Robust Program Equilibrium

Caspar Oesterheld, Foundational Research Institute
caspar.oesterheld@foundational-research.org

March 28, 2018

Abstract

One approach to achieving cooperation in the one-shot prisoner's dilemma is Tennenholtz's (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

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 (section 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 (Hoek, Witteveen, and Wooldridge, 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 section 3.

We then go on to generalize the construction exemplified in the prisoner's dilemma (see section 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 $u_i \colon A_1 \times A_2 \to \mathbb{R}$. 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

$$\mathbb{E}\left[u_i \mid \pi_1, \pi_2\right] := \sum_{(a_1, a_2) \in A_1 \times A_2} P((a_1, a_2) \mid \pi_1, \pi_2) \cdot u_i(a_1, a_2). \tag{2}$$

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.

We call a game *symmetric* if $A_1 = A_2$ and $u_1(a, b) = u_2(b, a)$ for all $a \in A_1$ and $b \in A_2$.

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 i-th 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 $p_{\sigma(i)}$ it plays against, where $o : \{1, 2\} \to \{1, 2\} : 1 \mapsto 2, 2 \mapsto 1$. For example, if p_i runs $apply(p_{\sigma(i)}, (p_i, p_{\sigma(i)}))$, i.e. simulates the opponent, then that's fine as long as $p_{\sigma(i)}$ does not also run $apply(p_{\sigma(i)}, (p_i, p_{\sigma(i)}))$, 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

$$P((a_1, a_2) \mid (p_1, p_2)) := P((a_1, a_2) \mid (apply(p_1, (p_1, p_2)), apply(p_2, (p_1, p_2))) \tag{3}$$

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

$$\mathbb{E}\left[u_i \mid (p_1, p_2)\right] := \sum_{(a_1,a_2)\in A_1\times A_2} P((a_1, a_2) \mid p_1, p_2) \cdot u_i(a_1, a_2). \tag{4}$$

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

$$B_i(p_{\sigma(i)}) = \operatorname*{arg\,max}_{p_i \in PROG_i(G)} \mathbb{E}\left[u_i \mid p_i, p_{\sigma(i)}\right]. \tag{5}$$

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

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 $p_{\sigma(i)}$ such that

$$\mathbb{E}\left[u_{\sigma(i)} \mid p_{\sigma(i)}, p_i\right] > \mathbb{E}\left[u_i \mid p_{\sigma(i)}, p_i\right] + N. \tag{6}$$

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

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, except in the first round where they have no information. Thus, a strategy π_i for player i non-deterministically maps opponent moves or the information of the lack thereof onto a move

$$\pi_i \colon \{0\} \cup A_{\sigma(i)} \rightsquigarrow A_i. \tag{7}$$

[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).

We call a strategy π_i *stationary* if for all $a \in A_i$, $\pi_i(b, a)$ is constant w.r.t. $b \in \{0\} \cup A_{\sigma(i)}$, where $\pi(b, a) := \pi(b)(a)$. 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 $h = a_0 b_0 a_1 b_1 ... a_n b_n$ and then ends is

$$P(h \mid (\pi_1, \pi_2)) := \pi_1(0, a_0)\pi_2(0, b_0)\epsilon(1-\epsilon)^n \prod_{i=1}^{n} \pi_1(b_{i-1}, a_i)\pi_2(a_{i-1}, b_i). \tag{8}$$

The expected value for player i given that strategy profile is

$$\mathbb{E}\left[u_i \mid \pi_1, \pi_2\right] := \sum_{h \in (A_1 \cdot A_2)^+} P(h \mid \pi_1, \pi_2) \cdot u_i(h), \tag{9}$$

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

$$u_i(a_0 b_0 a_1 b_1 ... a_n b_n) := \sum_{i=0}^{n} u_i(a_i, b_i). \tag{10}$$

The lax unordered summation is, again, unproblematic because of the absolute convergence of the series, which is a direct consequence of the proof of lemma 1 below.

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

$$B_i(\pi_{\sigma(i)}) = \underset{\pi_i : \{0\} \cup A_{\sigma(i)} \rightsquigarrow A_i}{\arg\max} \mathbb{E}\left[u_i \mid \pi_i, \pi_{\sigma(i)}\right]. \tag{11}$$

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

In a symmetric game G we call a strategy π_i N-exploitable for an $N \in \mathbb{R}_{\geq 0}$ if there exists a $\pi_{\sigma(i)}$ such that

$$\mathbb{E}\left[u_{\sigma(i)} \mid \pi_{\sigma(i)}, \pi_i\right] > \mathbb{E}\left[u_i \mid \pi_{\sigma(i)}, \pi_i\right] + N. \tag{12}$$

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

We now prove a few lemmas that we will need later on.

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

$$\sum_{h \in (A_1 \cdot A_2)^+} P(h \mid \pi_1, \pi_2) = 1. \tag{13}$$

Proof.

$$\begin{aligned}
\sum_h P(h \mid \pi_1, \pi_2) &\underset{\text{eq. } 8}{=} \sum_{a_0 b_0 ... a_n b_n \in (A_1 \cdot A_2)^+} \pi_1(0, a_0)\pi_2(0, b_0)\epsilon(1-\epsilon)^n \prod_{i=1}^{n} \pi_1(b_{i-1}, a_i)\pi_2(a_{i-1}, b_i) && (14)\\
&= \epsilon \sum_{n=0}^{\infty} (1-\epsilon)^n \sum_{a_0 b_0 ... a_n b_n \in (A_1 A_2)^{n+1}} \pi_1(0, a_0)\pi_2(0, b_0) \prod_{i=1}^{n} \pi_1(b_{i-1}, a_i)\pi_2(a_{i-1}, b_i) && (15)\\
&= \epsilon \sum_{n=0}^{\infty} (1-\epsilon)^n \sum_{a_0 b_0 \in A_1 A_2} \pi_1(0, a_0)\pi_2(0, b_0) \sum_{a_1 b_1 \in A_1 A_2} \pi_1(b_0, a_1)\pi_2(a_0, b_1) \ldots \sum_{a_n b_n \in A_1 A_2} \pi_1(b_{n-1}, a_n)\pi_2(a_{n-1}, b_n) && (16)\\
&= \epsilon \sum_{n=0}^{\infty} (1-\epsilon)^n && (17)\\
&= 1 && (18)
\end{aligned}$$

For seeing why equation 17 is true, notice that the inner-most sum is 1. Thus, the next sum is 1 as well, and so on. Since the ordering of the left hand side of eq. 15 is lax and because only the right-hand side is known to converge absolutely, the re-orderings 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 (see, e.g., Rudin, 1976, Theorem 3.26). □

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

$$P_{k,G_\epsilon}(a,b \mid \pi_1,\pi_2) := (1-\epsilon)^k \sum_{a_0b_0\ldots a_{k-1}b_{k-1}\in(A_1A_2)^k} \pi_1(b_{k-1},a)\pi_2(a_{k-1},b)\pi_1(0,a_0)\pi_2(0,b_0)\prod_{j=1}^{k-1}\pi_1(b_{j-1},a_j)\pi_2(a_{j-1},b_j). \tag{19}$$

For $k = 0$ we define

$$P_{0,G_\epsilon}(a,b \mid \pi_1,\pi_2) := \pi_1(0,a)\cdot\pi_2(0,b). \tag{20}$$

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.

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

$$\mathbb{E}\left[u_i \mid \pi_1,\pi_2\right] = \sum_{k=0}^{\infty}\sum_{ab\in A_1A_2} P_{k,G_\epsilon}(a,b \mid \pi_1,\pi_2)u_i(a,b). \tag{21}$$

Proof.

$$\begin{aligned}
\mathbb{E}_{G_\epsilon}\left[u_i \mid \pi_1,\pi_2\right] &= \sum_{h\in(A_1A_2)^+} P(h \mid \pi_1,\pi_2)u_i(h) && (22)\\
&\underset{\text{eq.s 8,10}}{=} \sum_{a_0b_0\ldots a_nb_n\in(A_1A_2)^+} \pi_1(0,a_0)\pi_2(0,b_0)\epsilon(1-\epsilon)^n\left(\prod_{j=1}^{n}\pi_1(b_{j-1},a_j)\pi_2(a_{j-1},b_j)\right)\sum_{k=0}^{n}u_i(a_k,b_k) && (23)\\
&= \sum_{k=0}^{\infty}\sum_{a_0b_0\ldots a_nb_n\in(A_1A_2)^{\geq k+1}} \pi_1(0,a_0)\pi_2(0,b_0)\epsilon(1-\epsilon)^n\left(\prod_{j=1}^{n}\pi_1(b_{j-1},a_j)\pi_2(a_{j-1},b_j)\right)u_i(a_k,b_k) && (24)\\
&= \sum_{k=0}^{\infty}\sum_{a_0b_0\ldots a_kb_k\in(A_1A_2)^{k+1}}\ \sum_{a_{k+1}b_{k+1}\ldots a_nb_n\in(A_1A_2)^*} \pi_1(0,a_0)\pi_2(0,b_0)\epsilon(1-\epsilon)^n u_i(a_k,b_k) && (25)\\
&\qquad\cdot\left(\prod_{j=1}^{k}\pi_1(b_{j-1},a_j)\pi_2(a_{j-1},b_j)\right)\left(\prod_{j=k+1}^{n}\pi_1(b_{j-1},a_j)\pi_2(a_{j-1},b_j)\right) && (26)\\
&= \sum_{k=0}^{\infty}\sum_{a_0b_0\ldots a_kb_k\in(A_1A_2)^{k+1}} \pi_1(0,a_0)\pi_2(0,b_0)\epsilon(1-\epsilon)^k u_i(a_k,b_k)\left(\prod_{j=1}^{k}\pi_1(b_{j-1},a_j)\pi_2(a_{j-1},b_j)\right) && (27)\\
&\qquad\cdot\sum_{a_{k+1}b_{k+1}\ldots a_nb_n\in(A_1A_2)^*}\epsilon(1-\epsilon)^{n-k}\left(\prod_{j=k+1}^{n}\pi_1(b_{j-1},a_j)\pi_2(a_{j-1},b_j)\right) && (28)\\
&\underset{\text{lemma 1}}{=} \sum_{k=0}^{\infty}\sum_{a_0b_0\ldots a_kb_k\in(A_1A_2)^{k+1}} \pi_1(0,a_0)\pi_2(0,b_0)\epsilon(1-\epsilon)^k u_i(a_k,b_k)\left(\prod_{j=1}^{k}\pi_1(b_{j-1},a_j)\pi_2(a_{j-1},b_j)\right) && (29)\\
&\underset{\text{eq. 19}}{=} \sum_{k=0}^{\infty}\sum_{a_kb_k\in A_1A_2} P_k(a_k,b_k \mid \pi_1,\pi_2)u_i(a_k,b_k) && (30)
\end{aligned}$$

□

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

$$P_{k,G_\epsilon}(a,b \mid \pi_i,\pi_{\sigma(i)}) = (1-\epsilon)^k \sum_{b'\in A_{\sigma(i)}} \pi_{\sigma(i)}(b')\pi_{\sigma(i)}(b)\pi_i(b',a) \tag{31}$$

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

Table 1: A payoff matrix for the prisoner's dilemma

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

$$P_1(a, b \mid \pi_i, \pi_{\sigma(i)}) \underset{\text{eq. } 19}{=} (1-\epsilon) \sum_{a_0 b_0} \pi_i(b_0, a)\pi_{\sigma(i)}(a_0, b)\pi_i(0, a_0)\pi_{\sigma(i)}(0, b_0) \tag{32}$$

$$= (1-\epsilon) \sum_{b_0} \pi_i(b_0, a)\pi_{\sigma(i)}(b)\pi_{\sigma(i)}(b_0) \sum_{a_0} \pi_i(0, a_0) \tag{33}$$

$$= (1-\epsilon) \sum_{b_0} \pi_i(b_0, a)\pi_{\sigma(i)}(b)\pi_{\sigma(i)}(b_0). \tag{34}$$

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

$$P_{k+1}(a, b \mid \pi_i, \pi_{\sigma(i)}) = (1-\epsilon)^{k+1} \sum_{a_0 b_0 \ldots a_k b_k \in (A_i A_{\sigma(i)})^{k+1}} \pi_i(b_k, a)\pi_{\sigma(i)}(a_k, b)\pi_i(0, a_0)\pi_{\sigma(i)}(0, b_0) \prod_{j=1}^{k} \pi_i(b_{j-1}, a_j)\pi_{\sigma(i)}(a_{j-1}, b_j) \tag{35}$$

$$\underset{\text{eq. } 19}{=} (1-\epsilon) \sum_{a_k b_k} P_k(a_k b_k \mid \pi_i, \pi_{\sigma(i)})\pi_i(b_k, a)\pi_{\sigma(i)}(b) \tag{36}$$

$$\underset{I.H.}{=} (1-\epsilon)^{k+1} \sum_{a_k b_k} \sum_{b'} \pi_{\sigma(i)}(b')\pi_{\sigma(i)}(b_k)\pi_{\sigma(i)}(b)\pi_i(b', a_k)\pi_i(b_k, a) \tag{37}$$

$$= (1-\epsilon)^{k+1} \sum_{b_k} \pi_{\sigma(i)}(b_k)\pi_{\sigma(i)}(b)\pi_i(b_k, a) \sum_{b'} \pi_{\sigma(i)}(b') \sum_{a_k} \pi_i(b', a_k) \tag{38}$$

$$= (1-\epsilon)^{k+1} \sum_{b_k} \pi_{\sigma(i)}(b_k)\pi_{\sigma(i)}(b)\pi_i(b_k, a) \tag{39}$$

□

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, section 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 strategy 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 $\sigma(i)$ would like player i to cooperate, ϵGroundedFairBot thus incentivizes cooperation, as long as ϵ is sufficiently small. 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

[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.

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

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.

against itself. Here, the probability of the game ending must be sufficiently small 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 $a_i \in \{C, D\}$
return $sample(apply(p_{\sigma(i)}, (p_1, p_2)))$

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 figure 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 simulated. 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 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.

[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`.

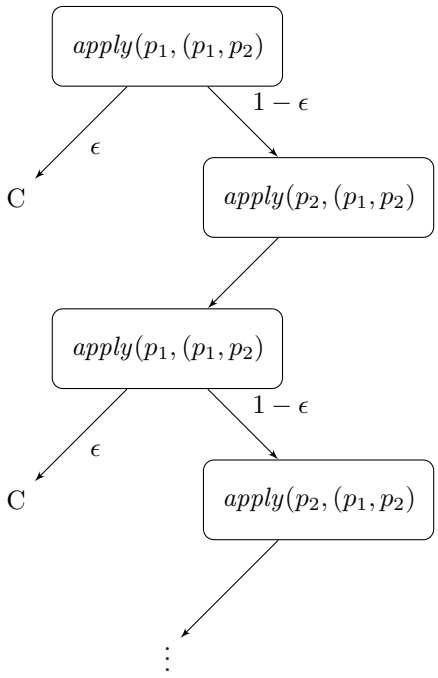

Figure 1: A dynamic call diagram describing how $p_1 = \epsilon$GroundedFairBot chooses when playing against $p_2 =$ NaiveFairBot.

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 $sample(\pi_i(sample(apply(p_{\sigma(i)}, (p_1, p_2)))))$

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 ϵGrounded$\pi_{\sigma(i)}$Bots or Naive$\pi_{\sigma(i)}$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, of course. 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 $a_i \in A_i$
1 **return** $sample(\pi_i(sample(apply(p_{\sigma(i)}, (p_1, p_2)))))$

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 $p_{\sigma(i)}$ be any program (not necessarily in $PROG_i(G)$) that calls apply at most once and halts with probability 1 if apply halts with probability 1. Then ϵGroundedπ_iBot and $p_{\sigma(i)}$ halt against each other with probability 1. In the worst case, executing ϵGroundedπ_iBot takes*

$$T_{\pi_i} + (T_{\pi_i} + T_{p_{\sigma(i)}})\frac{1-\epsilon}{\epsilon} \tag{40}$$

steps in expectation, where T_{π_i} is the number of steps to sample from π_i, and $T_{p_{\sigma(i)}}$ is the number of steps needed to sample from $apply(p_{\sigma(i)}, (\epsilon\text{Grounded}\pi_i\text{Bot}, p_{\sigma(i)}))$ minus the steps needed to execute $apply(p_{\sigma(i)}, \cdot)$.

Proof. In the worst case, the dynamic call graphs of both ϵGroundedπ_iBot and $p_{\sigma(i)}$ 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 $apply(p_{\sigma(i)}, (\epsilon\text{Grounded}\pi_i\text{Bot}, p_{\sigma(i)}))$. With a probability of $(1-\epsilon)$, we also have to execute the non-simulation part of $p_{\sigma(i)}$ and, for a second time, π_i. And so forth. The expected number of steps to execute ϵGroundedπ_iBot is thus

$$T_{\pi_i} + \sum_{j=1}^{\infty}(1-\epsilon)^j(T_{\pi_i} + T_{p_{\sigma(i)}}), \tag{41}$$

which can be shown to be equal to the term in 40 by using the well-known formula $\sum_{k=0}^{\infty} x^k = 1/(1-x)$ for the geometric series (see, e.g., Rudin, 1976, Theorem 3.26). □

Note that this argument would not work if there were more than two players, because in this case the natural extension of ϵGroundedπ_iBot may have to make multiple calls to other programs. Indeed, this is one of the reasons why the present paper only discusses 2-player games. Whether a similar result can nonetheless be obtained for more than 2 players is left to future research.

As special cases, for any strategy $\pi_{\sigma(i)}$, ϵGroundedπ_iBot terminates against ϵGrounded$\pi_{\sigma(i)}$Bot and Naive$\pi_{\sigma(i)}$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 $PROG_{\sigma(i)}(G)$ 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 G_ϵ, $p_i = \epsilon$Groundedπ_iBot and $p_{\sigma(i)} \in PROG_{\sigma(i)}(G)$ be any opponent program. We define $\pi_{\sigma(i)} = apply(p_{\sigma(i)}, (p_i, p_{\sigma(i)}))$. Then*

$$\mathbb{E}_G\left[u_i \mid p_i, p_{\sigma(i)}\right] = \epsilon \mathbb{E}_{G_\epsilon}\left[u_i \mid \pi_i, \pi_{\sigma(i)}\right]. \tag{42}$$

Proof. We separately transform the two expected values of equation 42 and then notice that they only differ by a factor ϵ.

$$\begin{aligned}
\mathbb{E}_{G_\epsilon}\left[u_i \mid \pi_i, \pi_{\sigma(i)}\right] &\underset{\text{lemma 2}}{=} \sum_{k=0}^{\infty} \sum_{ab \in A_i A_{\sigma(i)}} P_{k,G_\epsilon}(a, b \mid \pi_i, \pi_{\sigma(i)}) u_i(a, b) && (43)\\
&\underset{\text{lemma 3}}{=} \sum_{ab \in A_i A_{\sigma(i)}} \pi_i(0, a)\pi_{\sigma(i)}(b) u_i(a, b) + \sum_{k=1}^{\infty} \sum_{ab \in A_i A_{\sigma(i)}} (1-\epsilon)^k \sum_{b' \in A_{\sigma(i)}} \pi_{\sigma(i)}(b')\pi_i(b', a)\pi_{\sigma(i)}(b) u_i(a, b) && (44)\\
&= \sum_{ab \in A_i A_{\sigma(i)}} \pi_i(0, a)\pi_{\sigma(i)}(b) u_i(a, b) + \sum_{ab \in A_i A_{\sigma(i)}} \sum_{b' \in A_{\sigma(i)}} \pi_{\sigma(i)}(b')\pi_i(b', a)\pi_{\sigma(i)}(b) u_i(a, b) \sum_{k=1}^{\infty} (1-\epsilon)^k && (45)\\
&= \sum_{ab \in A_i A_{\sigma(i)}} \pi_i(0, a)\pi_{\sigma(i)}(b) u_i(a, b) + \frac{1-\epsilon}{\epsilon} \sum_{ab \in A_i A_{\sigma(i)}} \sum_{b' \in A_{\sigma(i)}} \pi_{\sigma(i)}(b')\pi_i(b', a)\pi_{\sigma(i)}(b) u_i(a, b) && (46)
\end{aligned}$$

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 (see, e.g., Rudin, 1976, Theorem 3.26).

On to the other expected value:

$$\begin{aligned}
\mathbb{E}_G\left[u_i \mid p_i, p_{\sigma(i)}\right] &\underset{\text{eq.s 4, 3, 1}}{=} \sum_{ab \in A_i A_{\sigma(i)}} apply(p_i, (p_i, p_{\sigma(i)}), a)\, apply(p_{\sigma(i)}, (p_i, p_{\sigma(i)}), b) u_i(a, b) && (47)\\
&\underset{\text{def.s } p_i,\ \pi_{\sigma(i)}}{=} \sum_{ab \in A_i A_{\sigma(i)}} \left(\epsilon \pi_i(0, a) + (1-\epsilon) \sum_{b' \in A_{\sigma(i)}} \pi_{\sigma(i)}(b')\pi_i(b', a)\right) \pi_{\sigma(i)}(b) u_i(a, b) && (48)\\
&= \epsilon \sum_{ab \in A_i A_{\sigma(i)}} \pi_i(0, a)\pi_{\sigma(i)}(b) u_i(a, b) + (1-\epsilon) \sum_{ab \in A_i A_\sigma(i)} \sum_{b' \in A_{\sigma(i)}} \pi_{\sigma(i)}(b')\pi_i(b', a)\pi_{\sigma(i)}(b) u_i(a, b) && (49)
\end{aligned}$$

Here, $apply(p_i, (p_i, p_{\sigma(i)}), a) := apply(p_i, (p_i, p_{\sigma(i)}))(a)$. 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 ϵGroundedπ_iBot.

Corollary 6. *Let G be a game. Let π_i be a computable strategy for player i in G_ϵ and let $p_i = \epsilon$Groundedπ_iBot.*

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

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

Intuitively speaking, corollary 6 shows that π_i and ϵGroundedπ_iBot provoke the same best responses. As a special case, if ϵ is sufficiently small, the best response to ϵGroundedFairBot is a program that cooperates.

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. □

Corollary 8. *Let G be a game and π_i be an N-inexploitable strategy for G_ϵ. Then ϵGroundedπ_iBot 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 ϵGroundedπ_iBot arbitrarily close to 0-inexploitable by decreasing ϵ.

Acknowledgements

I am indebted to Max Daniel for many helpful comments.

References

Anderlini, Luca (1990). "Some notes on Church's thesis and the theory of games". In: *Theory and Decision* 29.1, pp. 19–52.

Axelrod, Robert (2006). *The Evolution of Cooperation*. Basic Books.

Barasz, Mihaly et al. (2014). *Robust Cooperation in the Prisoner's Dilemma: Program Equilibrium via Provability Logic*. URL: `https://arxiv.org/abs/1401.5577`.

Binmore, Ken (1987). "Modeling Rational Players: Part I". In: *Economics & Philosophy* 3.2, pp. 179–214.

— (1988). "Modeling Rational Players: Part II". In: *Economics & Philosophy* 4.1, pp. 9–55.

Critch, Andrew (2016). *Parametric Bounded Löb's Theorem and Robust Cooperation of Bounded Agents*. URL: `https://arxiv.org/abs/1602.04184`.

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

Hoek, Wiebe van der, Cees Witteveen, and Micheal Wooldridge (2013). "Program equilibrium – a program reasoning approach". In: *International Journal of Game Theory* 42 (3), pp. 639–671.

Howard, J.V. (1988). "Cooperation in the Prisoner's Dilemma". In: *Theory and Decision* 24 (3).

McAfee, R. Preston (1984). *Effective Computability in Economic Decisions*. URL: `http://www.mcafee.cc/Papers/PDF/EffectiveComputability.pdf`.

Osborne, Martin J. (2004). *An Introduction to Game Theory*. Oxford University Press.

Rudin, Walter (1976). *Principles of Mathematical Analysis*. 3rd ed. McGraw-Hill.

Tennenholtz, Moshe (2004). "Program equilibrium". In: *Games and Economic Behavior* 49.2, pp. 363–373.