Program Equilibrium in the Prisoner's Dilemma via Löb's Theorem

Patrick LaVictoire
Quixey
278 Castro Street
Mountain View, CA 94041
patrick@quixey.com

Benja Fallenstein and Eliezer Yudkowsky
Machine Intelligence Research Institute
2030 Addison Street #300, Berkeley, CA 94703
benja@intelligence.org, eliezer@intelligence.org

Mihaly Barasz
Nilcons
Albisstrasse 22
Adliswil, CH-8134, Switzerland
klao@nilcons.com

Paul Christiano
University of California at Berkeley
Department of Computer Science
387 Soda Hall, Berkeley, CA 94720
paulfchristiano@eecs.berkeley.edu

Marcello Herreshoff
Google
1600 Ampitheatre Parkway
Mountain View, CA 94043
marcelloh@google.com

Abstract

Applications of game theory often neglect that real-world agents normally have some amount of out-of-band information about each other. We consider the limiting case of a one-shot Prisoner's Dilemma between algorithms with read-access to one anothers' source code. Previous work has shown that cooperation is possible at a Nash equilibrium in this setting, but existing constructions require interacting agents to be identical or near-identical. We show that a natural class of agents are able to achieve mutual cooperation at Nash equilibrium without any prior coordination of this sort.

1 Introduction

Can cooperation in a one-shot Prisoner's Dilemma be justified between rational agents? Rapoport (1999) argued in the 1960s that two agents with mutual knowledge of each others' rationality should be able to mutually cooperate. Howard (1988) explains the argument thus:

> Nonetheless arguments have been made in favour of playing C even in a single play of the PD. The one that interests us relies heavily on the usual assumption that both players are completely rational and know everything there is to know about the situation. (So for instance, Row knows that Column is rational, and Column knows that he knows it, and so on.) It can then be argued by Row that Column is an individual very similar to himself and in the same situation as himself. Hence whatever he eventually decides to do, Column will necessarily do the same (just as two good students given the same sum to calculate will necessarily arrive at the same answer). Hence if Row chooses D, so will Column, and each will get 1. However if Row chooses C, so will Column, and each will then get 2. Hence Row should choose C.

Hofstadter (1985) described this line of reasoning as "superrationality", and held that knowledge of similar cognitive aptitudes should be enough to establish it, though the latter contention is (to say the least) controversial within decision theory. However, one may consider a stronger assumption: what if each agent has some ability to predict in advance the actions of the other?

This stronger assumption suggests a convenient logical formalism. In the 1980s, Binmore (1987) considered game theory between programs which could read each other's source code before playing[1]:

> ...a player needs to be able to cope with hypotheses about the reasoning processes of the opponents other than simply that which maintains that they are the same as his own. Any other view risks relegating rational players to the role of the "unlucky" Bridge expert who usually loses but explains that his play is "correct" and would have led to his winning if only the opponents had played "correctly". Crudely, rational behavior should include the capacity to exploit bad play by the opponents.
>
> In any case, if Turing machines are used to model the players, it is possible to suppose that the play of a game is prefixed by an exchange of the players' Gödel numbers.

Having read-access to one another's source code would be unusual even among artificial agents, where this would in principle be possible. However, interacting agents do normally have some amount of information about each other, and the limiting case of read-access to the opponent's source seems like a reasonable starting point for investigating the importance of such information for game theory.

Howard (1988) and McAfee (1984) considered the Prisoner's Dilemma in this context, and each presented an example of an algorithm which would always return an answer, would cooperate if faced with itself, and would never cooperate when the opponent defected. (The solution discussed in both papers was a program that used quining of

[1] Binmore's analysis, however, eschews cooperation in the Prisoner's Dilemma as irrational!

the source code to implement the algorithm "cooperate if and only if the opponent's source code is identical to mine"; we represent it in this paper as Algorithm 3, which we call CliqueBot on account of the fact that it cooperates only with the 'clique' of agents identical to itself.)

More recently, Tennenholtz (2004) reproduced this result in the context of other research on multi-agent systems, noting that CliqueBot can be seen as a Nash equilibrium of the expanded game where two players decide which code to submit to the Prisoner's Dilemma with mutual source code read-access. This context (called "program equilibrium") led to several novel game-theoretic results, including folk theorems by Fortnow (2009) and Kalai, Kalai, Lehrer and Samet (2010), an answer by Monderer and Tennenholtz (2009) to the problem of seeking strong equilibria (many-agent Prisoner's Dilemmas in which mutual cooperation can be established in a manner that is safe from *coalitions* of defectors), a Bayesian framework by Peters and Szentes (2012), and more.

However, these approaches have an undesirable property: they restrict the circle of possible cooperators dramatically—in the most extreme case, only to agents that are syntactically identical! (Indeed, we will define examples of semantically distinct agents such that one would wish one's program to quickly cooperate with each of them.) Thus mutual cooperation for CliqueBots inherently requires prior coordination beyond the swap of source code, and an ecology of such agents would be akin to an all-out war between incompatible cliques.

This problem can be patched somewhat, but not cured, by prescribing a list of agents with whom mutual cooperation is desirable, but this approach is inelegant and still requires the creators of any pair of mutually cooperating agents to explicitly anticipate one another. We'd like to see agents that can decide on their own which other agents they should cooperate with.

A natural-seeming strategy involves simulating the other agent to see what they do when given one's own source code as input. Unfortunately, this leads to an infinite regress when two such agents are pitted against one another.

One attempt to put mutual cooperation on more stable footing is the model-checking result of van der Hoek, Witteveen, and Wooldridge (2011), which seeks "fixed points" of strategies that condition their actions on their opponents' output. However, in many interesting cases there are several fixed points, or none at all, and so this approach does not correspond to an algorithm as we would like.

Since the essence of this problem deals in counterfactuals—e.g. "what would they do if I did this"—it is worth considering modal logic, which was intended to capture reasoning about counterfactuals, and in particular the Gödel-Löb modal logic **GL** with provability as its modality. (See Boolos (1995) and Lindström (1996) for some good references on **GL**.) That is, if we consider agents that cooperate if and only if they can prove certain logical formulas, the structure of logical provability gives us a genuine framework for counterfactual reasoning, and in particular a powerful and surprising tool known as Löb's Theorem (Löb 1955):

Theorem 1.1 (Löb's Theorem)**.** *Let S be a formal system which includes Peano Arithmetic. If ϕ is any well-formed formula in S, let $\Box\phi$ be the formula in a Gödel encoding of S which claims that there exists a proof of ϕ in S; then whenever $S \vdash (\Box\phi \to \phi)$, in fact $S \vdash \phi$.*

We shall see that Löb's Theorem enables a flexible and secure form of mutual cooperation in this context. In particular, we first consider the intuitively appealing strategy "cooperate if and only if I can prove that my opponent cooperates", which we call FairBot. If we trust the formal system used by FairBot, we can conclude that it is unexploitable (in the sense that it never winds up with the sucker's payoff). When we play FairBot against itself (and give both agents sufficient power to find proofs), although either mutual cooperation or mutual defection seem philosophically consistent, it always finds mutual cooperation (Theorem 3.1)![2] Furthermore, we can construct another agent after the same fashion which improves on the main deficit of the above strategy: namely, that FairBot fails to correctly defect against CooperateBot, which cooperates with all opponents. We call this agent PrudentBot.

Moreover, the underpinnings of this result (and the others in this paper) do not depend on the syntactical details of the programs, but only on their semantic interpretations in provability logic; therefore two such programs can cooperate, even if written differently (in several senses, for instance if they use different Gödel encodings or different formal systems). Using the properties of Kripke semantics, one can algorithmically derive the fixed-point solutions to the action of one agent specified in the language of provability logic against another. Indeed, the authors have written a Haskell program which efficiently calculates the actions of two such agents defined via provability logic; the program is hosted at github.com/machine-intelligence/provability.

The results on Löbian cooperation reported here represent a formalized version of robust mutual cooperation on the Prisoner's Dilemma, further validating some of the intuitions on "superrationality" and raising new questions on decision theory. The Prisoner's Dilemma with exchange of source code is analogous to Newcomb's problem, and indeed, this work was inspired by some of the philosophical alternatives to causal and evidential decision theory proposed for that problem (see Drescher (2006) and Altair (2013)).

A brief outline of the structure of this paper: in Section 2, we define our formal framework more explicitly. In Section 3, we introduce FairBot, prove that it achieves mutual cooperation with itself and cannot be exploited (Theorem 3.1); we then introduce PrudentBot, and show that it is also unexploitable, cooperates mutually with itself and with FairBot, and defects against CooperateBot. In Section 4, we will explain our preference for PrudentBot over FairBot, and speculate on some future directions. Section 5 concludes.

[2] This result was proved by Vladimir Slepnev in an unpublished draft (2012), and the proof is reproduced later in this paper with his permission.

2 Agents in Formal Logic

There are two different formalisms which we will bear in mind throughout this paper. The first formalism is that of algorithms, where we can imagine two Turing machines `X` and `Y`, each of which is given as input the code for the other, and which have clearly defined outputs corresponding to the options C and D. (It is possible, of course, that one or both may fail to halt, though the algorithms that we will discuss will provably halt on all inputs.) This formalism has the benefit of concreteness: we could actually program such agents, although the ones we shall deal with are often very far from efficient in their requirements. On the other hand, deducing what happens when algorithms call upon each others' code is a difficult and messy affair in general.

For simplicity, then, we will therefore consider a class of (computationally intractable) agents which can consult an oracle to determine whether certain statements are provable in a particular formal system. This simplification is justified by the fact that all of the methods of this paper have analogous bounded versions; for example, variants of Löb's Theorem for bounded proof lengths are well-known among logicians. The interested reader can therefore construct algorithmic versions of all of the unbounded agents in this paper, and with the right parameters all of our theorems will hold for such agents. Our use of unbounded computational power is important primarily as a conceptual simplification.

For convenience, we will sometimes represent these unbounded agents as formulas in Peano Arithmetic with one free variable (any program can be represented as a formula of arithmetic, so this is not a real limitation in the class of agents we consider), and move freely between descriptions of agents in terms of programs and descriptions of agents in terms of formulas. Formally, fix a particular Gödel numbering scheme, and let `X` and `Y` each denote well-formed formulas with one free variable. Then let $X(Y)$ denote the formula where we replace each instance of the free variable in `X` with the Gödel number of `Y`. If such a formula holds in the standard model of Peano Arithmetic, we interpret that as `X` cooperating with `Y`; if its negation holds, we interpret that as `X` defecting against `Y`. Thus we can regard formulas of arithmetic with a single free variable as decision-theoretic agents, and we will use "source code" to refer to their Gödel numbers.

We will be particularly interested in agents which are defined in terms of provability in a particular formal system.[3] For a given theory T, write $\mathsf{T} \vdash \varphi$ for the assertion "φ is provable in T." Let PA be the usual theory of Peano Arithmetic and write $\Box\varphi$ for the formula in arithmetic which Gödel-encodes the statement "$\mathsf{PA} \vdash \varphi$". Define $\mathsf{PA+0} = \mathsf{PA}$, and define $\mathsf{PA+(n+1)}$ to be the extension of $\mathsf{PA+n}$ by the axiom $\neg\Box\cdots\Box\bot$ with $n+1$ boxes (that is, the assertion that $\mathsf{PA+n}$ is consistent).

Remark To maximize readability in the technical sections of this paper, we will use typewriter font for agents, which are formulas of Peano Arithmetic with a single free variable, like `X` and `CooperateBot`; we will use sans-serif font for the formal systems $\mathsf{PA+n}$; and we will use italics for logical formulas with no free variables such as C, D, and $X(Y)$.

Two agents which are easy to define and clearly efficiently implementable are the agent which always cooperates (which we will call `CooperateBot`, or `CB` for short) and the agent which always defects (which we will call `DefectBot`, or `DB`). In pseudocode:

```
Input  : Source code of the agent X
Output: $C$ or $D$
return $C$;
```

Algorithm 1: `CooperateBot (CB)`

```
Input  : Source code of the agent X
Output: $C$ or $D$
return $D$;
```

Algorithm 2: `DefectBot (DB)`

Remark In the Peano Arithmetic formalism, `CooperateBot` can be represented by a formula that is a tautology for every input, while `DefectBot` can be represented by the negation of such a formula. For any `X`, then, $\mathsf{PA} \vdash [CB(X) = C]$ and $\mathsf{PA} \vdash [DB(X) = D]$.

Note further that $\mathsf{PA} \nvdash \neg\Box[DB(X) = C]$, but that $\mathsf{PA+1} \vdash \neg\Box[DB(X) = C]$; this distinction is essential.

Howard (1988), McAfee (1984) and Tennenholtz (2004) introduced functionally equivalent agent schemas, which we've taken to calling `CliqueBot`; these agents use quining to recognize self-copies and mutually cooperate, while defecting against any other agent. In pseudocode:

```
Input  : Source code of the agent X
Output: $C$ or $D$
if X=CliqueBot then
    return $C$;
else
    return $D$;
end
```

Algorithm 3: `CliqueBot`

By the diagonal lemma, there exists a formula of Peano Arithmetic which implements `CliqueBot`. (The analogous tool for computable functions is Kleene's recursion theorem (Kleene 1938); in this paper, we informally use "quining" to refer to both of these techniques.)

`CliqueBot` has the nice property that it never experiences the sucker's payoff in the Prisoner's Dilemma. This is such a clearly important property that we will give it a name:

Definition We say that an agent `X` is *unexploitable* if there is no agent `Y` such that $X(Y) = C$ and $Y(X) = D$.

However, `CliqueBot` has a notable drawback: it can only elicit mutual cooperation from agents that are syntactically identical to itself. (If two `CliqueBots` were written

[3] Moreover, for any pair of the agents we will define in this paper, there exists an n such that their behavior against one another is decidable in $\mathsf{PA+n}$.

with different Gödel-numbering schemes, for instance, they would defect against one another!)

One might patch this by including a list of acceptable programs (or a schema for them), and cooperate if the opponent matches any of them; one would of course be careful to include only programs that would cooperate back with this variant. But this is a brittle form of mutual cooperation, and an opaque one: it takes a predefined circle of mutual cooperators as given. For this reason, it is worth looking for a more flexibly cooperative form of agent, one that can deduce for itself whether another agent is worth cooperating with.

3 Löbian Cooperation

A deceptively simple-seeming such agent is one we call `FairBot`. On a philosophical level, it cooperates with any agent that can be proven to cooperate with it. In pseudocode:

```
Input   : Source code of the agent X
Output: $C$ or $D$
if PA $\vdash$ [X(FairBot) $= C$] then
    return $C$;
else
    return $D$;
end
```

Algorithm 4: `FairBot (FB)`

`FairBot` references itself in its definition, but as with `CliqueBot`, this can be done via the diagonal lemma. By inspection, we see that `FairBot` is unexploitable: presuming that Peano Arithmetic is sound, `FairBot` will not cooperate with any agent that defects against `FairBot`.

The interesting question is what happens when `FairBot` plays against itself: it intuitively seems plausible either that it would mutually cooperate or mutually defect. As it turns out, though, Löb's Theorem guarantees that since the `FairBots` are each seeking proof of mutual cooperation, they both succeed and indeed cooperate with one another! (This was first shown by Vladimir Slepnev (2012).)

Theorem 3.1. $\mathsf{PA} \vdash [\textit{FairBot(FairBot)} = C]$.

Proof (Simple Version): By inspection of `FairBot`, we see that $\mathsf{PA} \vdash (\Box[FB(FB) = C]) \to [FB(FB) = C]$. Thus, by Löb's Theorem, Peano Arithmetic does indeed prove that *FairBot(FairBot)=C*. □

However, it is a tidy logical accident that the two agents are the same; we will understand better the mechanics of mutual cooperation if we pretend in this case that we have two distinct implementations, $\texttt{FairBot}_1$ and $\texttt{FairBot}_2$, and prove mutual cooperation from their formulas without using the fact that their actions are identical.

Proof of Theorem 3.1 (Real Version): Let A be the formula "$FB_1(FB_2) = C$" and B be the formula "$FB_2(FB_1) = C$". By inspection, $\mathsf{PA} \vdash \Box A \to B$ and $\mathsf{PA} \vdash \Box B \to A$. This sort of "Löbian circle" works out as follows:

$$\begin{array}{rl}
\mathsf{PA} \vdash (\Box A \to B) \wedge (\Box B \to A) & \text{(see above)} \\
\mathsf{PA} \vdash (\Box A \wedge \Box B) \to (A \wedge B) & \text{(follows from above)} \\
\mathsf{PA} \vdash \Box(A \wedge B) \to (\Box A \wedge \Box B) & \text{(tautology)} \\
\mathsf{PA} \vdash \Box(A \wedge B) \to (A \wedge B) & \text{(previous lines)} \\
\mathsf{PA} \vdash A \wedge B & \text{(Löb's Theorem).}
\end{array}$$

□

Remark One way to build a finitary version of `FairBot` is to write an agent `FiniteFairBot` that looks through all proofs of length $\leq N$ to see if any are a proof of $[X(\textit{FiniteFairBot}) = C]$, and cooperates iff it finds such a proof. If N is large enough, the bounded version of Löb's Theorem implies the equivalent of Theorem 3.1.

Remark Unlike a `CliqueBot`, `FairBot` will find mutual cooperation even with versions of itself that are written in other programming languages. In fact, even the choice of formal system does not have to be identical for two versions of `FairBot` to achieve mutual cooperation! It is enough that there exist a formal system S in which Löbian statements are true, such that anything provable in S is provable in each of the formal systems used, and such that S can prove the above. (Note in particular that even *incompatible* formal systems can have this property: a version of `FairBot` which looks for proofs in the formal system $\mathsf{PA}+\neg\mathsf{Con(PA)}$ will still find mutual cooperation with a `FairBot` that looks for proofs in $\mathsf{PA+1}$.)

However, `FairBot` wastes utility by cooperating even with `CooperateBot`[4]. Thus we would like to find a similarly robust agent which cooperates mutually with itself and with `FairBot` but which defects against `CooperateBot`.

Consider the agent `PrudentBot`, defined as follows:

```
Input   : Source code of the agent X
Output: $C$ or $D$
if PA $\vdash$ [X(PrudentBot)=C] and PA+1 $\vdash$
[X(DefectBot)=D] then
    return $C$;
end
return $D$;
```

Algorithm 5: `PrudentBot (PB)`

Theorem 3.2. *`PrudentBot` is unexploitable, mutually cooperates with itself and with `FairBot`, and defects against `CooperateBot`.*

Proof. Unexploitability is again immediate from the definition of `PrudentBot` and the assumption that PA is sound, since cooperation by `PrudentBot` requires a proof that its opponent cooperates against it.

In particular, $\mathsf{PA+1} \vdash [PB(DB) = D]$ (since $\mathsf{PA} \vdash [DB(PB) = D]$, $\mathsf{PA+1} \vdash \neg\Box[DB(PB) = C]$).

It is likewise clear that $\mathsf{PA+2} \vdash [PB(CB) = D]$.

[4] One might argue this is no great fault, but see Section 4.

Now since $\mathsf{PA}{+}1 \vdash [FB(DB) = D]$ and therefore $\mathsf{PA} \vdash \Box(\neg\Box\bot \to [FB(DB) = D])$, we again have the Löbian cycle where $\mathsf{PA} \vdash [PB(FB) = C] \leftrightarrow \Box[FB(PB) = C]$, and of course vice versa; thus `PrudentBot` and `FairBot` mutually cooperate.

And as we have established $\mathsf{PA}{+}1 \vdash [PB(DB) = D]$, we have the same Löbian cycle for `PrudentBot` and itself. □

Remark It is important that we look for proofs of $X(DB) = D$ in a stronger formal system than we use for proving $X(PB) = C$; if we do otherwise, the resulting variant of `PrudentBot` would lose the ability to cooperate with itself. However, it is not necessary that the formal system used for $X(DB) = D$ be stronger by only one step than that used for $X(PB) = C$; if we use a much higher $\mathsf{PA}{+}\mathsf{n}$ there, we broaden the circle of potential cooperators without thereby sacrificing safety.

4 Defecting Against CooperateBot

One might ask (on a philosophical level) why we object to FairBot in the first place; isn't it a feature, not a bug, that this agent offers up cooperation even to agents that blindly trust it? We suggest that it is too tempting to anthropomorphize agents in this context, and that many problems which can be interpreted as playing a Prisoner's Dilemma against a CooperateBot are situations in which one would not hesitate to "defect" in real life without qualms.

For instance, consider the following situation: You've come down with the common cold, and must decide whether to go out in public. If it were up to you, you'd stay at home and not infect anyone else. But it occurs to you that the cold virus has a "choice" as well: it could mutate and make you so sick that you'd have to go to the hospital, where it would have a small chance of causing a full outbreak! Fortunately, you know that cold viruses are highly unlikely to do this. If you map out the payoffs, however, you find that you are in a Prisoner's Dilemma with the cold virus, and that it plays the part of a CooperateBot. Are you therefore inclined to "cooperate" and infect your friends in order to repay the cold virus for not making you sicker?

The example is artificial and obviously facetious, but not entirely spurious. The world does not come with conveniently labeled "agents"; entities on scales at least from viruses to governments show signs of goal-directed behavior. Given a sufficiently broad counterfactual, almost any of these could be defined as a CooperateBot on a suitable Prisoner's Dilemma. And most of us feel no compunction about optimizing our human lives without worrying about the flourishing of cold viruses.[5]

5 Conclusions

Howard (1988) and McAfee (1984) independently proposed the same solution — CliqueBot — for allowing two algorithms with access to each other's source code to cooperate in a one-shot prisoner's dilemma when the opponent is of the same form, while being unexploitable (never experiencing the sucker's payoff against any opponent). But Howard's version was written in BASIC, whereas McAfee pinpointed a formula in the language of recursion theory. Since CliqueBot requires the opponents to use exactly the same source code, their programs would have defected against each other (once fed to a suitable interpreter).

In this paper, we have shown how a similar result can be obtained without this requirement for prior coordination. We first considered FairBot, which cooperates with its opponent if it can find a proof that its opponent will cooperate back. By Löb's theorem, FairBot will cooperate with itself, and a FairBot originally written in BASIC will cooperate with a FairBot originally specified using Kleene's recursion theorem.

But FairBot has a defect that CliqueBot avoids: it cooperates with CooperateBot, even though this is not necessary for enticing CooperateBot to cooperate back. In order to address this problem, we introduced PrudentBot, a slightly more complicated agent that is still unexploitable and achieves mutual cooperation with FairBot and itself, but which defects against CooperateBot. We have argued that this behavior is more reasonable for a rational agent than FairBot's.

Other agents akin to `FairBot` and `PrudentBot` can be constructed using the same framework of provability logic, and their actions against one another can also be calculated via Kripke semantics. Again, the authors have written a Haskell program at github.com/machine-intelligence/provability which allows for agents like these to be defined and which efficiently derives the outcomes of their interactions.

More realistic agents may represent their beliefs as probability assignments rather than simple assertions, and may employ a combination of deductive and inductive inference rather than a simple proof search. In this setting, we could also consider agents who have probabilistic beliefs about each other's algorithms rather than deterministic read-access to one anothers' source code. To the extent that deductive reasoning continues to play an important role for sophisticated probabilistic reasoners, our results may extend to more realistic agents.

Do these results imply that sufficiently intelligent and rational agents will reach mutual cooperation in one-shot Prisoner's Dilemmas? In a word, no, not yet. Many things about this setup are notably artificial, most prominently the perfectly reliable exchange of source code (and after that, the intractably long computations that might perhaps be necessary for even the finitary versions). Nor does this have direct implications among human beings; our abilities to read each other psychologically, while occasionally quite impressive, bear only the slightest analogy to the extremely artificial setup of access to the opponent's source code. Governments and corporations may be closer analogues to our agents (and

[5] Note that it would, in fact, be different if a virus were intelligent enough to predict the macroscopic behavior of their host and base their mutations on that! In such a case, one might well negotiate with the virus. Alternatively, if one's concern for the well-being of viruses reached a comparable level to one's concern for healthy friends, that would change the payoff matrix so that it was no longer a Prisoner's Dilemma. But both of these considerations are far more applicable to human beings than to viruses.

indeed, game theory has been applied much more successfully on that scale than on the human scale), but the authors would not consider the application of these results to such organizations to be straightforward, either. The theorems herein are not a demonstration that a more advanced approach to decision theory (i.e. one which does not fail on what we consider to be common-sense problems) is practical, only a demonstration that it is possible.

Acknowledgments

This project was developed at a research workshop held by the Machine Intelligence Research Institute (MIRI) in Berkeley, California, in April 2013; the authors gratefully acknowledge MIRI's hospitality and support throughout the workshop.

Patrick LaVictoire was partially supported by NSF Grant DMS-1201314 while working on this project.

Thanks to everyone who has commented on various partial results and drafts, in particular Alex Altair, Stuart Armstrong, Andrew Critch, Wei Dai, Daniel Dewey, Gary Drescher, Kenny Easwaran, Cameron Freer, Bill Hibbard, Vladimir Nesov, Vladimir Slepnev, Jacob Steinhardt, Nisan Stiennon, Jessica Taylor, and Qiaochu Yuan, and readers of the blog LessWrong for their comments on a preprint of this article.

References

Altair, A. 2013. A comparison of decision algorithms on Newcomblike problems.

Binmore, K. 1987. Modeling rational players: Part I. *Economics and Philosophy* 3(02):179–214.

Boolos, G. 1995. *The Logic of Provability*. Cambridge University Press.

Drescher, G. 2006. *Good And Real: Demystifying Paradoxes from Physics to Ethics*. A Bradford Book. MIT Press.

Fortnow, L. 2009. Program equilibria and discounted computation time. *Proc. 12th Conference on Theoretical Aspects of Rationality and Knowledge* 128–133.

Hoek, W.; Witteveen, C.; and Wooldridge, M. 2011. Program equilibrium—a program reasoning approach. *International Journal of Game Theory* 1–33.

Hofstadter, D. R. 1985. *Metamagical Themas: Questing for the Essence of Mind and Pattern*. BasicBooks.

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

Kalai, A. T.; Kalai, E.; Lehrer, E.; and Samet, D. 2010. A commitment folk theorem. *Games and Economic Behavior* 69(1):127 – 137. Special Issue In Honor of Robert Aumann.

Kleene, S. C. 1938. On notation for ordinal numbers. *J. Symb. Log.* 3(4):150–155.

Lindström, P. 1996. Provability logic—a short introduction. *Theoria* 62(1-2):19–61.

Löb, M. H. 1955. Solution of a Problem of Leon Henkin. *The Journal of Symbolic Logic* 20(2):pp. 115–118.

McAfee, R. P. 1984. Effective computability in economic decisions.

Monderer, D., and Tennenholtz, M. 2009. Strong mediated equilibrium. *Artif. Intell.* 173(1):180–195.

Peters, M., and Szentes, B. 2012. Definable and contractible contracts. *Econometrica* 80(1):363–411.

Rapoport, A. 1999. *Two-Person Game Theory*. Dover Books on Mathematics Series. Dover.

Slepnev, V. 2012. Self-referential algorithms for cooperation in one-shot games (unpublished draft).

Tennenholtz, M. 2004. Program equilibrium. *Games Econom. Behav.* 49(2):363–373.