k, b_k) \\
&\quad \cdot \left(\prod_{j=1}^{k} \pi_1(b_{j-1}, a_j)\pi_2(a_{j-1}, b_j) \right) \\
&\quad \cdot \sum_{a_{k+1} b_{k+1} \cdots a_n b_n \in (A_1 A_2)^*} \epsilon(1-\epsilon)^{n-k} \left(\prod_{j=k+1}^{n} \pi_1(b_{j-1}, a_j)\pi_2(a_{j-1}, b_j) \right) \\
&\underset{\text{lemma } 1}{=} \sum_{k=0}^{\infty} \sum_{a_0 b_0 \cdots a_k b_k \in (A_1 A_2)^{k+1}} \pi_1(0, a_0)\pi_2(0, b_0)\epsilon(1-\epsilon)^k u_i(a_k, b_k) \\
&\quad \cdot \left(\prod_{j=1}^{k} \pi_1(b_{j-1}, a_j)\pi_2(a_{j-1}, b_j) \right) \\
&\underset{\text{Eq. } 7}{=} \sum_{k=0}^{\infty} \sum_{a_k b_k \in A_1 A_2} P_k(a_k, b_k \mid \pi_1, \pi_2) u_i(a_k, b_k)
\end{aligned}
$$

□

To find the probability of player i choosing a in round k, we usually have to calculate the probabilities of all actions in all previous rounds. After all, player i reacts to player $-i$'s previous move, who in turn reacts to player i's move in round $k-2$, and so on. This is what makes Eq. 7 so long. However, imagine that player $-i$ uses a stationary strategy. This, of course, means that player $-i$'s probability distribution over moves in round k (assuming the game indeed reaches round k) can be computed directly as $\pi_{-i}(b)$. Player i's distribution over moves in round k is almost as simple to calculate, because it only depends on player $-i$'s distribution over moves in round $k-1$, which can also be calculated directly. We hence get the following lemma.

Lemma 3 *Let G_ϵ be a game, let π_i be a any strategy for G_ϵ and let π_{-i} be a stationary strategy for G_ϵ. Then, for all $k \in \mathbb{N}_+$, it is*

$$
P_{k,G_\epsilon}(a, b \mid \pi_i, \pi_{-i}) = (1-\epsilon)^k \sum_{b' \in A_{-i}} \pi_{-i}(b')\pi_{-i}(b)\pi_i(b', a).
$$

Proof We conduct our proof by induction over k. For $k = 1$, it is

$$
P_1(a, b \mid \pi_i, \pi_{-i}) \underset{\text{Eq. } 7}{=} (1-\epsilon) \sum_{a_0 b_0} \pi_i(b_0, a)\pi_{-i}(a_0, b)\pi_i(0, a_0)\pi_{-i}(0, b_0)
$$

Table 1 Payoff matrix for the prisoner's dilemma

Player 1	Player 2	
	Cooperate	Defect
Cooperate	3, 3	1, 4
Defect	4, 1	2, 2

$$= (1-\epsilon)\sum_{b_0}\pi_i(b_0,a)\pi_{-i}(b)\pi_{-i}(b_0)\sum_{a_0}\pi_i(0,a_0)$$
$$= (1-\epsilon)\sum_{b_0}\pi_i(b_0,a)\pi_{-i}(b)\pi_{-i}(b_0).$$

If the lemma is true for k, it is also true for $k+1$:

$$P_{k+1}(a,b\mid\pi_i,\pi_{-i}) = (1-\epsilon)^{k+1}\sum_{a_0b_0\cdots a_kb_k\in(A_iA_{-i})^{k+1}}\pi_i(b_k,a)\pi_{-i}(a_k,b)\pi_i(0,a_0)\pi_{-i}(0,b_0)$$
$$\cdot\prod_{j=1}^{k}\pi_i(b_{j-1},a_j)\pi_{-i}(a_{j-1},b_j)$$
$$\underset{\text{Eq. 7}}{=}(1-\epsilon)\sum_{a_kb_k}P_k(a_kb_k\mid\pi_i,\pi_{-i})\pi_i(b_k,a)\pi_{-i}(b)$$
$$\underset{I.H.}{=}(1-\epsilon)^{k+1}\sum_{a_kb_k}\sum_{b'}\pi_{-i}(b')\pi_{-i}(b_k)\pi_{-i}(b)\pi_i(b',a_k)\pi_i(b_k,a)$$
$$= (1-\epsilon)^{k+1}\sum_{b_k}\pi_{-i}(b_k)\pi_{-i}(b)\pi_i(b_k,a)\sum_{b'}\pi_{-i}(b')\sum_{a_k}\pi_i(b',a_k)$$
$$= (1-\epsilon)^{k+1}\sum_{b_k}\pi_{-i}(b_k)\pi_{-i}(b)\pi_i(b_k,a).$$

□

3 Robust program equilibrium in the prisoner's dilemma

Discussions of the program equilibrium have traditionally used the well-known prisoner's dilemma (or trivial variations thereof) as an example to show how the program equilibrium rationalizes cooperation where the Nash equilibrium fails (e.g., Tennenholtz 2004, Sect. 3; McAfee 1984; Howard 1988; Barasz et al. 2014). The present paper is no exception. In this section, we will present our main idea using the example of the prisoner's dilemma; the next section gives the more general construction and proofs of properties of that construction. For reference, the payoff matrix of the prisoner's dilemma is given in Table 1.

I propose to use the following decision rule: with a probability of $\epsilon \in (0,1]$, cooperate. Otherwise, act as your opponent plays against you. I will call this strat-

```
Data: program profile (p_1, p_2)
Result: action a_i ∈ {C, D}
if sample(0, 1) < ϵ then
    return C
end
return sample(apply(p_{-i}, (p_1, p_2)))
```

Algorithm 1: The ϵGroundedFairBot for player i. The program makes use of a function *sample* which samples uniformly from the given interval or probability distribution. It is assumed that ϵ is computable.

egy ϵGroundedFairBot. A description of the algorithm in pseudo-code is given in Algorithm 1.[3]

The proposed program combines two main ideas. First, it is a version of FairBot (Barasz et al. 2014). That is, it chooses the action that its opponent would play against itself. As player $-i$ would like player i to cooperate, ϵGroundedFairBot thus incentivizes cooperation, as long as $\epsilon < 1/2$. In this, it resembles the tit for tat strategy in the iterated prisoner's dilemma (IPD) (famously discussed by Axelrod 2006), which takes an empirical approach to behaving as the opponent behaves against itself. Here, the probability of the game ending must be sufficiently small (again, less than $1/2$ for the given payoffs) in each round for the threat of punishment and the allure of reward to be persuasive reasons to cooperate.

The second main idea behind ϵGroundedFairBot is that it cooperates with some small probability ϵ. First and foremost, this avoids running into the infinite loop that a naive implementation of FairBot—see Algorithm 2— runs into when playing against opponents who, in turn, try to simulate FairBot. Note, again, the resemblance with the tit for tat strategy in the iterated prisoner's dilemma, which cooperates when no information about the opponent's strategy is available.

```
Data: program profile (p_1, p_2)
Result: action a_i ∈ {C, D}
return sample(apply(p_{-i}, (p_1, p_2)))
```

Algorithm 2: The NaiveFairBot for player i

To better understand how ϵGroundedFairBot works, consider its behavior against a few different opponents. When ϵGroundedFairBot faces NaiveFairBot, then both cooperate. For illustration, a dynamic call graph of their interaction is given in Fig. 1. It is left as an exercise for the reader to analyze ϵGroundedFairBot's behavior against other programs, such as another instance of ϵGroundedFairBot or a variation of ϵGroundedFairBot that defects rather than cooperates with probability ϵ.

When playing against strategies that are also based on simulating their opponent, we could think of ϵGroundedFairBot as playing a "mental IPD". If the opponent program decides whether to cooperate, it has to consider that it might currently only be simu-

[3] This program was proposed by Abram Demski in a conversation discussing similar (but worse) ideas of mine. It has also been proposed by Jessica Taylor at https://agentfoundations.org/item?id=524, though in a slightly different context.

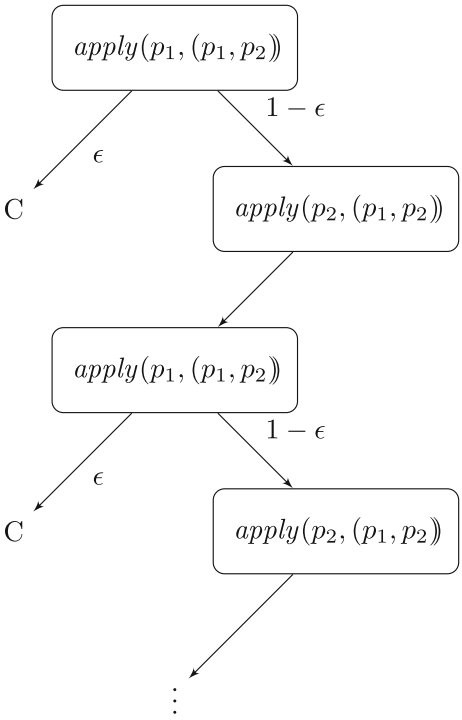

Fig. 1 Dynamic call diagram describing how $p_1 = \epsilon$GroundedFairBot chooses when playing against $p_2 =$ NaiveFairBot.

lated. Thus, it will choose an action with an eye toward gauging a favorable reaction from ϵGroundedFairBot one recursion level up. Cooperation in the first "round" is an attempt to steer the mental IPD into a favorable direction, at the cost of cooperating if *sample* $(0, 1) < \epsilon$ already occurs in the first round.

In addition to proving theoretical results (as done below), it would be useful to test ϵGroundedFairBot "in practice", i.e., against other proposed programs for the prisoner's dilemma with access to one another's source code. I only found one informal tournament for this version of the prisoner's dilemma. It was conducted in 2013 by Alex Mennen on the online forum and community blog LessWrong.[4] In the original set of submissions, ϵGroundedFairBot would have scored 6th out of 21. The reason why it is not a serious contender for first place is that it does not take advantage of the many exploitable submissions (such as bots that decide without looking at their opponent's source code). Once one removes the bottom 9 programs, ϵGroundedFairBot scores second place. If one continues this process of eliminating unsuccessful programs for another two rounds, ϵGroundedFairBot ends up among the four survivors that cooperate with each other.[5]

[4] The tournament was announced at https://www.lesserwrong.com/posts/BY8kvyuLzMZJkwTHL/prisoner-s-dilemma-with-visible-source-code-tournament and the results at https://www.lesserwrong.com/posts/QP7Ne4KXKytj4Krkx/prisoner-s-dilemma-tournament-results-0.

[5] For a more detailed analysis and report on my methodology, see https://casparoesterheld.files.wordpress.com/2018/02/transparentpdwriteup.pdf.

4 From iterated game strategies to robust program equilibria

We now generalize the construction from the previous section. Given any computable strategy π_i for a sequential game G_ϵ, I propose the following program: with a small probability ϵ sample from $\pi_i(0)$. Otherwise, act how π_i would respond (in the sequential game G_ϵ) to the action that the opponent takes against this program. I will call this program ϵGroundedπ_iBot. A description of the program in pseudo-code is given in Algorithm 3. As a special case, ϵGroundedFairBot arises from ϵGroundedπ_iBot by letting π_i be tit for tat.

Data: program profile (p_1, p_2)
Result: action $a_i \in A_i$
if *sample* $(0, 1) < \epsilon$ **then**
  return *sample*$(\pi_i(0))$
end
return *sample*$(\pi_i($*sample*$($*apply*$(p_{-i}, (p_1, p_2)))))$

Algorithm 3: The ϵGroundedπ_iBot for player i. The program makes use of a function *sample* which samples uniformly from a given interval or a given probability distribution. It is assumed that π_i and ϵ are computable.

Again, our proposed program combines two main ideas. First, ϵGroundedπ_iBot responds to how the opponent plays against ϵGroundedπ_iBot. In this, it resembles the behavior of π_i in G_ϵ. As we will see, this leads ϵGroundedπ_iBot to inherit many of π_i's properties. In particular, if (like tit for tat) π_i uses some mechanism to incentivize its opponent to converge on a desired action, then ϵGroundedπ_iBot incentivizes that action in a similar way.

Second, it—again—terminates immediately with some small probability ϵ to avoid the infinite loops that Naiveπ_iBot—see Algorithm 4—runs into. Playing $\pi_i(0)$ in particular is partly motivated by the "mental G_ϵ" that ϵGroundedπ_iBot plays against some opponents (such as ϵGroundedπ_{-i}Bots or Naiveπ_{-i}Bots). The other motivation is to make the relationship between ϵGroundedπ_iBot and π_i cleaner. In terms of the strategies that are optimal against ϵGroundedπ_iBot, the choice of that constant action cannot matter much if ϵ is small. Consider, again, the analogy with tit for tat. Even if tit for tat started with defection, one should still attempt to cooperate with it. In practice, however, it has turned out that the "nice" version of tit for tat (and nice strategies in general) are more successful (Axelrod 2006, ch. 2). The transparency in the program equilibrium may render such "signals of cooperativeness" less important—e.g., against programs like Barasz et al.'s (2014, Sect. 3) FairBot. Nevertheless, it seems plausible that—if only because of mental G_ϵ-related considerations—in transparent games the "initial" actions matter as well.

Data: program profile (p_1, p_2)
Result: action $a_i \in A_i$
return *sample*$(\pi_i($*sample*$($*apply*$(p_{-i}, (p_1, p_2)))))$

Algorithm 4: The Naiveπ_iBot for player i. It is assumed that π_i is computable.

We now ground these two intuitions formally. First, we discuss ϵGroundedπ_iBot's halting behavior. We then show that, in some sense, ϵGroundedπ_iBot behaves in G like π_i does in G_ϵ.

4.1 Halting behavior

For a program to be a viable option in the "transparent" version of G, it should halt against a wide variety of opponents. Otherwise, it may be excluded from $PROG(G)$ in our formalism. Besides, it should be efficient enough to be practically useful. As with ϵGroundedFairBot, the main reason why ϵGroundedπ_iBot is benign in terms of the risk of infinite loops is that it generates strictly less than one new function call in expectation and never starts more than one. While we have no formal machinery for analyzing the "loop risk" of a program, it is easy to show the following theorem.

Theorem 4 *Let π_i be a computable strategy for a game G_ϵ. Furthermore, let p_{-i} be any program (not necessarily in $PROG_i(G)$) that calls $apply(p_i, (p_i, p_{-i}))$ at most once and halts with probability 1 if apply halts with probability 1. Then ϵGroundedπ_iBot and p_{-i} halt against each other with probability 1 and the expected number of steps required for executing ϵGroundedπ_iBot is at most*

$$T_{\pi_i} + (T_{\pi_i} + T_{p_{-i}})\frac{1-\epsilon}{\epsilon}, \tag{8}$$

where T_{π_i} is the maximum number of steps to sample from π_i, and $T_{p_{-i}}$ is the maximum number of steps needed to sample from $apply(p_{-i}, (\epsilon\text{Grounded}\pi_i\text{Bot}, p_{-i}))$ excluding the steps needed to execute $apply(\epsilon\text{Grounded}\pi_i\text{Bot}, (\epsilon\text{Grounded}\pi_i\text{Bot}, p_{-i}))$.

Proof It suffices to discuss the cases in which p_{-i} calls p_i once with certainty, because if any of our claims were refuted by some program p_{-i}, they would also be refuted by a version of that program that calls p_i once with certainty. If p_{-i} calls p_i once with certainty, then the dynamic call graphs of both ϵGroundedπ_iBot and p_{-i} look similar to the one drawn in 1. In particular, it only contains one infinite path and that path has a probability of at most $\lim_{i\to\infty}(1-\epsilon)^i = 0$.

For the time complexity, we can consider the dynamic call graph as well. The policy π_i has to be executed at least once (with probability ϵ with the input 0 and with probability $1-\epsilon$ against an action sampled from $apply(p_{-i}, (\epsilon\text{Grounded}\pi_i\text{Bot}, p_{-i}))$. With a probability of $(1-\epsilon)$, we also have to execute the non-simulation part of p_{-i} and, for a second time, π_i. And so forth. The expected number of steps to execute ϵGroundedπ_iBot is thus

$$T_{\pi_i} + \sum_{j=1}^{\infty}(1-\epsilon)^j(T_{\pi_i} + T_{p_{-i}}),$$

which can be shown to be equal to the term in 8 by using the well-known formula $\sum_{k=0}^{\infty} x^k = 1/(1-x)$ for the geometric series. □

Note that this argument would not work if there were more than two players or if the strategy for the iterated game were to depend on more than just the last opponent move, because in these cases, the natural extension of ϵGroundedπ_iBot would have to make multiple calls to other programs. Indeed, this is one of the reasons why the present paper only discusses two-player games and iterated games with such short-term memory. Whether a similar result can nonetheless be obtained for more than 2 players and strategies that depend on the entire past history is left to future research.

As special cases, for any strategy π_{-i}, ϵGroundedπ_iBot terminates against ϵGroundedπ_{-i}Bot and Naiveπ_{-i}Bot (and these programs in turn terminate against ϵGroundedπ_iBot). The latter is especially remarkable. Our ϵGroundedπ_iBot terminates and leads the opponent to terminate even if the opponent is so careless that it would not even terminate against a version of itself or, in our formalism, if $PROG_{-i}(G)$ gives the opponent more leeway to work with simulations.

4.2 Relationship to the underlying iterated game strategy

Theorem 5 *Let G be a game, π_i be a strategy for player i in G_ϵ, $p_i = \epsilon$Groundedπ_iBot and $p_{-i} \in PROG_{-i}(G)$ be any opponent program. We define $\pi_{-i} = apply(p_{-i}, (p_i, p_{-i}))$, which makes π_{-i} a strategy for player $-i$ in G_ϵ. Then*

$$\mathbb{E}_G\left[u_i \mid p_i, p_{-i}\right] = \epsilon \mathbb{E}_{G_\epsilon}\left[u_i \mid \pi_i, \pi_{-i}\right].$$

Proof We separately transform the two expected values in the equation that is to be proven and then notice that they only differ by a factor ϵ:

$$\begin{aligned}
\mathbb{E}_{G_\epsilon}\left[u_i \mid \pi_i, \pi_{-i}\right] &\underset{\text{lemma 2}}{=} \sum_{k=0}^{\infty} \sum_{ab \in A_i A_{-i}} P_{k,G_\epsilon}(a, b \mid \pi_i, \pi_{-i}) u_i(a, b) \\
&\underset{\text{lemma 3}}{=} \sum_{ab \in A_i A_{-i}} \pi_i(0, a) \pi_{-i}(b) u_i(a, b) \\
&\quad + \sum_{k=1}^{\infty} \sum_{ab \in A_i A_{-i}} (1-\epsilon)^k \sum_{b' \in A_{-i}} \pi_{-i}(b') \pi_i(b', a) \pi_{-i}(b) u_i(a, b) \\
&\underset{\text{absolute convergence}}{=} \sum_{ab \in A_i A_{-i}} \pi_i(0, a) \pi_{-i}(b) u_i(a, b) \\
&\quad + \sum_{ab \in A_i A_{-i}} \sum_{b' \in A_{-i}} \pi_{-i}(b') \pi_i(b', a) \pi_{-i}(b) u_i(a, b) \sum_{k=1}^{\infty} (1-\epsilon)^k \\
&\underset{\text{sum of geometric series}}{=} \sum_{ab \in A_i A_{-i}} \pi_i(0, a) \pi_{-i}(b) u_i(a, b) \\
&\quad + \frac{1-\epsilon}{\epsilon} \sum_{ab \in A_i A_{-i}} \sum_{b' \in A_{-i}} \pi_{-i}(b') \pi_i(b', a) \pi_{-i}(b) u_i(a, b).
\end{aligned}$$

The second-to-last step uses absolute convergence to reorder the sum signs. The last step uses the well-known formula $\sum_{k=0}^{\infty} x^k = 1/(1-x)$ for the geometric series.

Onto the other expected value:

$$\begin{aligned}
&\mathbb{E}_G\left[u_i \mid p_i, p_{-i}\right] \\
&\underset{\text{Eqs. } 3,\ 2,\ 1}{=} \sum_{ab \in A_i A_{-i}} apply(p_i, (p_i, p_{-i}), a) apply(p_{-i}, (p_i, p_{-i}), b) u_i(a, b) \\
&\underset{\text{def.s } p_i,\ \pi_{-i}}{=} \sum_{ab \in A_i A_{-i}} \left(\epsilon \pi_i(0, a) + (1-\epsilon) \sum_{b' \in A_{-i}} \pi_{-i}(b') \pi_i(b', a) \right) \pi_{-i}(b) u_i(a, b) \\
&\quad = \epsilon \sum_{ab \in A_i A_{-i}} \pi_i(0, a) \pi_{-i}(b) u_i(a, b) \\
&\qquad + (1-\epsilon) \sum_{ab \in A_i A_{-i}} \sum_{b' \in A_{-i}} \pi_{-i}(b') \pi_i(b', a) \pi_{-i}(b) u_i(a, b).
\end{aligned}$$

Here, $apply(p_i, (p_i, p_{-i}), a) := apply(p_i, (p_i, p_{-i}))(a)$. The hypothesis follows immediately. □

Note that the program side of the proof does not involve any "mental G_ϵ". Using Theorem 5, we can easily prove a number of property transfers from π_i to $\epsilon\text{Grounded}\pi_i\text{Bot}$.

Corollary 6 *Let G be a game. Let π_i be a computable strategy for player i in G_ϵ and let $p_i = \epsilon\text{Grounded}\pi_i\text{Bot}$.*

1. *If $p_{-i} \in B_{-i}(p_i)$, then $apply(p_{-i}, (p_i, p_{-i})) \in B_{-i}^{s,c}(\pi_i)$.*
2. *If $\pi_{-i} \in B_i^{s,c}(\pi_i)$ and $apply(p_{-i}, (p_i, p_{-i})) = \pi_{-i}$, then $p_{-i} \in B_{-i}(p_i)$.*

Proof Both 1. and 2. follow directly from Theorem 5. □

Intuitively speaking, Corollary 6 shows that π_i and $\epsilon\text{Grounded}\pi_i\text{Bot}$ provoke the same best responses. Note that best responses in the program game only correspond to best stationary computable best responses in the repeated game. The computability requirement is due to the fact that programs cannot imitate incomputable best responses. The corresponding strategies for the repeated game further have to be stationary because $\epsilon\text{Grounded}\pi_i\text{Bot}$ only incentivizes opponent behavior for a single situation, namely the situation of playing against $\epsilon\text{Grounded}\pi_i\text{Bot}$. As a special case of Corollary 6, if $\epsilon < 1/2$, the best response to $\epsilon\text{GroundedFairBot}$ is a program that cooperates against $\epsilon\text{GroundedFairBot}$ because in a IPD with a probability of ending of less than $1/2$ a program that cooperates is the best (stationary computable) response to tit for tat.

Corollary 7 *Let G be a game. If (π_1, π_2) is a Nash equilibrium of G_ϵ and π_1 and π_2 are computable, then $(\epsilon\text{Grounded}\pi_1\text{Bot}, \epsilon\text{Grounded}\pi_2\text{Bot})$ is a program equilibrium of G.*

Proof Follows directly from Theorem 5. □

4.3 Exploitability

Besides forming an equilibrium against many opponents (including itself) and incentivizing cooperation, another important reason for tit for tat's success is that it is "not very exploitable" (Axelrod 2006). That is, when playing against tit for tat, it is impossible to receive a much higher reward than tit for tat itself. We now show that (in)exploitability transfers from strategies π_i to $\epsilon\text{Grounded}\pi_i\text{Bots}$.

We call a game $G = (A_1, A_2, u_1, u_2)$ *symmetric* if $A_1 = A_2$ and $u_1(a, b) = u_2(b, a)$ for all $a \in A_1$ and $b \in A_2$. If G is symmetric, we call a strategy π_i for G_ϵ N-exploitable in G_ϵ for an $N \in \mathbb{R}_{\geq 0}$ if there exists a π_{-i}, such that

$$\mathbb{E}\left[u_{-i} \mid \pi_{-i}, \pi_i\right] > \mathbb{E}\left[u_i \mid \pi_{-i}, \pi_i\right] + N.$$

We call π_i N-inexploitable if it is not N-exploitable.

Analogously, in a symmetric game G we call a program p_i N-exploitable for an $N \in \mathbb{R}_{\geq 0}$ if there exists a p_{-i}, such that

$$\mathbb{E}\left[u_{-i} \mid p_{-i}, p_i\right] > \mathbb{E}\left[u_i \mid p_{-i}, p_i\right] + N.$$

We call p_i N-inexploitable if it is not N-exploitable.

Corollary 8 *Let G be a game and π_i be an N-inexploitable strategy for G_ϵ. Then $\epsilon\text{Grounded}\pi_i\text{Bot}$ is ϵN-inexploitable.*

Proof Follows directly from Theorem 5. □

Notice that if – like tit for tat in the IPD – π_i is N-inexploitable in G_ϵ for all ϵ, then we can make $\epsilon\text{Grounded}\pi_i\text{Bot}$ arbitrarily close to 0-inexploitable by decreasing ϵ.

5 Conclusion

In this paper, we gave the following recipe for constructing a program equilibrium for a given two-player game:

1. Construct the game's corresponding repeated game. In particular, we consider repeated games in which each player can only react to the opponent's move in the previous round (rather than the entire history of previous moves by both players) and the game ends with some small probability ϵ after each round.
2. Construct a Nash equilibrium for that iterated game.
3. Convert each of the strategies into a computer program that works as follows (see Algorithm 3): with probability ϵ do what the strategy does in the first round. With probability $1 - \epsilon$, apply the opponent program to this program; then do what the underlying strategy would reply to the opponent program's output.

The result is a program equilibrium which we have argued is more robust than the equilibria described by Tennenholtz (2004). More generally, we have shown that translating an individual's strategy for the repeated game into a program for the stage game

in the way described in step 3 retains many of the properties of the strategy for the repeated game. Thus, it seems that "good" programs to submit may be derived from "good" strategies for the repeated game.

Acknowledgements I am indebted to Max Daniel and an anonymous referee for many helpful comments.

References

Anderlini, L. (1990). Some notes on Church's thesis and the theory of games. *Theory and Decision*, *29*(1), 19–52.

Axelrod, R. (2006). *The Evolution of Cooperation*. New York: Basic Books.

Barasz, M., et al. (2014). Robust cooperation in the prisoner's dilemma: Program equilibrium via provability logic. arxiv: 1401.5577.

Binmore, K. (1987). Modeling rational players: Part I. *Economics and Philosophy*, *3*(2), 179–214.

Binmore, K. (1988). Modeling rational players: Part II. *Economics and Philosophy*, *4*(1), 9–55.

Critch, A. (2016). Parametric bounded Löb's theorem and robust cooperation of bounded agents. arxiv:1602.04184.

Cutland, N. (1980). *Computability: An introduction to recursive function theory*. Cambridge: Cambridge University Press.

van der Hoek, W., Witteveen, C., & Wooldridge, M. (2013). Program equilibrium—A program reasoning approach. *International Journal of Game Theory*, *42*(3), 639–671.

Howard, J. V. (1988). Cooperation in the prisoner's dilemma. *Theory and Decision*, *24*(3), 203–213.

McAfee, R. P. (1984). Effective computability in economic decisions. http://www.mcafee.cc/Papers/PDF/EffectiveComputability.pdf. Accessed 10 Nov 2018.

Osborne, M. J. (2004). *An introduction to game theory*. Oxford: Oxford University Press.

Tennenholtz, M. (2004). Program equilibrium. *Games and Economic Behavior*, *49*(2), 363–373.