Program Equilibrium — A Program Reasoning Approach

Wiebe van der Hoek* Cees Witteveen+ Michael Wooldridge*

*Department of Computer Science, University of Liverpool
Liverpool L69 3BX, UK

+Department of Software Technology, TU Delft
Mekelweg 4, 2628 CD, Delft, The Netherlands

Abstract

The concept of *program equilibrium*, introduced by Tennenholtz in 2004, represents one of the most ingenious and potentially far-reaching applications of ideas from computer science in game theory to date. The basic idea is that a player in a game selects a strategy by entering a program, whose behaviour may be conditioned on the programs submitted by other players. Thus, for example, in the prisoner's dilemma, a player can enter a program that says "If his program is the same as mine, then I cooperate, otherwise I defect". It can easily be shown that if such programs are permitted, then rational cooperation is possible in the prisoner's dilemma. In the original proposal of Tennenholtz, comparison between programs was limited to syntactic comparison of program texts. While this has the considerable advantage of simplicity, it does have some important limitations. In this paper, we investigate an approach to program equilibrium in which richer conditions are allowed, based on model checking – one of the most successful approaches to reasoning about programs. We introduce a decision-tree model of strategies, which may be conditioned on strategies of others. We then formulate and investigate a notion of "outcome" for our setting, and investigate the complexity of reasoning about outcomes. We illustrate our approach with many examples.

1 Introduction

The prisoner's dilemma is one of the most important and troubling scenarios considered by game theory (see, e.g., [1] for a classic study of the prisoner's dilemma). The prisoner's dilemma is important because the basic structure of the game seems to be very common in the everyday world: the dilemma has something to say about scenarios ranging from how superpower nations will behave in nuclear arms reduction treaties, to whether or not a bus passenger will pay their fare. The dilemma is troubling because it seems frustratingly paradoxical: the dominant strategy equilibrium outcome (mututal defection) makes *every* player *strictly* worse off than an alternative outcome (mutual cooperation). Given its importance and seemingly paradoxical nature, it is hardly surprising that many researchers have attempted to "recover" cooperation from the prisoner's dilemma – to try to find some way of explaining how and why mutual cooperation can and should rationally occur (see, e.g., [3, 4] for extensive references and critical discussion on this topic). The best-known variation in which mutual cooperation is rational is when the players know that they will play the game with each other again with a certain probability: if the probability is sufficiently high, then mutual cooperation becomes a rational outcome [2, p.357].

One of the reasons we find the prisoner's dilemma so very frustrating is that cooperation seems so close. The problem is that each prisoner must commit to either cooperate or defect, whereas intuitively, what each prisoner wants to say is "I'll cooperate if he will". In other words, each prisoner wants to make his commitment contingent on the commitments of others. The difficulty, however, is making this precise. In 2004, Moshe Tennenholtz introduced a fundamentally new way of looking at the problem, which uses ideas from computer science to formalise the notion of a strategy that is contingent on the strategies of others [11]. The basic idea in his approach is that a player in a game selects a strategy by entering what we call a *program strategy*: a program whose behaviour may be conditioned on the programs submitted by other players. Thus, for example, in the prisoner's dilemma, a player can enter a program strategy that intuitively says "If his program is the same as mine, then I cooperate, otherwise I defect". It can easily be shown that if such program strategies are permitted, then rational cooperation is possible in the prisoner's dilemma.

In the original proposal of Tennenholtz [11], comparison between program strategies was only considered as *textual comparison*, i.e., string comparison of program strategy text. While this has the advantage of being computationally simple and straightforward to implement, it does have some important limitations. For example, it is very natural to consider the possibility of richer comparisons between programs strategies; so that one can say "I'll cooperate if his program *has the same behaviour as mine*". This conditions behaviour on whether program strategies are *semantically* equal, rather than *syntactically* equal. Of course, such an approach is fraught with difficulties: we would have to define some appropriate notion of semantic equivalence, and such program reasoning is notoriously complex, both computationally and mathematically.

In this paper, we investigate an approach to program equilibrium in which richer comparisons between program strategies are allowed, based on *model checking* [5] – one of the most successful practical approaches to reasoning about programs. We introduce a decision-tree model of strategies, which may be conditioned on strategies of others. We then formulate and investigate a notion of "outcome" for our setting. We show that, in the prisoner's dilemma, for example, the approach permits mutual cooperation as an outcome, while not requiring that program strategies are syntactically equal. With respect to formal results, we investigate the complexity of reasoning about outcomes, and show that checking the existence of outcomes is NP-complete in general. Our approach is illustrated with many examples.

2 Setting the Scene

In his seminal proposal [11], Tennenholtz proposed the idea of players in a game entering complex strategies expressed as programs that may be conditioned on the program strategies of others. In [11], the conditions permitted on programs were restricted to be comparisons of program text; that is, comparing whether the "source files" for two programs were the same. Using this scheme, Tenneholtz

proposed that a player in the prisoner's dilemma game should enter a program strategy as follows[1]:

```
IF HisProgram == MyProgram THEN
 DO(COOPERATE);
ELSE
 DO(DEFECT);
END-IF.
```

The intended meaning of this program strategy is straightforward: `HisProgram` is a string variable containing the text of the other player's program strategy, `MyProgram` is a string variable containing the text of the program in which it is referred, `DO(...)` indicates a commitment to choosing one of the available actions, and "==" denotes the string comparison test. Now, suppose one player enters the program above: *then the other player can do no better than enter the same program*, yielding the overall outcome of mutual cooperation as an equilibrium.

This is a remarkably simple and intuitive result, and represents one of the most compelling applications of ideas from computer science to game theory to date. It makes precise the intuition that we discussed in the opening section, namely, the idea of one player saying "I'll cooperate if he will", thereby conditioning his strategy on the strategies of others. But the idea also raises a number of questions: How does the choice of programming language affect the types of outcome that can be obtained? What other types of equilibrium might be considered in such a setting? In this paper, we focus on just one issue: how to reason about the behaviour of program strategies within the program strategy language. A key problem is that the comparison `HisProgram` == `MyProgram` is very strict. It requires that the two programs have the *same source code*, which is *not* the same as requiring that they are the *same programs*: many different source code programs can define the same program. So, as a first modification of the basic idea, let us see if we can relax the requirement for syntactic equivalence.

To make the discussion formal, we introduce some notation. Let $Ag = \{1, \ldots, n\}$ denote the set of players in the game under consideration. Let Ac denote the set of actions that may be performed by a player – in the prisoner's dilemma, for example, we have $Ac = \{C, D\}$. (For convenience, we are here assuming that all players have available the same set of actions.) Next, let $\Pi = \{\pi_1, \ldots\}$ denote the set of possible program strategies; for the moment, we do not define exactly what these are. Crucially, we distinguish a program strategy from its syntactic representation in the strategy programming language. Let $\mathcal{L}_\Pi = \{\ell_1, \ldots\}$ denote the set of possible *strategy program texts*, i.e., the set of "legal source code files" for strategy programs. Notice that strategy program texts $\mathcal{L}_\Pi$ are *syntactic objects* while strategy programs Π are *semantic objects*. It is possible for multiple source program texts to define the same program strategy. Now, a program strategy for a player i in an n player game takes as input n program strategy texts, one for each player, (including its own program text), and produces as output an action. Formally, we can understand such a program strategy as a function:

$$\pi_i : \underbrace{\mathcal{L}_\Pi \times \cdots \times \mathcal{L}_\Pi}_{n \text{ times}} \rightarrow Ac.$$

The strategy for the prisoner's dilemma can then be understood as follows:

$$\pi_i(\ell_i, \ell_j) = \begin{cases} C & \text{if } \ell_j = \ell_i \\ D & \text{otherwise.} \end{cases} \quad (1)$$

[1] We will not here formally define the "program strategy language" used by Tennenholtz – the key features should be easy to understand from the example.

Here, the test $\ell_j = \ell_i$ is a purely syntactic comparison of program texts. So, suppose we have available a function $[\![\ldots]\!]$, which gives the *denotation* of a program text. Formally:

$$[\![\ldots]\!] : \mathcal{L}_\Pi \to \Pi$$

that is to say:

$$[\![\ldots]\!] : \mathcal{L}_\Pi \to (\underbrace{\mathcal{L}_\Pi \times \cdots \times \mathcal{L}_\Pi}_{n \text{ times}} \to Ac).$$

If the denotation function is available as a primitive in the strategy programming language $\mathcal{L}_\Pi$ itself, then we might imagine writing a "semantic" equivalent of the prisoner's dilemma program strategy as follows:

```
IF [[HisProgram]] == [[MyProgram]] THEN
 DO(COOPERATE);
ELSE
 DO(DEFECT);
END-IF.
```

This results in the following strategy program π_i for player i:

$$\pi_i(\ell_i, \ell_j) = \begin{cases} C & \text{if } [\![\ell_j]\!] = [\![\ell_i]\!] \\ D & \text{otherwise.} \end{cases} \tag{2}$$

This seems to solve the problem of requiring syntactic equality of programs, but it raises some problems of its own. It is, of course, unrealistic to have a denotation function of the type discussed above in a realistic programming language, and comparing programs is equally unrealistic. So, what can we do instead? The idea we pursue in this paper is to use simpler forms of program reasoning within the language, and in particular, to allow programs to reason about each other's behaviour using *model checking* [5]. Model checking is an extremely successful technology for analysing the behaviour of (typically finite state) systems and protocols by using temporal logic. The basic idea is that the state transition graph of a system $\mathcal{S}$ can be understood as a model $\mathcal{M}_\mathcal{S}$ for a temporal logic, and so checking whether the system $\mathcal{S}$ satisfies a property φ expressed using the temporal logic amounts to evaluating the model checking problem $\mathcal{M}_\mathcal{S} \models \varphi$. Crucially, such evaluation can typically be done much more efficiently than program reasoning of the kind suggested by (2), above. So, suppose we allow our programs to reason about each other's behaviour using model checking. We might then imagine writing a program strategy such as the following[2]:

```
IF HisProgram(MyProgram, HisProgram) |= DO(COOPERATE) THEN
 DO(COOPERATE);
ELSE
 DO(DEFECT);
END-IF.
```

Notice one important subtlety in this definition: when evaluating the behaviour of `HisProgram` using the model checking query, the parameters `MyProgram` and `HisProgram` are passed. This is because we need to understand how `HisProgram` will behave *when faced with* `MyProgram`. The basic idea of the remainder of the paper is to explore this idea in more detail; since using a "real" programming language would lead to great technical complexity (and almost certainly high computational complexity), we define a "simpler" strategy programming language – intuitively, based on decision trees.

[2]The notation used in this example intended to be suggestive only; don't take it too seriously.

3 The Formal Framework

First, some preliminary definitions. We assume a set $Ag = \{1, \ldots, n\}$ of *agents* ($|Ag| = n$, $n > 0$). Each agent $i \in Ag$ has a repertoire $Ac_i = \{\alpha_1, \ldots, \alpha_k\}$ of *actions* (moves) that it can perform. To keep the model simple, we will assume there are no pre-conditions attached to actions: every one of an agent's actions can be performed in every situation. We do not require that action sets Ac_i are disjoint. We assume that a "noop" action *null* is included in every action set.

3.1 Strategies

We now define strategies in our framework. Conventionally, a strategy is assumed to be a kind of conditional plan, which defines how an agent is to act in every possible circumstance. Typically, a strategy is modelled as a function that maps a history of the system/game to date to a chosen action. In our framework, inspired by the work of [11] as described above, strategies have a rather different feel. Our strategies allow the behaviour of agents to be mutually conditioned on each other; and in particular, we allow a strategy for agent *i* to be conditioned on the past *and future* behaviour of other agents, allowing us to naturally, explicitly, and formally define strategies along the lines of "I will cooperate if he will". The difficulty is making this idea precise, and in particular, in defining the notion of a "sensible" outcome when we may have a circular chain of conditions on agent behaviour (e.g., where *i*'s strategy is conditional on *j*'s strategy, and *j*'s strategy is conditional on *i*'s strategy). We define the notion of a *stable outcome* to capture the notion of a reasonable outcome in such a setting, and investigate the properties of stable outcomes.

Technically, strategies in our framework look very much like decision trees, with an unusual kind of condition on transitions. More formally, a strategy is a complete binary tree[3] in which vertices are labelled with actions and transitions/edges in the tree are labelled with *transition guards*. A transition guard is a predicate on the behaviour of agents in the system. Suppose an edge (s, s') in the strategy tree for agent *i* is labelled with a transition guard Φ, and s' is labelled with the action α. Then any outcome in which (s, s') appears must satisfy Φ; and thus any outcome in which *i* performs α in s' must satisfy Φ. The condition Φ is not just a predicate on the *past* behaviour of agents in the system: it may also refer to their *future* behaviour. The transition guard Φ may in fact also refer to the behaviour of the agent *i* itself, giving strategies a somewhat self-referential flavour.

We formally define the language for transition guards later, but be assured that it *will* be rigorously defined. For now, it suffices to note that $\mathcal{L}_C$ will denote the set of formulae of the language, and that the language contains the usual Boolean connectives ($\wedge, \vee, \neg, \ldots$) with classical semantics. We write $\Phi \equiv \Psi$ to mean that $\Phi, \Psi \in \mathcal{L}_C$ are equivalent – again, this notion will be formally defined later, but the formulation is relatively conventional.

Formally, a *strategy*, σ_i, for agent $i \in Ag$ is a structure:

$$\sigma_i = \langle S_i, R_i, s_i^0, C_i, L_i \rangle$$

where:

- S_i is a (finite, non-empty) set of strategy *states*;
- $R_i \subseteq S_i \times S_i$ is a binary *transition relation* on S_i, such that $R_i \neq \emptyset$;

[3] The decision to restrict strategies to binary decision trees means that we cannot directly represent multiway selections in our strategies. This restriction *greatly* simplifies the constraints that strategies are required to satisfy, and the subsequent technical presentation. The binary tree assumption is *not* an essential part of our framework; it just makes the exposition simpler.

- $s_i^0 \in S_i$ is the *initial state*;
- $C_i : R_i \rightarrow \mathcal{L}_C$ labels each edge in R_i with a transition guard;
- $L_i : S_i \rightarrow Ac_i$ associates an action with each state;

such that:

- (S_i, R_i) forms a complete binary tree with root node s_i^0;
- $L_i(s_i^0) = null$;
- if $\{(s, s'), (s, s'')\} \subseteq R_i$ then $C_i(s, s') \equiv \neg C_i(s, s'')$.

The final condition ensures that branching in the tree has an "`if...then...else...`" form. For this reason, we often write the transition guard on the second outgoing edge of a node as "`else`", understanding that it is the negation of the condition on the first outgoing edge.

Let $leaves(\sigma_i)$ denote the set of leaf nodes of σ_i.

3.2 Encounters and Outcomes

An *encounter* captures the idea of a collection of agents meeting, each with a strategy, conditioned on the strategies of others. Formally, an encounter is a structure:

$$E = \langle Ag, Ac_1, \ldots, Ac_n, \sigma_1, \ldots, \sigma_n \rangle$$

with components as previously discussed; we will require that all strategies have the same length and breadth. Our goal is now to define what we mean by the "outcome" of a such an encounter. We start with the notion of an *individual outcome*. An individual outcome for an agent $i \in Ag$ is simply a path through i's strategy tree, starting from the initial state s_0^i and ending in a leaf node. Formally, an individual outcome ω_i for $i \in Ag$ with strategy σ_i is a sequence of states

$$\omega_i = (s_0^i, \ldots, s_k^i)$$

such that:

- s_0^i is the initial state of σ_i;
- $s_k^i \in leaves(\sigma_i)$; and
- $\forall j \in \mathbb{N}$ s.t. $0 \leq j \leq k-1$, $(s_j^i, s_{j+1}^i) \in R_i$.

When it is clear from the context whose strategies we are talking about, or when the agent is not relevant, we will also write ω for the outcome and $s_0, s_1, \ldots$ for the states. Let $|\omega_i|$ denote the length of ω_i. We denote the element of ω_i at position $u \in \mathbb{N}$, $(0 \leq u < |\omega_i|)$ by $\omega_i[u]$. Thus $\omega_i[0]$ is the initial state of ω_i, and $\omega_i[|\omega_i| - 1]$ is the final state. Let Ω_i denote the set of individual outcomes for i (over E).

A *collective outcome* $\varpi = \langle \omega_1, \ldots, \omega_n \rangle$ is then simply a tuple of such individual outcomes, one for each agent $i \in Ag$. Let Ω_E denote the set of collective outcomes for encounter E. Notice that the definition of a collective outcome says nothing about whether an outcome is "reasonable" or not; we address this issue later.

	a_2	b_2
a_1	o_1	o_2
b_1	o_4	o_5

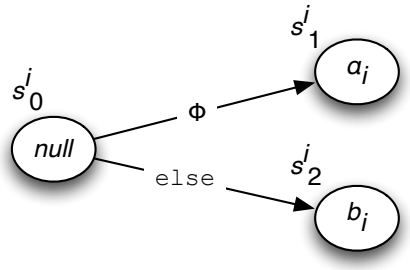
	c_2	d_2
c_1	$(3,3)$	$(0,5)$
d_1	$(5,0)$	$(1,1)$

Figure 1: An abstract two-player strategic game (left) and the Prisoner's Dilemma as an instance (right)

Example 1 (Strategic Game Form, Prisoner's Dilemma) *Strategic games are a simple yet important class of games. In a simplest setting, we have two agents, $A = \{1, 2\}$, each with two actions, $Ac_i = \{a_i, b_i\}$. In such a game, actions and strategies coincide: the game's duration is only one joint action. For representation of such a game, see the table on the left hand side of Figure 1, where o_1 is the collective outcome, or state, that is the result of 1 doing a_1 while 2 chooses a_2, etc.*

An instance of such a strategic game is obtained in the Prisoner's Dilemma. Here, two prisoners face the choice of either cooperating (c_i), or defecting (d_i), when interrogated seperately about a suspected crime. The sentence for their crime would normally be five years, but for this, there must be sufficient evidence. The possibilities available to the prisoners are as follows: if one prisoner, say agent 1, cooperates with the other prisoner and remains silent during interrogation (he plays c_1), then there are two possibilities. Either agent 2 also cooperates (in which case there is lack of evidence, and both prisoners will be sentenced for two years, a gain of 3 over the worst possible sentence). If however prisoner 2 defects, and declares the other prisoner to be guilty, prisoner 1 will get his five year sentence (a payoff of 0) and prisoner 2 will walk free (a payoff of 5). This situation corresponds with the outcome $(0, 5)$ in the right hand side of Figure 1 for the entry $\langle c_1, d_2\rangle$. The game is symmetric, and finally, if both players defect and give evidence about each other, they will be each locked up for four years, a payoff of 1.

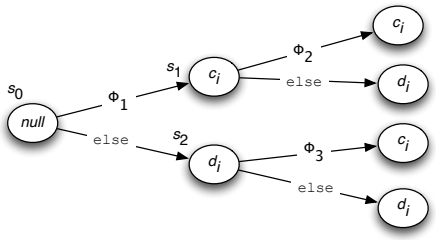
The strategy for player i in our framework may be depicted as follows:



Here, Φ is a formula from our transition guard language $\mathcal{L}_C$ to be introduced shortly.

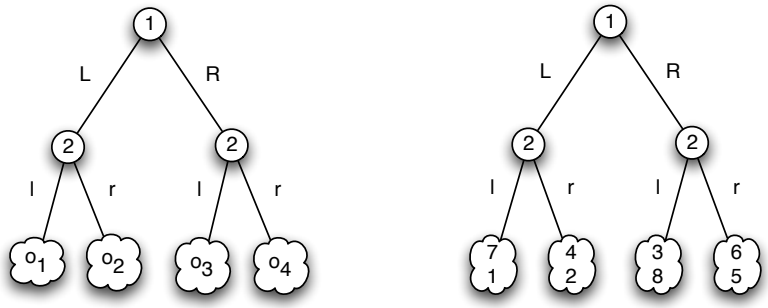
The individual outcomes for agent 1 of the strategy depicted above are (s^1_0, s^1_1) (the agent plays a_1) and (s^1_0, s^1_2) (he plays b_1). An example of a collective outcome is $\langle (s^1_0, s^1_1), (s^2_0, s^2_1)\rangle$ (1 plays a_1 and 2 plays a_2): there are four collective outcomes possible in total.

Example 2 (Iterated Prisoner's Dilemma) *As the name suggests, in the Iterated Prisoner's Dilemma (IPD), players play the PD game a number of times. A strategy for a two-round PD looks as follows:*

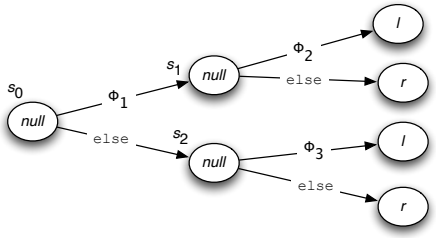


Note that the actions available after each round are the same, but the conditions to play them (Φ_1, Φ_2, Φ_3) or not, may be different.

Example 3 (Extensive Game Forms) *An example of game in extensive form is given below: again, we have two players (1 and 2), and this time $Ac_1 = \{L, R\}$ and $Ac_2 = \{l, r\}$. In the two games below, player 1 plays first, and once he is finished, player 2 makes a move. Then, the game is finished. In the game on the left, if player 1 plays L and then 2 chooses r, the game ends in outcome o_2. A concrete instance of this outcome is given on the right: 4 for player 1, and 2 for player 2.*



One possible way to model the strategies for player 2 for this game is as in the figure below. In fact, as we will see shortly, the strategy given here is more general than for the specific game: we will shortly see how specific choices for the Φ_i's make the strategies suitable for the game in extensive form.



3.3 A Logic for Individual Outcomes

Recall that earlier, we mentioned the logic $\mathcal{L}_{\mathcal{C}}$, which is used for transition guards in strategies. We can now reveal exactly what $\mathcal{L}_{\mathcal{C}}$ is: it is a logic for expressing properties of collective outcomes ϖ. To define $\mathcal{L}_{\mathcal{C}}$, we first define a logic $\mathcal{L}_{\mathcal{I}}$ for expressing the properties of individual outcomes. $\mathcal{L}_{\mathcal{I}}$

is in fact a linear temporal logic with tense modalities for referring to the past and future [6]. The primitive operators of $\mathcal{L}_{\mathcal{I}}$ are of the form $do(\alpha)$, with the fairly obvious interpretation "action α is performed". These operators are combined with the classical Boolean connectives ($\wedge, \vee, \neg, \ldots$), and with the future-time tense operators "$\bigcirc$" (next), "$\Diamond$" (eventually), and "$\Box$" (always), as well as the past-time counterparts of these operators, $\bullet$, $\blacklozenge$,$\blacksquare$. The syntax of $\mathcal{L}_{\mathcal{I}}$ is defined by the following grammar:

$$\varphi ::= do(\alpha) \mid \neg\varphi \mid \varphi \vee \varphi \mid \bullet\varphi \mid \bigcirc\varphi \mid \Diamond\varphi \mid \blacklozenge\,\varphi.$$

where $\alpha \in Ac_1 \cup \cdots \cup Ac_n$. The remaining Boolean connectives are defined as abbreviations in the usual way. The "always" and "heretofore" operators are defined as the duals of the diamond operators:

$$\Box\varphi \mathrel{\hat{=}} \neg\Diamond\neg\varphi \qquad \blacksquare\varphi \mathrel{\hat{=}} \neg\blacklozenge\,\neg\varphi.$$

Formulae of $\mathcal{L}_{\mathcal{I}}$ are interpreted with respect to a strategy σ_i, an individual outcome $\omega_i \in \Omega_i$, and a temporal index $u \in \mathbb{N}$, via the satisfaction relation $\models_{\mathcal{I}}$:

$$\begin{array}{ll}
\sigma_i, \omega_i, u \models_{\mathcal{I}} do(\alpha) & \text{iff } L(\omega[u]) = \alpha \\
\sigma_i, \omega_i, u \models_{\mathcal{I}} \neg\varphi & \text{iff not } \sigma_i, \omega_i, u \models_{\mathcal{I}} \varphi \\
\sigma_i, \omega_i, u \models_{\mathcal{I}} \varphi \vee \psi & \text{iff } \sigma_i, \omega_i, u \models_{\mathcal{I}} \varphi \text{ or } \sigma_i, \omega_i, u \models_{\mathcal{I}} \psi \\
\sigma_i, \omega_i, u \models_{\mathcal{I}} \bullet\varphi & \text{iff } u > 0 \;\;\&\;\; \sigma_i, \omega_i, u-1 \models_{\mathcal{I}} \varphi \\
\sigma_i, \omega_i, u \models_{\mathcal{I}} \bigcirc\varphi & \text{iff } u < |\omega_i| \;\;\&\;\; \sigma_i, \omega_i, u+1 \models_{\mathcal{I}} \varphi \\
\sigma_i, \omega_i, u \models_{\mathcal{I}} \Diamond\varphi & \text{iff } \exists v \in \mathbb{N}, u \leq v < |\omega_i| \text{ s.t. } \sigma_i, \omega_i, v \models_{\mathcal{I}} \varphi \\
\sigma_i, \omega_i, u \models_{\mathcal{I}} \blacklozenge\,\varphi & \text{iff } \exists v \in \mathbb{N}, 0 \leq v \leq u \text{ s.t. } \sigma_i, \omega_i, v \models_{\mathcal{I}} \varphi
\end{array}$$

Note that $\Diamond\varphi$ in fact means 'now, or sometime in the future'. We can define 'sometime in the (real) future' as $\bigcirc\Diamond\varphi$. Similarly for $\bigcirc\,\Box$ ('always in the real future'), $\bullet\,\blacklozenge$ ('sometime in the real past') and $\bullet\,\Box$ ('always in the real future'). Next, define a nullary predicate *start* $= \bullet\,\bot$. It is easily seen that *start* marks the beginning of time: it is only true when $u = 0$. (Although we did not explicitly define $\bot$, one might take $\bot = (i : do(\alpha)) \wedge \neg(i : do(\alpha))$, for a given agent symbol i and action symbol α.)

3.4 A Logic for Collective Outcomes

We now extend $\mathcal{L}_{\mathcal{I}}$ to the language $\mathcal{L}_{\mathcal{C}}$ that will be used for transition guards. Formulae of $\mathcal{L}_{\mathcal{C}}$ express properties of collective outcomes. Primitive expressions of $\mathcal{L}_{\mathcal{C}}$ are of the form $(i : \varphi)$, where $i \in Ag$ and $\varphi \in \mathcal{L}_{\mathcal{I}}$, meaning that the individual outcome ω_i for agent i satisfies φ. These expressions are combined with the classical Boolean connectives $\wedge, \vee, \neg, \ldots$. The syntax of formulae of $\mathcal{L}_{\mathcal{C}}$ is thus given by the following grammar:

$$\Phi ::= (i : \varphi) \mid \neg\Phi \mid \Phi \vee \Phi$$

where $i \in Ag$ and $\varphi \in \mathcal{L}_{\mathcal{I}}$. The semantics of the language are defined with respect to the satisfaction relation "$\models_{\mathcal{C}}$", via the following rules:

$$\begin{array}{ll}
E, \varpi, u \models_{\mathcal{C}} (i : \varphi) & \text{iff } \sigma_i, \omega_i, u \models_{\mathcal{I}} \varphi \\
E, \varpi, u \models_{\mathcal{C}} \neg\Phi & \text{iff not } E, \varpi, u \models_{\mathcal{C}} \Phi \\
E, \varpi, u \models_{\mathcal{C}} \Phi \vee \Psi & \text{iff } E, \varpi, u \models_{\mathcal{C}} \Phi \text{ or } E, \varpi, u \models_{\mathcal{C}} \Psi
\end{array}$$

We again assume the remaining Boolean connectives are defined as abbreviations. We say a formula $\Phi \in \mathcal{L}_{\mathcal{C}}$ is *satisfiable* if $E, \varpi, u \models_{\mathcal{C}} \Phi$ for some E, ϖ, u, and *valid* if $E, \varpi, u \models_{\mathcal{C}} \Phi$ for all E, ϖ, u; we indicate that Φ is valid by writing $\models_{\mathcal{C}} \Phi$. If $\models_{\mathcal{C}} \Phi \leftrightarrow \Psi$ then we say that Φ and Ψ are *equivalent*.

Example 1 (Continued) *Let us focus on the prisoner's dilemma. By taking $\Phi = \top$ we get the strategy c_i in PD, while $\Phi = \bot$ gives d_i. For i any of the agents, let $\bar{\imath}$ denote the other agent. Now consider $\Phi = (\bar{\imath} : c_{\bar{\imath}})$. This strategy for i says that i will cooperate iff $\bar{\imath}$ does. Note that nothing prevents an agent to condition his choices on his own choices, so for instance a condition $\Phi = (i : d_i)$ in the PD would correspond with a strategy in which i would cooperate if and only if he defects! Of course, we need to explain what such a strategy 'means', which we will do shortly.*

Example 2 (Continued) *When specifying strategies in the iterated PD, we can make full use of the temporal operators. For instance, a well known strategy in this game is the following. It is based on the following simple decision (also known as* reciprocal altruism *in biology):*

> TIT-FOR-TAT*: if my opponent cooperated in the previous move, then that is what I do now, if however he defected, I will defect as well.*

Of course the strategy should also prescribe what to do in the first round. Let us suppose the strategy starts with a cooperative move. Then the TIT-FOR-TAT strategy with initial cooperate move *can be given as follows: label every transition to any state labeled with c_i with the guard $copy^c$:*

$$start \vee (\bar{\imath} : \bullet\, c_{\bar{\imath}}) \tag{3}$$

(and consequently, label all other transitions with 'else'.) In a case like this, we say that the strategy is such that all c_i actions are conditioned on (3).

The strategy COPY-NOW *would merely be the game where i will do exactly what $\bar{\imath}$ will do: the guard for c_i-transitions would simply be $(\bar{\imath} : \bigcirc c_{\bar{\imath}})$*

The following strategy is less forgivefull than TIT-FOR-TAT*:*

> GRIM-TRIGGER*: I will cooperate, but as soon as my opponent defected on me I will defect as well, and never go back to cooperate again.*

The guards for transitions leading to a state labeled with d_i would be:

$$(\bar{\imath} : \bullet \blacklozenge\, d_{\bar{\imath}}) \tag{4}$$

Forgiveness usually relates to events that happened in the past, but nothing prevents agent i from conditioning his d_i actions on what $\bar{\imath}$ does now, or even in the future:

$$(\bar{\imath} : \blacklozenge\, d_{\bar{\imath}}) \vee (\bar{\imath} : d_{\bar{\imath}}) \vee (\bar{\imath} : \Diamond d_{\bar{\imath}}) \tag{5}$$

The strategy that conditions d_i on (5) prescribes the following.

> UNFORGIVING-EVER*: I will defect if my opponent ever defects — be it in the past, now, or in the future*

Of course there are many variants of this: a strategy that defects if the opponent defected the previous k rounds, for example (also known as TIT-FOR-k-TATS*):*

$$(\bar{\imath} : (d_{\bar{\imath}} \wedge \bullet\, (d_{\bar{\imath}} \wedge \ldots \wedge \bullet\, (d_{\bar{\imath}} \wedge \bullet\, d_{\bar{\imath}}) \ldots))) \tag{6}$$

or whenever the opponent defected k times in the past

$$(\bar{\imath} : \blacklozenge\, (d_{\bar{\imath}} \wedge \bullet \blacklozenge\, (d_{\bar{\imath}} \wedge \ldots \wedge \bullet \blacklozenge\, (d_{\bar{\imath}} \wedge \bullet \blacklozenge\, d_{\bar{\imath}}) \ldots))) \tag{7}$$

Example 4 (Multiple Player Games) *Let us briefly look at games where the number of players is n. There are many strategies that one can think of in such situation. For instance, agent i could condition an action a_i on the majority (if any) chosing that action (now, or in the previous move, or if a was the most popular action choosen thus far). Another conidition for action a_i might be that everybody choose a, etc.*

4 Stable Outcomes

Suppose we are given an encounter E. How are we to identify the "reasonable" collective outcomes of E? The key solution concept we define is the notion of a *stable* outcome. A stable outcome is one in which every transition guard on every transition in every individual outcome is satisfied; and thus all the mutually conditioned constraints imposed by agents are satisfied. Formally, if $\varpi = \langle \omega_1, \ldots, \omega_n \rangle \in \Omega_E$ then ϖ is said to be *stable* if it satisfies the following condition:

$$\forall i \in Ag,\ \forall u \in \mathbb{N} \text{ s.t. } 0 \leq u < |\omega_i|, \quad E, \varpi, u \models_{\mathcal{C}} C_i(\omega_i[u], \omega_i[u+1]).$$

Given an encounter E, let $stab(E)$ denote the set of stable outcomes of E. In other words, an encounter is stable if every condition for doing an action along a path, is true.

Example 1 (Continued) *Let us look at the one shot PD game explained earlier. Let the condition for doing c_i for each agent i be Φ_i.*

1. *First suppose $\Phi_1, \Phi_2 \in \{\top, \bot\}$. Each combination of such guards gives rise to one stable outcome: the four combinations in total account for the table at the right-hand-side of Figure 1. More specifically, consider $\Phi_1 = \top$ and $\Phi_2 = \bot$ then the only stable outcome is $\langle (s_0^1, s_1^1), (s_2^0, s_2^2) \rangle$ (this corresponds to the outcome (c_1, d_2), or (0,5), in our example): similarly for the other choices for conditions from $\{\top, \bot\}$.*

2. *Suppose $\Phi_1 = \top$. If $\Phi_2 = (1 : \bigcirc do(c_1))$, this gives rise to the stable outcome $\langle (s_0^1, s_1^1), (s_2^0, s_1^2) \rangle$ (both players cooperate: $\langle c_1, c_2 \rangle$). If $\Phi_2 = (2 : \bigcirc do(c_2))$ then there are* two *stable outcomes, corresponding to the action profiles $\langle c_1, c_2 \rangle$ and $\langle c_1, d_2 \rangle$.*

3. *Suppose $\Phi_1 = (1 : \bigcirc do(c_1))$. If $\Phi_2 = (2 : \bigcirc do(c_2))$, then* all *possible collective strategies are stable: If agent i plays c_i, it is 'justified' by the condition Φ_i, and if i plays d_i, it is justified by the alternative $\neg\Phi_1$.*

4. *Finally, suppose $\Phi_1 = (2 : \bigcirc do(c_2)$ ('I cooperate if he cooperates'). If $\Phi_2 = (1 : \bigcirc do(c_1))$ then the two stable outcomes correspond to the action profiles $\langle c_1, c_2 \rangle$ and $\langle d_1, d_2 \rangle$.*

Notice that the final example above illustrates the straightforward encoding of Tennenholtz's program for the prisoner's dilemma in our setting. As in Tennenholtz's setting, mutual cooperation is a stable outcome, and if $\Phi_i = (\bar{\imath} : \bigcirc do(c_{\bar{\imath}}))$ then agent i cannot suffer the "suckers payoff", where i cooperates and $\bar{\imath}$ defects. However, the benefit of our approach is that we are no longer reliant on the syntactic form of the strategy; for example if player 2 used the syntactically different, but semantically equivalent condition $\Phi_2 = \neg\neg(1 : \bigcirc do(c_1))$, then the result would be the same: mutual cooperation remains a stable outcome.

Example 2 (Continued) *Suppose prisoner 1 plays* TIT-FOR-TAT. *If 2 plays this as well, a stable outcome is one in which both prisoners cooperate, all along the game. This is also the only stable outcome: if a prisoner would defect, this is not justified by the condition (3). If prisoner 2 plays* GRIM-TRIGGER, *the unique stable outcome is the same and this also holds when 2 plays* UNFORGIVING-EVER *(5). Suppose now 1 is of type* TIT-FOR-TAT, *while 2 is like that, but he starts with a defecting move:*

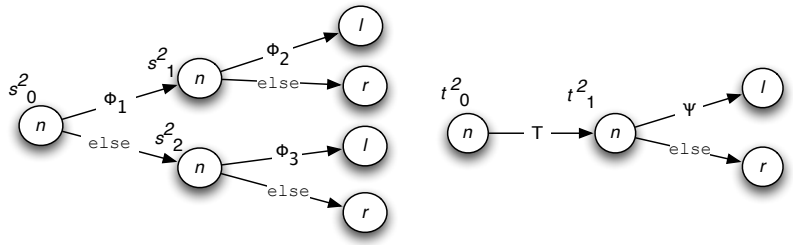
$$\neg start \vee (1 : \bullet c_1) \tag{8}$$

In this case, the stable outcome is the one in which 1 plays c_1 in every odd round and d_1 in every even round, while 2 plays d_2 in odd rounds and c_2 in even rounds.

Let us now drop the assumption that a prisoner plays TIT-FOR-TAT. *Assuming that both prisoners use* GRIM-TRIGGER, *the stable outcomes are again plays in which only c_i is used. In general, when 2 uses* GRIM-TRIGGER *stable outcomes are plays of length n such that 2 and 1 play k times c_i, 2 plays d_2 at step $k+1$ while 1 still plays c_1, and from $k+1$ on, 2 only plays d_2 (what 1 will do in the game from $k+1$ on, depends on his strategy). If 1 plays* TIT-FOR-TAT *or* GRIM-TRIGGER, *$k = n$, the length of the game.*

If one player is of type GRIM-TRIGGER *while the other is* UNFORGIVING-EVER, *again the only stable outcome is the one in which only players cooperate. If 1 is of type* GRIM-TRIGGER *while 2 if of type* UNFORGIVING-EVER, *however, there are two stable outcomes: the one in which only cooperation is played, and the one in which 1 uses c_1 as a first move, while other moves of both players are d.*

Example 3 (Continued) *Let us look at strategies for the game in extensive form of this example. Take the strategy for agent 2 again, depicted at the left of the following figure (in which n represents 'null'). Φ_1 could be $(1 : \bigcirc do(R))$, in which case state s_1^2 corresponds with a state in which agent 1 has played R, and s_2^2 corresponds with a state where 1 played L.*

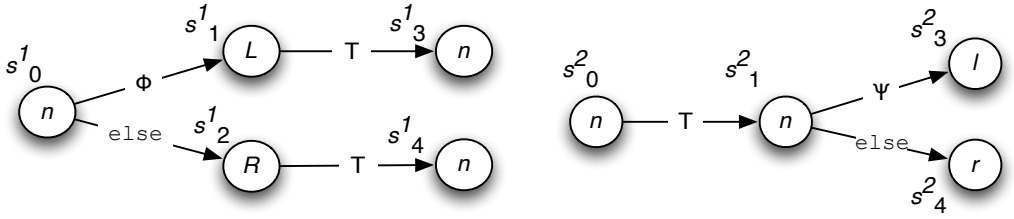


The diagram on the right of the figure shows a transformed version of the strategy on the left. First of all, we adopt the convention that if an edge has an outgoing arc labelled $\top$, then we do not need to represent the alternative outgoing edge (so, R_i is strictly speaking not binary any longer, but it is easy to add a non-significant node with a transition to it, a transition that will never be taken).

More importantly, another difference between the two strategies depicted above is that, in the strategy on the right hand side, the condition Ψ to play l is uniform, in the sense it seems to be regardless of what has happened earlier. However, we can cater for this in our object language as follows. Suppose $\Phi_1 = (1 : \bigcirc do(R))$. Then, with choosing

$$\Psi = ((1 : \bullet do(R)) \wedge \Phi_2) \vee ((1 : \bullet do(L)) \wedge \Phi_3)$$

we would get 'a similar' strategy. This claim needs to be made precise and can be generalised, but for the moment we will use strategies as depicted on the right when talking about extensive game forms. The two strategies for the agents would then look as follows:



So what are stable outcomes in this game? Let us first assume agent 2 only cares about his own payoff: $\Psi = (1 : \bullet do(R))$. *For agent 1, let us first assume that he knows that 2 is rational:* $\Phi = \bot$. *The resulting encounter has one stable outcome: and leads to the outcome in which agent 1 obtains 3, and 2 obtains 8. Next, suppose* $\Phi = \neg((1 : do(L)) \rightarrow (2 : \bigcirc do(r)))$ *('I play R iff playing L would imply that would 2 answer with playing r'). Now there are two stable outcomes:* $\langle (s_0^1, s_1^1, s_1^1), (s_0^2, s_1^2, s_4^2) \rangle$ *corresponding to 1 playing L, and 2 playing r, and the second stable outcome being* $\langle (s_0^1, s_2^1, s_4^1), (s_0^2, s_1^2, s_3^2) \rangle$, *corresponding to 1 playing R and 2 playing l.*

Keeping the same Ψ, *we can also identify a condition* Φ *for which there are three stable outcomes:* $\Phi = (1 : do(L))$. *Those stable outcomes correspond to the plays* $(L, l), (L, r)$ *and* (R, r). *Finally, it is easy to see that for* $\Phi = (1 : do(L))$ *and* $\Psi = (2 : do(l))$ *all four possible global outcomes are stable.*

Example 4 (Continued) *We briefly revisit the example with several players. Suppose each agent can do two actions (or votes), say yes and no. We specify the condition* Φ_i *leading to the yes node. Suppose for every i,*

$$\Phi_i = All_i = \bigwedge_j (j : do(yes))$$

('I vote yes iff everybody does'.) The the only stable outcomes are those in which voting happens unanomously: either all vote 'yes', or all vote 'no'. (Let us call this set of (two) collective strategies U.) To see this, note that in any other collective outcome, there is a 'yes' vote and a 'no' vote. Then the agent i voting yes does not satisfy his condition All_i *to do so. How about*

$$\Phi_i = All\text{-}Other_i = \bigwedge_{j \neq i} (j : do(yes))$$

('I vote yes iff everybody else does'.) Certainly, the global outcome in which everybody votes yes is stable. In fact, everybody using $All\text{-}Other_i$ *again leads to the set U of stable outcomes. U is also obtained if* $\Phi_i = Some_i = \bigvee_j (j : do(yes))$ *and* $Some\text{-}Other_i = \bigvee_{j \neq i} (j : do(yes))$.

Suppose n is odd, say $n + 1 = 2 \cdot k$, *then we can define*

$$\Phi_i = Maj_i = \bigvee_{j_1, .. j_k}^{j_x \neq j_y} \bigwedge_{x=1}^{x=k} (j_x : do(yes))$$

('I vote yes iff a majority does'). Maybe surprisingly, the stable outcomes are again those in U.

Let us finally assume that not all agents use the same condition. Suppose $n+1 = 2k$, and the first k agents use All_i and the remaining $k-1$ agents use Maj_i. Again the stable set is U. However, if we assume the first k agents use Maj_i and the remaining k_1 agents use All_i, the stable set consists U together with the collective outcome in which exactly the first k agents vote yes.

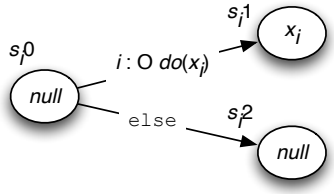
From the discussion above, we can distill the following conclusions.

Theorem 1 *(1) There exist encounters E such that $stab(E) = \emptyset$. (2) There exist encounters E such that $|stab(E)| = 1$. (3) There exist encounters E such that $|stab(E)| > 1$. (4) Let $E = \langle Ag, Ac_1, \ldots, Ac_n, \sigma_1, \ldots, \sigma_n \rangle$ be an encounter. Then it is possible to change the labeling function C_i in the strategy σ_i for agent i in such a way, that the resulting encounter E' satisfies the property that all collective outcomes are stable.*

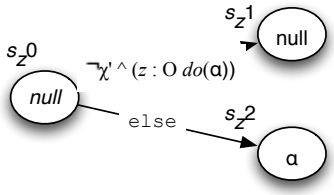
Now, consider the decision problem NON-EMPTY STABLE SET, in which we are given an encounter E, and asked whether or not $stab(E) \neq \emptyset$.

Theorem 2 NON-EMPTY STABLE SET *is* NP*-complete.*

Proof: Membership is by "guess-and-check". For hardness, we reduce SAT. Given a SAT instance χ over Boolean variables $x_1, \ldots, x_k$, which we assume w.l.o.g. is in CNF, we construct an encounter E_χ as follows. For each Boolean variable x_i, create an agent i with $Ac_i = \{x_i, null\}$, and define σ_i to be:



Then create an agent z with $Ac_z = \{null, \alpha\}$. Construct an $\mathcal{L}_\mathcal{C}$ formula χ' by transforming the SAT instance χ as follows: systematically substitute for each positive literal x_i the $\mathcal{L}_\mathcal{C}$ expression $(i : \bigcirc do(x_i))$, and for each negative literal $\neg x$ substitute $(i : \bigcirc do(null))$. Now σ_z to be:



We claim that $stab(E_\chi) \neq \emptyset$ iff the χ is satisfiable. ($\Leftarrow$) Assume $stab(E_\chi) \neq \emptyset$. Now, consider agent z. First, notice that the individual outcome (s_z^0, s_z^1) cannot be in any stable outcome, since this would require that in the next state agent z does α, whereas in fact it does *null*. So, the individual outcome for z contained in the stable outcome must be (s_z^0, s_z^2), and hence the collective outcome must satisfy $\neg(\neg\chi' \wedge (z : \bigcirc do(\alpha))) = \neg\neg\chi' \vee \neg(z : \bigcirc do(\alpha)) = (z : \bigcirc do(\alpha)) \rightarrow \chi'$. Since the $L(s_z^2) = \alpha$ this implies that the antecedent of this condition is satisfied, hence the collective outcome must satisfy χ'. This immediately implies that χ is satisfiable. ($\Rightarrow$) Assume χ is satisfiable. Then let $X \subseteq \{x_1, \ldots, x_k\}$ be the set of variables made true under some satisfying assignment for χ. We construct a stable

outcome $\varpi_\chi(\omega_1, \ldots, \omega_n)$ for E_χ as follows. First, for each agent i corresponding to variable x_i, if $x_i \in X$ then $\omega_i = (s_i^0, s_i^1)$, while if $x_i \notin X$ then $\omega_i = (s_i^0, s_i^2)$. Finally, set $\omega_z = (s_z^0, s_z^2)$. We claim that the outcome ϖ_χ thus constructed is stable. The only non-obvious part is for the individual outcome ω_z: here the point is that the construction of the formula χ' and outcomes for variable agents $1, \ldots, k$ are such that the outcomes for $1, \ldots, k$ will satisfy χ', ensuring that the guard on the transition (s_z^0, s_z^2) is satisfied. ∎

Notice that in the proof of Theorem 2, the encounter E_χ that we construct for χ is such that $stab(E_\chi)$ contains outcomes that are in a one-to-one correspondence with satisfying assignments for χ. We may thus conclude:

Corollary 1 *Given an encounter E, the problem of computing $|stab(E)|$ is #P-complete.*

5 Related Work and Conclusions

Several other authors have begun to consider aspects of program equilibria. Kalai et al. abstract away from programs completely, and assume that each player has a "mutually conditioned commitment device" [8]. In this setting, they prove a "commitment folk theorem", analogous to the Nash folk theorem in iterated games [9, p.143]. Fortnow considered the idea of playing a game over time, and used a Turing machine model of program strategies, proving a generalised version of the folk theorem [7]. Peters and Szentes consider the issue of "definable contracts"; their idea is to use a Gödel numbering scheme for program strategies, so that a program strategy can intuitively say "I'll cooperate if his Gödel number is the same as mine" [10]. In future work, it would be interesting to consider in more detail issues such as richer, more intuitive programming languages, and the questions of what kinds of different equilibrium might be defined, and how the choice of languages affects the ability to reach such equilibria. It would also be interesting to somehow introduce preferences into the language, so that we can define generic strategies, that take into account preferences when selecting actions. And finally, of course, it would be interesting to look at the cases where finding a stable outcome is tractable.

References

[1] R. Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[2] K. Binmore. *Fun and Games: A Text on Game Theory*. D. C. Heath, 1992.

[3] K. Binmore. *Game Theory and the Social Contract Vol 1*. MIT Press, 1994.

[4] K. Binmore. *Game Theory and the Social Contract Vol 2*. MIT Press, 1998.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[6] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science Vol B*. Elsevier, 1990.

[7] L. Fortnow. Program equilibria and discounted computation time. In *Proc. TARK-09*.

[8] A. T. Kalai, E. Kalai, E. Lehrer, and D. Samet. A commitment folk theorem. *Games and Econ. Beh.*, 2009.

[9] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[10] M. Peters and B. Szentes. Definable and contractible contracts. Unpublished, 2008.

[11] M. Tennenholtz. Program equilibrium. *Games and Econ. Beh.*, 49:363–373, 2004.