Int J Game Theory
DOI 10.1007/s00182-011-0314-6

Program equilibrium—a program reasoning approach

Wiebe van der Hoek · Cees Witteveen · Michael Wooldridge

Received: 8 December 2010 / Accepted: 28 November 2011

Abstract The concept of *program equilibrium*, introduced by Howard (Theory and Decision 24(3):203–213, 1988) and further formalised by Tennenholtz (Game Econ Behav 49:363–373, 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 even in the one-shot prisoner's dilemma. In the original proposal of Tennenholtz, comparison between programs was limited to syntactic comparison of program texts. While this approach has some considerable advantages (not the least being computational and semantic simplicity), it also has 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 focus on *coherent outcomes*: outcomes in which every decision by every player is justified by the conditions in his program. We identify a condition under which there exist a unique coherent outcome. We also compare our notion of (coherent) outcome with that of (supported) semantics known from logic programming. We illustrate our approach with many examples.

W. van der Hoek (✉) · M. Wooldridge
Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK
e-mail: wiebe.Van-Der-Hoek@liv.ac.uk

C. Witteveen
Department of Software Technology, TU Delft, Mekelweg 4, 2628 CD Delft, The Netherlands

Published online: 11 December 2011

Keywords Program equilibrium · Non-cooperative games · Repeated games · Logic programming · Programs as strategies · Equality of programs

JEL Classification C7 · C72 · C6 · C60 · C62

1 Introduction

It is well-known in the game theory literature that there is an important difference between playing a game once and playing a game a number of times. The Nash equilibria of a repeated game will typically admit outcomes that are not simply iterations of the Nash equilibria of the component game. Specifically, the Nash Folk Theorems (Osborne and Rubinstein 1994, p. 143) tell us that every strategy profile leading to players obtaining better than their min max payoff can be achieved in an infinitely repeated game. The simple technical device used in the proof of Nash's Folk Theorems is a construction known as a *trigger strategy*, which can be understood as "punishing" players that do not "cooperate". Such punishment is possible in repeated games because the players will meet each other again, and players can punish a deviant player by enforcing that player's min max payoff.

The Nash Folk Theorems are perhaps most famous in their application to the *prisoner's dilemma* (see e.g., Axelrod 1984). The prisoner's dilemma is troubling because the dominant strategy equilibrium of the game (mutual 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., Binmore 1994, 1998 for extensive references and critical discussion on this topic). Repeated games, and the Nash Folk Theorems, provide a solution: in the infinitely repeated prisoner's dilemma, mutual cooperation forms an equilibrium.

In one-shot games, however, it seems that trigger strategies are not possible. The problem is that each prisoner must commit to either cooperate or defect, whereas intuitively, what each prisoner wants to do is *make his commitment contingent on the commitments of others*. The difficulty, however, is making this precise. One interesting way of doing this with computer programs was suggested by Howard (1988):

> Replacing individuals by programs in real or imaginary situations seems to us often to make a philosophical problem much clearer, and to illuminate some of its essential features. (Howard 1988, p. 204).

The idea proposed forward by Howard is to trade in the assumption of rationality of the players by the idea that opponents can in some sense "perceive the way their opponents think". They can do this because they are "given as data the computer program of their opponent" (Howard 1988, p. 206).[1] In 2004, Moshe Tennenholtz used a

[1] What Howard in fact shows is that it is possible to have a computer program that "recognises itself": he presents a program in BASIC that prints "Same Program" if it is fed with a program identical to itself, and "Different Program" otherwise.

similar idea to formalise the notion of a strategy that is contingent on the strategies of others (Tennenholtz 2004). 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 one-shot prisoner's dilemma. Tennenholtz proved a version of Nash's folk theorem for his program equilibrium setting.

In the original proposals of Howard (1988) and of Tennenholtz (2004), 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 and semantically simple, as well as being straightforward to implement, it does have some important limitations. Howard for instance notes that simply adding a number of spaces to the program source code (which would not, in most programming languages, imply that the *program* was different) would yield a player of a different "type". Given this consideration, 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 would condition behaviour on whether program strategies are *semantically* equal, rather than *syntactically* equal. However, such an approach would be fraught with difficulties. For example, we would have to define some appropriate notion of semantic equivalence, and such program reasoning is mathematically rather complex. Moreover, in general, the problem of checking whether two programs are semantically equivalent is easily seen to be undecidable.

In this paper, we investigate an approach to program equilibrium in which richer comparisons between program strategies are allowed, based on *model checking* (Clarke et al. 2000)—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 (see our discussion in Sect. 4). 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, Tennenholtz (2004) 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 Tennenholtz (2004), 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, Tennenholtz

proposed that a player in the prisoner's dilemma game should enter a program strategy as follows[2]:

```
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, but we will denote how to understand them shortly. 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

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

for each player, (including its own program text), and produces as output an action. Formally, we can understand such a program strategy for a player $i \in Ag$ 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:

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

Here, the test $\ell_j = \ell_i$ is a purely syntactic comparison of the program texts ℓ_1 and ℓ_2.

Now suppose we have available a function $[\![\ldots]\!]$, which gives the *meaning* or *denotation* of a program text. Formally:

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

that is to say:

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

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* (Clarke et al. 2000). 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[3] :

```
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.

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 (which is thought as the past of the current moment) to a chosen action. In our framework, inspired by the work of Tennenholtz (2004) 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 *coherent outcome* to capture

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

the notion of a reasonable outcome in such a setting, and investigate the properties of coherent 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 binary tree[4] 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 i.e., path in the strategy tree, 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 for now, it suffices to note that $\mathcal{L}_{\mathcal{C}}$ 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 $\Phi, \Psi \in \mathcal{L}_{\mathcal{C}}$ 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;
- $s_i^0 \in S_i$ is the *initial state*;
- $C_i : R_i \to \mathcal{L}_{\mathcal{C}}$ labels each edge in R_i with a transition guard;
- $L_i : S_i \to Ac_i$ associates an action with each state;

such that:

- (S_i, R_i) forms a complete binary tree with root node s_i^0;
- $L_i(s_i^0) = null$;
- for any $s' \neq s''$, 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.

[4] The decision to restrict strategies to binary decision trees means that we cannot directly represent multi-way 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.

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. 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

$$\omega_i = (s_i^0, \ldots, s_i^k)$$

such that:

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

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.

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, $Ac_i = \{a_i, b_i\}$. To represent such a game, see the table on the left hand side of Fig. 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 with the other prisoner (c_i), or defecting (d_i), when interrogated separately about a suspected crime. We assume the reader is familiar with the details of this example, see for instance Osborne and Rubinstein (1994, Example 16.2) or the book (Axelrod 1984).

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

	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	$(5, 0)$	$(1, 1)$

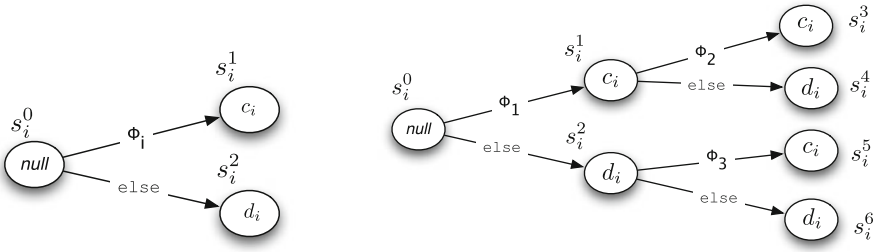


Fig. 2 Strategy for i for the one-shot PD (left) and for the 2-round PD (right)

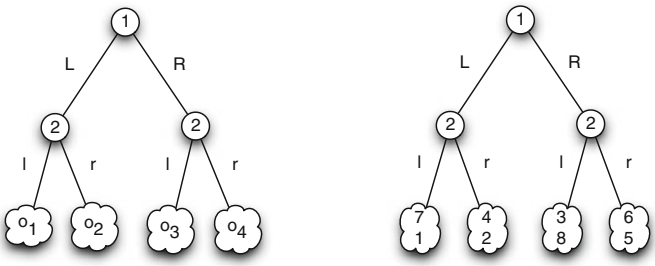
The strategies for player i in our framework may be depicted as in Fig. 2 (left). Here, Φ_i is a formula from our transition guard language $\mathcal{L}_C$ to be introduced shortly.

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 $\langle (s_0^1, s_1^1), (s_0^2, s_1^2) \rangle$ (1 plays a_1 and 2 plays a_2). We will sometimes write $\langle (a_1), (a_2) \rangle$ for such an outcome, thereby identifying a state and the action performed in the state. There are four collective outcomes possible in total.

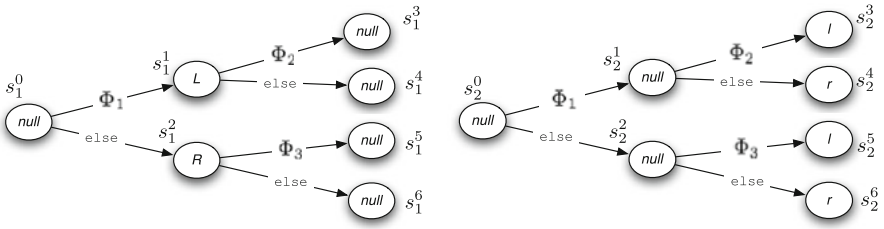
Example 2 (Iterated Prisoner's Dilemma) As the name suggests, in the Iterated Prisoner's Dilemma (IPD), players play the PD game a number of times. A strategy for a two-round PD is as shown in Fig. 2 (right). Note that the actions available after each round are the same, but the conditions to play them (Φ_1, Φ_2, Φ_3) or not, may be different.

An example of a collective outcome in this game is $\omega = \langle (s_1^0, s_1^1, s_1^4), (s_2^0, s_2^2, s_2^5) \rangle$, which we will also write as $\langle (c_1, d_2), (d_1, c_2) \rangle$. It corresponds to 1 playing first c and then d, and 2 playing first d and then c.

Example 3 (Extensive game forms) An example of a 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 players 1 (left) and 2 (right) for this game is as in the figure below (the reader may want to revisit this after having read Sect. 3.3). Note how the guards in both strategies are the same: in node s_2^0 of agent 2's strategy for example, by imposing Φ_1 as a guard, it is in fact agent 1's decision to which node s_2^1 or s_2^2, both labelled *null*, we are proceeding. (Alternatively, we could have replaced the guard Φ_1 by the condition $(1 : \bigcirc do(L))$: see Sect. 3.4 for what this means.)



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 (Emerson 1990). 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:

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

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 $\models_\mathcal{I}$:

$$\begin{array}{ll}
\sigma_i, \omega_i, u \models_\mathcal{I} do(\alpha) & \text{iff } L(\omega_i[u]) = \alpha \\
\sigma_i, \omega_i, u \models_\mathcal{I} \neg\varphi & \text{iff not } \sigma_i, \omega_i, u \models_\mathcal{I} \varphi \\
\sigma_i, \omega_i, u \models_\mathcal{I} \varphi \vee \psi & \text{iff } \sigma_i, \omega_i, u \models_\mathcal{I} \varphi \text{ or } \sigma_i, \omega_i, u \models_\mathcal{I} \psi \\
\sigma_i, \omega_i, u \models_\mathcal{I} \bullet\varphi & \text{iff } u > 0 \ \&\ \sigma_i, \omega_i, u-1 \models_\mathcal{I} \varphi \\
\sigma_i, \omega_i, u \models_\mathcal{I} \bigcirc\varphi & \text{iff } u < |\omega_i| - 1 \ \&\ \sigma_i, \omega_i, u+1 \models_\mathcal{I} \varphi
\end{array}$$

$$\begin{array}{ll} \sigma_i, \omega_i, u \models_{\mathcal{I}} \Diamond\varphi & \text{iff } \exists v \in \mathbb{N}, u \leq v < |\omega_i| \text{ s.t. } \sigma_i, \omega_i, v \models_{\mathcal{I}} \varphi \\ \sigma_i, \omega_i, u \models_{\mathcal{I}} \blacklozenge\varphi & \text{iff } \exists v \in \mathbb{N}, 0 \leq v \leq u \text{ s.t. } \sigma_i, \omega_i, v \models_{\mathcal{I}} \varphi \end{array}$$

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 $start = \neg\bullet\top$. It is easily seen that $start$ marks the beginning of time: it is only true when $u = 0$. (Although we did not explicitly define $\top$, one might take $\top = do(\alpha) \vee \neg do(\alpha)$, for a given 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. Note that transition guards for a player's strategy can therefore refer to past, current and future actions of other players and of the player itself. 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 "$\models_{\mathcal{C}}$", where a formula Φ of $\mathcal{L}_{\mathcal{C}}$ is interpreted in an encounter $E = \langle Ag, Ac_1, \ldots, Ac_n, \sigma_1, \ldots, \sigma_n \rangle$, a collective outcome φ and a temporal index $u \in \mathbb{N}$, using the interpretation of formulas $\varphi \in \mathcal{L}_{\mathcal{I}}$ in an individual strategy σ_i, an individual outcome $\varphi \in \Omega_i$ and the same index $u \in \mathbb{N}$:

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

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 $\Phi = (\bar{\imath} : \bigcirc do(c_{\bar{\imath}}))$. 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 $\Phi = (i : \bigcirc do(d_i))$ in the PD would correspond with a strategy in which i would cooperate if and only if he defects! Of course, we need to formally explain what such a strategy "means", which we will do shortly, when we define the notion of coherent outcome.

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:

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

(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 $(\bar{\imath} : \bigcirc do(c_{\bar{\imath}}))$.

The following strategy is less forgiving than TIT- FOR- TAT:

GRIM- TRIGGER: I will cooperate, but if my opponent defects, I will defect as well, and never cooperate again.

(Notice that this is a trigger strategy, as discussed in the context of Nash Folk Theorems in Sect. 1.)

In the following, it is easier to specify $\neg\Phi$, i.e., the guard for playing d_i. For GRIM-TRIGGER the guard for playing d_i would be:

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

Forgiving 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:

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

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):

$$(\bar{\imath} : (do(d_{\bar{\imath}}) \wedge \underbrace{\bullet (do(d_{\bar{\imath}}) \wedge \cdots \wedge \bullet (do(d_{\bar{\imath}}) \wedge \bullet do(d_{\bar{\imath}}))\ldots)}_{k-1 \text{ times}})) \tag{6}$$

or whenever the opponent defected k times in the past

$$(\bar{\imath} : \blacklozenge (do(d_{\bar{\imath}}) \wedge \underbrace{\bullet\blacklozenge (do(d_{\bar{\imath}}) \wedge \cdots \wedge \bullet\blacklozenge (do(d_{\bar{\imath}}) \wedge \bullet\blacklozenge do(d_{\bar{\imath}}))\ldots)}_{k-1 \text{ times}})) \tag{7}$$

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

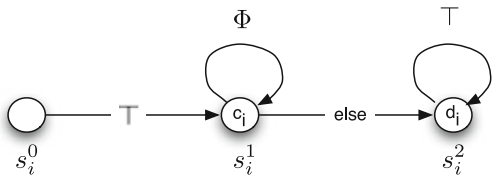


Fig. 3 A PD strategy for i, starting with a number of c_i-moves, and ending with d_i-moves

instance, agent i could condition an action a_i on the majority (if any) choosing that action (now, or in the previous move, or if a was the most popular action chosen thus far). Another condition for action a_i might be that everybody choose a, etc.

3.5 A more general set-up

We have already mentioned the assumption that players have two moves available at every node. We should emphasise that there is no technical reason for this constraint: it was made for simplicity of exposition. Note that we made another assumption about our strategies, namely that (1) they consist of *finitely* many states and (2) can be represented as a *binary tree*. This implies that we can only implement only finite games.

It would not be difficult to relax one of the requirements (1) and (2). Without giving the complete details, an example of a strategy for i in the PD under a more liberal regime is given in Fig. 3. This strategy for player i is such that i first plays a number of c_i-moves, until we reach a point where Φ is true, after which i only defects. If the condition Φ is chosen to be $(\bar{\imath} : \blacklozenge do(d_{\bar{\imath}}))$, the strategy obtained is exactly GRIM TRIGGER. Player i starts with cooperate, then keeps on cooperating as long as $\bar{\imath} : \blacklozenge (do(d_{\bar{\imath}})$ (the other player has defected) becomes true, and from then on, i only defects. Let us call such strategies *binary*: they are graphs, with one state s^0 with $in\text{-}degree(s^0) = 0$ (the starting state) and for all states s, $1 \leq out\text{-}degree(s) \leq 2$ (there are at most two alternatives). Moreover, edges are labeled: if there are two outgoing edges from s, then their labels Φ_1 and Φ_2 are such that $\Phi_1 \vee \Phi_2$ is a tautology, if there is only one outgoing edge, the label $\Phi = \top$.

The notion of an outcome ω_i can easily be extended for strategies that are represented by such graphs: an outcome is an infinite sequence of states, some of which may appear infinitely often. Allowing such outcomes would have the advantage that the temporal operators would be interpreted on more "standard" models for temporal logic, where models usually represent an infinite stream of time. Technically speaking, $\bigcirc\top$ is a validity in temporal logic (there is always a next state) where it is not in our strategies as defined in Sect. 3.2. Using the graphs (rather than finite binary trees) to represent strategies would naturally pave the way to represent infinite games, of which the strategy of Fig. 3 is in fact an example. Taking $\Phi = (\bar{\imath} : c_{\bar{\imath}})$, it is not hard to see for instance, that if such a strategy were to play against itself, the two coherent outcomes would be those in which both players either always cooperate, or cooperate once and then always defect. So this would yield an example of an infinite game that has a finite representation using our graphs. However, like with using machines, there are strategies however that cannot be represented by a finite graph: as an example, consider the strategy that says "I start with cooperate, and continue to cooperate: however, every defect-move by my opponent will be replied to with two defects on my

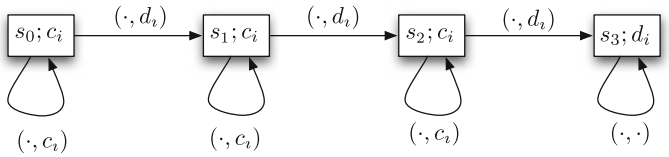


Fig. 4 A machine for the PD strategy in which i cooperates until $\bar{\imath}$ has defected 3 times

behalf'. Such a strategy would entail that if the opponent $\bar{\imath}$ starts with n defect moves and from then on always cooperates, player i would need to respond with $2n$ defect moves. But remembering to play $2n$ defect moves requires $2n$ different states, which is impossible to do for all n if we only have a finite graph.

Consider the strategy of Fig. 3 again. If one takes as the condition for playing d_i, that is for $\neg\Phi$ the formula $(\bar{\imath} : \blacklozenge(do(d_{\bar{\imath}}) \wedge \bullet\blacklozenge(do(d_{\bar{\imath}}) \wedge \bullet\blacklozenge(do(d_{\bar{\imath}})))))$ for example, the strategy amounts to the one where i will cooperate until $\bar{\imath}$ has defected 3 times (see (7)). One can also implement such a strategy using the machines as introduced in Osborne and Rubinstein (1994, Sect. 8.4). Such an implementation is given in Fig. 4. In the four states of that machine for GRIM TRIGGER, following Osborne and Rubinstein (1994) we denote the name s_i of the state, and then the action taken (d_i in s_3, and c_i in the other states). The transitions are labeled with guards, which in the case of machines, are sets of strategy profiles: $(\cdot, c_{\bar{\imath}})$ for instance denotes the set $\{(c_i, c_{\bar{\imath}}), (d_i, c_{\bar{\imath}})\}$. In our notation, this would correspond with the condition $\bar{\imath} : \bigcirc do(c_{\bar{\imath}}))$. So, guards of a machine only refer to the "current" profiles, there is no reference to past or future actions. This implies that we can represent any strategy that is represented as a machine as a program in our framework.

Using a machine, one can also encode at least *some* past behaviour of other agents in the states though: in Fig. 4 for instance, the state s_n represents the fact that thus far, $\bar{\imath}$ has defected n times. Thus representing the strategy (7) as a machine will take $k+1$ states if i is willing to allow $\bar{\imath}$ to defect k times (those states are needed to count the number of defect-moves by $\bar{\imath}$), whereas in our graph representation, any strategy for i that allows $\bar{\imath}$ to defect k times before i plays defect, could be represented in the graph of Fig. 3: the counting will be done in the guard Φ! It is not directly clear how for instance a condition for playing an action that refers to the future (like $(\bar{\imath} : \Diamond do(d_{\bar{\imath}}))$ could be represented using machines. We leave the exact formal connection between machines and our programs for future research.

4 Coherent outcomes

Suppose we are given an encounter E. How are we to identify the "reasonable" collective outcomes of E? The key concept we define is the notion of a *coherent* outcome. A coherent 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. In other words, every action a player takes is justified by the guard in his strategy labelling that choice. So coherence is *not* a notion of rationality in the sense that it does not refer to the utilities or preferences in the game.

Formally, if $\varpi = \langle \omega_1, \ldots, \omega_n \rangle \in \Omega_E$ then ϖ is said to be *coherent* if it satisfies the following condition:

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

Intuitively, a coherent outcome is an outcome where each individual action of a player does not occur without a reason: the guard on the transition leading to the action is satisfied. In other words, an encounter is coherent if every condition for doing an action along a path, is true. Given an encounter E, let $coh(E)$ denote the set of coherent outcomes of E.

The proof of the following theorem is given in the example immediately following it.

Theorem 1 1. *There exist encounters E such that $coh(E) = \emptyset$.*
2. *There exist encounters E such that $|coh(E)| = 1$.*
3. *There exist encounters E such that $|coh(E)| > 1$.*

Example (1, continued) Let us look at the one shot PD game explained earlier. Recall that a strategy for player i looks like the tree in Fig. 2 (left). Let the condition for doing c_i for each agent i be Φ_i. We will give instances of the conditions Φ_i such that the resulting encounter has 0, 1, 2, 3, or 4 coherent outcomes.

1. We first give an encounter that has no coherent outcomes. Let $\Phi_1 = (1 : \bigcirc do(d_1))$, i.e., player 1's program says that he will cooperate if and only if he will defect. Irrespective of 2's program, there is no coherent outcome. To see this, consider the two possible outcomes for 1: $\omega_1^a = (s_1^0, s_1^1)$ (i.e., 1 cooperates) and $\omega_1^b = (s_1^0, s_1^2)$ (he defects). Let ω_2 be an arbitrary outcome of player 2. Now first consider $\omega_a = \langle \omega_1^a, \omega_2 \rangle$. Note that in this case $C_1(s_1^0, s_1^1) = \bigcirc do(d_1)$, the condition for 1 to play c_1 is to play d_1. However, in the resulting encounter E^a, we have $E^a, \omega_a, 0 \models \neg\bigcirc do(d_1)$: we assume that in ω_1^a, player 1 cooperates. Similarly, if we combine ω_1^b with an arbitrary ω_2, the condition $C_1(s_1^0, s_1^2)$ becomes $\neg\bigcirc do(d_1)$, but in the encounter E^b, this condition is false in $E^b, \langle \omega_1^b, \omega_2 \rangle, 0$.
2. Now suppose $\Phi_1, \Phi_2 \in \{\top, \bot\}$. Each combination of such guards gives rise to one coherent outcome: the four combinations in total account for the table at the right-hand-side of Fig. 1. More specifically, consider $\Phi_1 = \top$ and $\Phi_2 = \bot$ then the only coherent outcome is $\langle (s_1^0, s_1^1), (s_2^0, s_2^2) \rangle$ (this corresponds to the outcome (c_1, d_2), with payoffs (0,5), in our example): similarly for the other choices for conditions from $\{\top, \bot\}$. So here, every strategy with Φ_1 taken from $\{\top, \bot\}$ gives rise to exactly one coherent outcome.
3. Suppose $\Phi_1 = \top$. If $\Phi_2 = (1 : \bigcirc do(c_1))$, we get the coherent outcome $\langle (s_1^0, s_1^1), (s_2^0, s_2^1) \rangle$ (both players cooperate: $\langle c_1, c_2 \rangle$). If $\Phi_2 = (2 : \bigcirc do(c_2))$ then there are *two* coherent outcomes, corresponding to the action profiles $\langle c_1, c_2 \rangle$ and $\langle c_1, d_2 \rangle$. Now, suppose $\Phi_1 = (2 : \bigcirc do(c_2)$ ("I cooperate if he cooperates"). If $\Phi_2 = (1 : \bigcirc do(c_1))$ then the two coherent outcomes correspond to the action profiles $\langle c_1, c_2 \rangle$ and $\langle d_1, d_2 \rangle$. Intuitively, this makes sense, and note that the reading of each strategy in fact is "I cooperate iff he cooperates": if my opponent cooperates, I will as well (this is outcome $\langle c_1, c_2 \rangle$, and if my opponent defeats,

I will as well ($\langle d_1, d_2\rangle$). One might suggest to make the condition for cooperation even "more conditional", to something like: "I will cooperate iff given that I cooperate, he cooperates as well". It is not difficult to see that if both agents adopt $\Phi_i = ((i : \bigcirc c_i) \rightarrow (\bar{\imath} : \bigcirc c_{\bar{\imath}})$, the coherent outcomes remain $\langle c_1, c_2\rangle$ and $\langle d_1, d_2\rangle$.

4. An example of an encounter where we have three coherent outcomes is one in which $\Phi_1 = (1 : \bigcirc do(c_1)) \wedge (2 : \bigcirc do(c_2))$ (1 cooperates iff both cooperate) and $\Phi_2 = \neg((1 : \bigcirc do(d_1)) \wedge (2 : \bigcirc do(d_2)))$ (2 defects iff both defect). The coherent outcomes correspond to the plays (c_1, c_2), (d_1, c_2) and (d_1, d_2).
5. Suppose $\Phi_1 = (1 : \bigcirc do(c_1))$. If $\Phi_2 = (2 : \bigcirc do(c_2))$, then *all* possible collective strategies are coherent: 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 $\neg\Phi_i$.

Notice that the example above in which $\Phi_1 = (2 : \bigcirc do(c_2)$ and $\Phi_2 = (1 : \bigcirc do(c_1))$ illustrates the straightforward encoding of Tennenholtz's program for the prisoner's dilemma in our setting. Notice that this case permits $\langle c_1, c_2\rangle$ as a coherent outcome, as announced in our introduction. As in Tennenholtz's setting, mutual cooperation is a coherent 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 $\Phi_2 = \neg(1 : \bigcirc\neg do(c_1))$, then the result would be the same: mutual cooperation remains a coherent outcome. Also notice that $\Phi_1 = (2 : \bigcirc do(c_2)$ indeed represents a *conditional commitment*: even though $\langle d_1, d_2\rangle$ is one of the two coherent outcomes if $\Phi_2 = (1 : \bigcirc do(c_1))$, under any joint strategy under which player 2 plays c_2, player 1 will play c_1.

The fact that there are encounters with no, or with more than one coherent outcome is a natural phenomenon using our programs as strategies: sometimes there is no way to coherently combine several programs (take for instance $\Phi_1 = (2 : \bigcirc do(c_2)$ and $\Phi_2 = (1 : \bigcirc do(d_1))$ in the PD game), or even one program on itself can be the source of incoherency (see the fist item in the example above). Moreover, it may happen that there are alternative ways to combine the programs into a coherent outcome. This need not be an objection to our notion of coherent outcome, in a similar way as the fact that one can have no, one, or similar Nash equilibria in classical game theory does not make that solution concept inappropriate.

Example (2, continued) Suppose prisoner 1 plays TIT- FOR- TAT. If 2 plays this as well, a coherent outcome is one in which both prisoners cooperate, all along the game. This is also the only coherent outcome: if a prisoner would defect, this is not justified by the condition (3). If prisoner 2 plays GRIM- TRIGGER, the unique coherent 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:

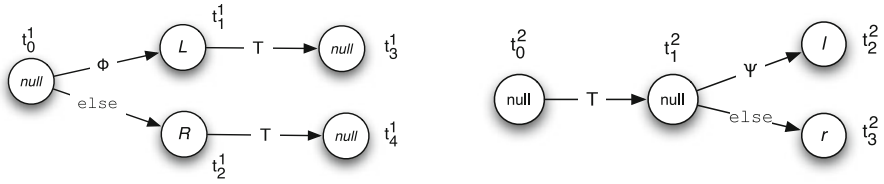
$$\neg start \vee (1 : do(c_1)) \tag{8}$$

In this case, the coherent 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 coherent outcomes are again plays in which only c_i is used. In general, when 2 uses GRIM- TRIGGER coherent 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 coherent 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 coherent 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 strategies for agents 1 and 2 again, depicted in the following figure (agent 1's strategy is depicted left, that of 2 is on the right).



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, one that will never be taken).

By choosing

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

one sees that this representation is "equivalent", in a sense we won't make precise, to the one represented earlier, in Sect. 2. By putting additional constraints on Ψ (like, it cannot refer to any $1 : do(\alpha)$), we can model games with imperfect information, where the choice of player 2 is not supposed to depend on that of player 1. We leave these notions as issues of further research (but see also Sect. 4.1).

So what are coherent outcomes in this game, represented by the figure above? Let us in this example reason about such coherent outcomes using the payoffs of the game, although the two notions are orthogonal (see Sect. 4, first paragraph). Let us first assume agent 2 only cares about his own payoff: $\Psi = (1 : do(R))$. For agent 1, let us first assume that he knows that 2 is rational: $\Phi = \bot$. The resulting encounter has one coherent outcome $\langle (t_1^0, t_1^2, t_1^4), (t_2^0, t_2^1, t_2^2) \rangle$, in which agent 1 obtains 3, and agent 2 obtains 8 (see Example 3 for the associated payoffs). Next, suppose $\Phi = \neg((1 : \bigcirc do(L)) \rightarrow (2 : \bigcirc\bigcirc do(r)))$ ("I play R iff playing L would imply that would 2 answer with playing r"). Now there are two coherent outcomes: $\langle (t_1^0, t_1^1, t_1^3), (t_2^0, t_2^1, t_2^3) \rangle$ corresponding to 1 playing L, and 2 playing r, and the second

coherent outcome being $\langle (t_1^0, t_1^2, t_1^4), (t_2^0, t_2^1, t_2^2)\rangle$, corresponding to 1 playing R and 2 playing l.

Are there conditions such that the game has three coherent outcomes? Yes, take for instance $\Psi = (1 : do(L)) \wedge (2 : \bigcirc do(l))$: player 2 only plays l if both players play "left". This leads to the coherent outcomes corresponding 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 outcomes are coherent.

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,

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

("I vote yes iff everybody does".) Then the only coherent outcomes are those in which voting happens unanimously: 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

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

("I vote yes iff everybody else does".) Certainly, the global outcome in which everybody votes yes is coherent. In fact, everybody using $All\text{-}Other_i$ again leads to the set U of coherent outcomes. U is also obtained if $\Phi_i = Some_i = \bigvee_j (j : \bigcirc do(yes))$ and $Some\text{-}Other_i = \bigvee_{j \neq i} (j : \bigcirc do(yes))$. Suppose n is odd, say $n + 1 = 2 \cdot k$, then we can define

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

("I vote yes iff a majority does"). Maybe surprisingly, the coherent 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 coherent set is U. However, if we assume the first k agents use Maj_i and the remaining k_1 agents use All_i, the coherent set consists U together with the collective outcome in which exactly the first k agents vote yes.

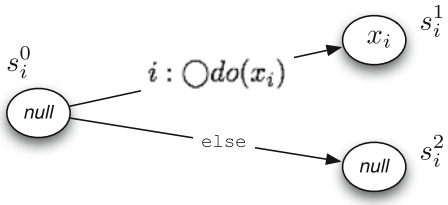
Having motivated and established our formal framework, an obvious question is the extent to which this formal framework is practical, and in particular, the extent to which, given an encounter E, we can answer questions relating to the set $coh(E)$ of coherent outcomes. Such questions are the domain of computational complexity theory (Papadimitriou 1994). A first natural question is whether, given an encounter E and an outcome ϖ, we have $\varpi \in coh(E)$, i.e., the question of checking whether

ϖ is a coherent outcome of E. In fact, this problem is easily seen to be decidable in polynomial time, through dynamic programming.

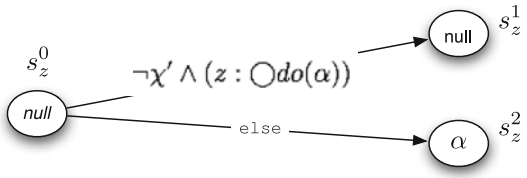
So, consider the decision problem NON- EMPTY COHERENT SET, in which we are given an encounter E, and asked whether or not $coh(E) \neq \emptyset$. In contrast to checking whether a particular outcome is coherent, this problem is computationally hard.

Theorem 2 NON- EMPTY COHERENT 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 $coh(E_\chi) \neq \emptyset$ iff the χ is satisfiable. ($\Leftarrow$) Assume $coh(E_\chi) \neq \emptyset$. Now, consider agent z. First, notice that the individual outcome (s_z^0, s_z^1) cannot be in any coherent 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 coherent 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 coherent 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 coherent. 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 $coh(E_\chi)$ 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 $|coh(E)|$ is #P-complete.*

4.1 Non-determinacy

Let us call an encounter E *deterministic* if $|coh(E)| = 1$, i.e., there is exactly one coherent outcome. An obvious question is whether we can, given the strategies (and in particular the guards that occur in them), predict whether an encounter is deterministic or not. The examples in the previous section suggest that an encounter allows for more than one coherent outcome, or lack any such outcome, if the actions of some player i depend somehow on his future actions, or if they depend on other agents' actions, which, in turn, depend on the choices of player i. We will now formalise the existence of such cycles and show that, if such cycles do not exist, determinism of E is guaranteed.

We define some auxiliary notions. Take an encounter $E = \langle Ag, Ac_1, \ldots, Ac_n, \sigma_1, \ldots, \sigma_n \rangle$. Given a state s in a strategy σ_i, let the time $\tau(s)$ of a state be the distance to the root s_i^0. If $\tau(s) > 0$, let $\Phi(s)$ be the guard on the edge (s', s). A *decision point* is an agent-time pair (i, t). We now want to formalise a dependency relation D_E on decision points, where the intuitive meaning of $D_E(i', t')(i, t)$ is that "the decision of i at time t depends on the decision of i' at time t'", or, "i' needs to decide at t' before i can decide at t". We will, for a given point (i, t), collect such dependencies in a function Δ_E, where $\Delta_E(i, t) = \{(i', t') \mid D_E(i', t')(i, t)\}$ collects the decision points that i depends on, at time t. To start with, define $\Delta_E(i, 0) = \{\}$: at time $t = 0$, we assume all agents take the action *null* and this does not depend on anything else. For $t > 0$, define $\Delta_E(i, t) = \cup_{\{s \in \sigma_i \mid \tau(s) = t\}} \delta_E(i, s, \Phi(s))$, where

$$\delta_E(i, s, \Phi(s)) = \begin{cases} \delta_E(i, s, \Psi) & \text{if } \Phi(s) = \neg\Psi \\ \delta_E(i, s, \Psi_1) \cup \delta_E(i, s, \Psi_i) & \text{if } \Phi(s) = \Phi_1 \vee \Phi_2 \\ d_E(j, \{\tau(s) - 1\}, \varphi) & \text{if } \Phi(s) = (j : \varphi) \\ \{\} & \text{if } \Phi(s) = \top \end{cases}$$

The function $d_E(i, T, \varphi)$ takes an agent i, a set of time points T and an individual outcome formula φ. Before defining it, consider the following encounter based on the two-shot PD denoted in Fig. 2 (right). Suppose $\Phi_1^1 = \top$ and $\Phi_2^1 = (1 : \Diamond c_1) \wedge (2 : do(null))$, i.e., 1 plays c_1 and 2 conditions his choice for c_2 on 1 ever going to play c_1 and 2 having played *null*. We then get $\Delta_E(1, 1) = \delta_E(1, s_1^1, \Phi_1^1) \cup \delta_E(1, s_1^2, \neg\Phi_1^1) = \{\}$. Moreover, $\Delta_E(2, 1) = \delta_E(2, s_2^1, \Phi_2^1) \cup \delta_E(2, s_2^2, \neg\Phi_2^1) = d_E(1, \{0\}, \Diamond do(c_1)) \cup$

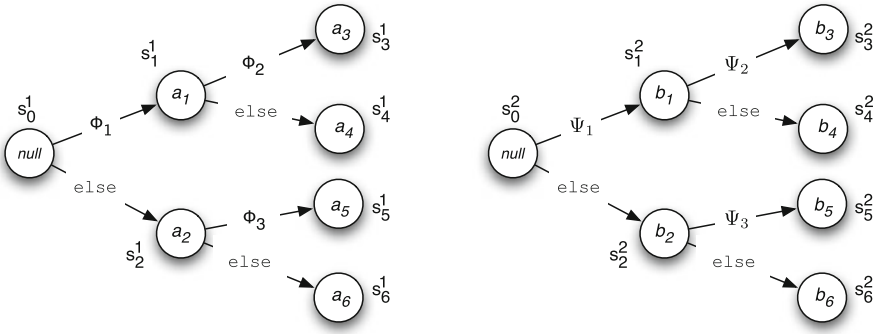


Fig. 5 Two encounters

$d_E(2, \{0\}, do(null))$. One would expect $d_E(2, \{0\}, do(null))$ to be $\{(2, 0)\}$: whether 2 will chose *null* at $t = 0$ depends on what 2 choses at $t = 0$. Moreover, whether 1 makes $\Diamond do(c_1)$ true at $t = 0$ depends on what 1 choses at $t = 0, t = 1$ and $t = 2$, which in turn may depend on $\Delta_E(i, t)(t \leq 2)$, i.e., the decision points that 1 depends on, at those time points. I.e., we will find that $d_E(1, \{0\}, \Diamond do(c_1)) = \{(1, 0)\}, (1, 1), (1, 2)\} \cup_{0 \leq t \leq 2} \Delta_E(1, t)$. Formally, d_E is defined as follows:

$$d_E(i, T, \varphi) = \begin{cases} \{(i, t) \mid t \in T\} \cup_{t \in T} \Delta_E(i, t) & \text{if } \varphi = do(\alpha) \\ d_E(i, T, \psi) & \text{if } \varphi = \neg\psi \\ d_E(i, T, \psi_1) \cup d_E(i, T, \psi_2) & \text{if } \varphi = \psi_1 \vee \psi_2 \\ d_E(i, \{t - 1 \mid t \in T\}, \psi) & \text{if } \varphi = \bullet\psi \ \& \ t > 0 \\ d_E(i, \{t + 1 \mid t \in T\}, \psi) & \text{if } \varphi = \bigcirc\psi \ \& \ t <\mid \sigma \mid \\ d_E(i, \{t' \mid t \leq t' \leq \mid \sigma \mid\}, \psi) & \text{if } \varphi = \Diamond\psi \\ d_E(i, \{t' \mid 0 \leq t' \leq t\}, \psi) & \text{if } \varphi = \blacklozenge\psi \end{cases}$$

Given a strategy profile σ, call i's strategy *a-cyclic* if for all states $s \in \sigma_i$ with $\tau(s) = t$, there is no pair $(i, t') \in \delta(\sigma, i, s)$ for which $t' > t$.

Example 5 Consider the encounters depicted in Fig. 5. Suppose $\Phi_1 = \top$, $\Phi_2 = (2 : do(b_1))$ and $\Phi_3 = (2 : do(b_2))$.

1. For our first encounter E, put $\Psi_1 = \bot$, $\Psi_2 = (1 : do(a_1))$ and $\Psi_3 = (1 : do(a_2))$. We get $\delta_E(1, s_1^1, \Phi_1) = \delta_E(1, s_2^1, \neg\Phi_1) = \{\}$: what 1 does at s_1^1 and s_2^1 does not depend on any agents' choices. Hence $\Delta_E(1, 1) = \{\}$, and likewise $\Delta_E(2, 1) = \{\}$. Next, we compute $\Delta_E(1, 2)$. $\delta_E(1, s_3^1, \Phi_2) = \delta_E(1, s_4^1, \neg\Phi_2) = d_E(2, \{1\}, do(b_1)) = \{(2, 1)\} \cup \Delta_E(2, 1) = \{(2, 1)\} \cup \delta_E(2, s_1^2, \Psi_1) \cup \delta_E(2, s_2^2, \neg\Psi_1)$. The guard Ψ_1 for s_1^2 is $\bot$ so that the latter union reduces to $\{(2, 1)\}$. In words: what 1 does at state s_3^1 or s_4^1, depends only on what 2 does at $t = 1$. Likewise, $\delta_E(1, s_5^1, \Phi_3) = \delta_E(1, s_6^1, \neg\Phi_3) = \{(2, 1)\}$ so that $\Delta_E(1, 2) = \{(2, 1)\}$. The complete dependency graph for this encounter is given in Fig. 6 (left).
2. Now assume E is obtained by setting $\Psi_1 = \bot$, $\Psi_2 = (1 : \bigcirc(do(a_3) \vee do(a_5))$ and $\Psi_3 = (1 : \bigcirc do(a_2))$.
   Again, we have $\Delta_E(1, 1) = \Delta_E(2, 1) = \{\}$ and $\Delta_E(1, 2) = \{(2, 1)\}$. We

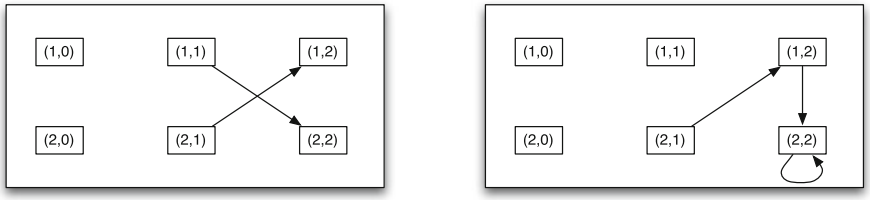


Fig. 6 Dependency graphs for Example 5

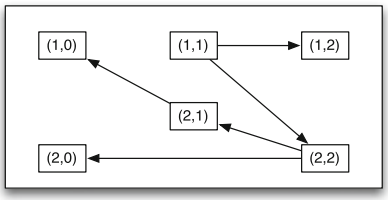


Fig. 7 Dependency graph for Example 6

finally compute $\Delta_E(2,2) = \delta_E(2, s_3^2, \Psi_2) \cup \delta_E(2, s_4^2, \neg\Psi_2) \cup \delta_E(2, s_5^2, \Psi_3) \cup \delta_E(2, s_6^2, \neg\Psi_3)$. We have $\delta_E(2, s_3^2, \Psi_2) = \delta_E(2, s_4^2, \neg\Psi_2) = d_E(1, \{1\}, \bigcirc(do(a_3 \vee do(a_5))) = \{(2,2)\} \cup \Delta_E(1,2) = \{(2,2),(1,2)\}$. We also get $\delta_E(2, s_5^2, \Psi_3) = \delta_E(2, s_6^2, \neg\Psi_3) = \{(2,2),(1,2)\}$. The dependency graph for this encounter is given in Fig. 6 (right).

Example 6 Consider the encounter E based on Fig. 5 for which $\Phi_1 = (1 : \bigcirc\bigcirc do(a_3) \wedge 2 : \bigcirc\bigcirc do(b_3)$, $\Phi_2 = \top$ and $\Psi_1 = (1 : do(null)$, $\Psi_2 = (2 : \blacklozenge do(b_2))$ and $\Psi_3 = \top$. We leave it to the reader to verify that this gives rise to the dependency graph of Fig. 7.

Proposition 1 *Let E be an encounter with an acyclic dependency graph. Then: $|coh(E)| = 1$.*

Proof The argument is standard when having a Directed Acyclic Graph (DAG) for dependencies. Start with the decision nodes (i', t') without incoming arrows: such nodes exist in any finite DAG. The guards in the strategies that correspond to these nodes (that is, the states $s_{i'}$ in strategy $\omega_{i'}$ for which $\tau(s_{i'} = t')$ do not depend on any other decision points, so they can be evaluated as either true or false, and hence the actions corresponding to those states can be determined. Next, visit the decision points (i, t) for which $D_E(i', t')(i, t)$: this time, the actions in states s_i of strategies ω_i with $\tau(s_i) = t$ can be determined. This process will terminate and visit all decision points, and it generates a unique outcome that is coherent. □

The following consequence confirms one's intuition that if players only choose unconditionally (the guard is $\top$ or $\bot$) or respond to what happened in the past, the encounter is deterministic.

Corollary 2 *Let E be an encounter in which all guards use only $\top$, $\bot$ and formulas in which every occurrence of $do(\alpha)$ is in the scope of past time operators $\bullet$ and $\blacklozenge$. Then $|coh(E)| = 1$.*

One can use our dependency graph also to define notions of imperfect information: for instance, a graph without arrows from any decision point (i, t) to any point (j, t') would mean that agent j cannot refer to any of agent i's actions. We leave a further exploration of such issues for future work.

5 Logic programming semantics

Intuitively, a coherent outcome is an outcome where each individual action of a player does not occur without a reason: the guard on the transition leading to the action is always satisfied. Hence, we could say that the satisfaction of such a guard Φ provides an argument for choosing this action $do(a)$. Then it is only one step further to consider the combination of a guard leading to an action as a rule $\Phi \rightarrow do(a)$ whose antecedent has to be fulfilled in order for the consequent to occur. Thus, the coherent outcome semantics can be considered as providing an intuitive semantics of an encounter conceived as such a set of rules.

In the computer science community, *logic programming* (Apt 1990; Lloyd 1987) has provided us with a rich variety of semantics capturing various intuitive interpretations of such rule systems. Nowadays, the so-called *answer set semantics* (Ferraris and Lifschitz 2005) is considered as the dominant semantics in logic programming capturing the intuitive meaning of a logic program. This immediately raises the question how the various logic programming semantics, and the answer set semantics in particular, are related to our coherent outcome semantics.

Thus, in order to place the coherent outcome semantics in a broader perspective, first we will relate encounters as defined earlier with logic programs. Next, we use the ideas underlying the coherent outcome semantics for encounters to identify a semantics for logic programs derived from encounters. Then we investigate whether this "coherent model" semantics can be used to provide an intuitively acceptable semantics for these logic programs, taking into account the supported model semantics, the minimal model semantics and the answer set semantics.

5.1 From encounters to logic programs

We will use predicates $state(i, X, t)$ to indicate a possible state for an agent i at time t, where X is a variable indicating the state history. Assume a fixed enumeration $< S_i >$ of states in S_i. The initial state of i is denoted by $state(i, <>, 0)$, and, proceeding inductively, whenever for a state s at time t, denoted by $state(i, X, t)$, we have $(s, s'), (s, s'') \in R^i$, where s' occurs before s'' in $< S_i >$, then s' is denoted by $state(i, X.0, t + 1)$ and s'' by $state(i, X.1, t + 1)$. As an example, take the encounter on the left of Fig. 5. Let the enumeration of the states simply be $s_1^0, s_1^1, s_1^2, s_1^3, s_1^4, s_1^5, s_1^6$. The predicate $state(1, \langle 1.0 \rangle, 2)$ represents the state s_1^5 in that encounter: it is at depth 2, and the history 1.0 indicates that at time 0, the state which is higher in the enumeration was chosen (i.e., s_1^2, rather than s_1^1, and at time 1, the history indicates that the state lower down the enumeration was chosen (i.e., s_1^5, rather than s_1^6). Likewise, $state(1, 0.0, 2)$ represents the state s_1^3 (the history indicates that twice the state that occurs earlier in the enumeration was chosen).

Given an individual outcome $\omega_i = \langle s_i^0, \ldots, s_i^{k-1} \rangle$ we use state predicates and atomic predicates $does(i, a, t)$ to associate a k-run $\mathbf{r}_i = \mathbf{r}_i(\omega_i)$ with it, where

$$\mathbf{r}_i = \{state(i, <>, 0), does(i, a_0, 0), state(i, X_1, 1), does(i, a_1, 1), \ldots, state(i, X_{k-1}, k-1), does(i, a_{k-1}, k-1)\}$$

such that for $0 \leq t < k$, $state(i, X_t, t)$ denotes s_i^t and $a_t = L_i(s_i^t)$, i.e., $a_t \in Ac_i$ is the action that agent i takes at time t in ω_i.

The set of atoms for our logic programs for encounters of depth k is the set $At_k^E = \{state(i, X, t) \mid i \in Ag, X \in \{0, 1\}^t, t < k\} \cup \{does(i, a, t) \mid i \in Ag, a \in Ac_i, 0 \leq t < k\}$. An interpretation for a logic program is a subset I of At_k^E. If $\omega = \langle \omega_1, \ldots, \omega_n \rangle$ is a collective outcome of an encounter E, then we say that the interpretation I_ω associated with ω is $I_\omega = \mathbf{r}_1(\omega_1) \cup \cdots \cup \mathbf{r}_n(\omega_n)$.

Let $\mathcal{L} = \mathcal{L}(Ag_k^E)$ be the language built from atoms in At_k^E using Boolean connectives. Given a strategy profile σ, and $u \in \mathbb{N}$, define a translation t on guards from $\mathcal{L}_\mathcal{T}$ such that $t(u, \Phi) \in \mathcal{L}$ as follows. If Φ is a guard occurring at depth u in a strategy for some agent, its translation will be denoted by $t(u, \Phi)$. First of all, $t(u, \Phi_1 \wedge \Phi_2) = t(u, \Phi_1) \wedge t(u, \Phi_2)$, while $t(u, \neg\Phi) = \neg t(u, ßΦ)$. Also $t(u, \top) = \top$ and $t(u, \bot) = \bot$. Finally, $t(u, (i : \varphi)) = T(u, i, \varphi)$, where (the translation for the past time operators is similar):

$$\begin{aligned} T(u, i, do(\alpha)) &= does(i, \alpha, u) \\ T(u, i, \bigcirc\varphi) &= T(u+1, i, \varphi) \\ T(u, i, \Diamond\varphi) &= T(u, i, \varphi) \vee T(u+1, i, \varphi) \vee \cdots \vee T(k-1, i, \varphi) \end{aligned}$$

Example 7 Consider the 2-round Prisoner's Dilemma from Fig. 2 (right). Suppose we have $\Phi_1 = (2 : \Diamond(do : c_2))$, $\Phi_2 = \neg((1 : do(c_1)) \wedge (2 : do(d_2)))$. Then the formula φ_1 associated with Φ_1 is $does(2, c_2, 0) \vee does(2, c_2, 1) \vee does(2, c_2, 2)$. The formula φ_2 is $\neg(does(1, c_1, 1) \wedge does(2, d_2, 1))$.

Given an encounter E of depth k, and a strategy σ_i in E, a logic program $P_{\sigma_i}^E$ for agent i is a set of rules, such that

1. for $t = 0$ the program contains exactly one rule

$$\top \rightarrow state(i, <>, 0),$$

2. for every $0 < t < k$ there are exactly two rules of the form[5]

$$state(i, X, t), t(t, \varphi) \rightarrow state(i, X.0, t+1), does(i, a_i, t)$$
$$state(i, X, t), \neg t(t, \varphi) \rightarrow state(i, X.1, t+1), does(i, b_i, t)$$

[5] Strictly speaking, in an extended logical program we only allow a conjunction of literals to occur in the body of the rules. Therefore, for each φ, if φ is $\psi_1 \vee \cdots \vee \psi_k$ in disjunctive normal form, then each rule $\varphi \rightarrow p$ should be interpreted as a set of rules $\{\psi_j \rightarrow p \mid 1 \leq j \leq k)\}$. Moreover, usually, in such a program we only allow one literal to occur in the head of a rule. To simplify notation, however, here we also allow a conjunction of literals $p_1, p_2, \ldots, p_k$ to occur in the head of a rule. Each such a rule $\varphi \rightarrow p_1, p_2, \ldots, p_k$ then should be interpreted as a set of rules $\{\varphi \rightarrow p_i \mid 1 \leq j \leq k)\}$.

whenever

(a) there is a state s, denoted by $state(i, X, t)$, occurring at depth t and there are states s' and s'', denoted by $state(i, X.0, t+1)$ and $state(i, X.1, t+1)$, respectively, occurring at depth $t+1$;
(b) $\{(s, s'), (s, s'')\} \subseteq R_i$, $C_i(s, s') = \varphi$, and $C_i(s, s'') = \neg\varphi$;
(c) $L_i(s') = a_i$ and $L_i(s'') = b_i$.

If $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ is the strategy profile for the encounter E and $P^E_{\sigma_i}$ is the logic program associated with σ_i, then $P^E = P^E_{\sigma_1} \cup \cdots \cup P^E_{\sigma_n}$ is called the logic program associated with the encounter E.

Example 8 Let E denote the k-round Prisoner's dilemma, and consider the program $P^E = P^E_{\sigma_1} \cup P^E_{\sigma_2}$ where σ_i is the strategy TIT FOR TAT for player i with cooperative first move:

$$\begin{aligned}
\top &\rightarrow state(1, <>, 0)\\
state(1, <>, 0), \top &\rightarrow state(1, <1>, t), does(1, c_1, 1)\\
state(1, X, t), does(2, c_2, t) &\rightarrow state(1, X.0, t+1), does(1, c_1, t+1)\\
state(1, X, t), \neg does(2, c_2, t) &\rightarrow state(1, X.1, t+1), does(1, d_1, t+1)\\
\top &\rightarrow state(2, <>, 0)\\
state(2, <>, 0), &\rightarrow state(2, <1>, t), does(2, c_2, 1)\\
state(2, X, t), does(1, c_1, t) &\rightarrow state(2, X.0, t+1), does(2, c_2, t+1)\\
state(2, X, t), \neg does(1, c_1, t) &\rightarrow state(2, X.1, t), does(2, d_2, t+1)
\end{aligned}$$

Here, t is a variable taking values $t = 1, 2, \ldots, k-2$.

We say that an interpretation $I \subseteq At^G_k$ satisfies a formula φ over At^E_k, denoted by $I \models \varphi$, if the truth assignment τ_I defined by

$$\tau_I(p) = \begin{cases} 1 & \text{if } p \in I,\\ 0 & \text{else} \end{cases} \tag{9}$$

satisfies φ. Note that such an assignment is well-defined since I is a subset of *positive literals* over At^G_k.

Our first result is an easy consequence of the preceding discussion relating outcomes ω of an encounter E to interpretations I_ω for P^E:

Lemma 1 *Let E be an encounter of depth k, ω be an outcome in E and $I_\omega = \mathbf{r}_1(\omega_1) \cup \cdots \cup \mathbf{r}_n(\omega_n)$. Then, for $0 \leq u < k$ and $\Phi \in \mathcal{L}_I$, it holds that $E, \omega, u \models_{\mathcal{C}} \Phi$ iff $I_\omega \models t(u, \Phi)$.*

Since we are not primarily interested in outcomes in general, but in *coherent* outcomes of an encounter, we now investigate which requirements should hold for interpretations I_ω of a logic program P^E in order to correspond to coherent outcomes ω.

5.2 From coherent outcomes of encounters to coherent models of programs

Since a logic program P is a set of rules, in order to be acceptable, an interpretation I for P should respect the program rules. That is, whenever an interpretation I satisfies the condition part (antecedent) φ of a rule $\varphi \rightarrow p_1, \ldots, p_k$, I should satisfy the all consequents p_i as well. More precisely, let P be a logic program over a set of atoms[6] At. A *model* M of a logic program P is an interpretation $M \subseteq At$ such that M satisfies all the rules in P, i.e. for every rule $\varphi \rightarrow p_1, \ldots, p_k$ of P, if $M \models \varphi$ then for all $i = 1, \ldots, k$, $p_i \in M$.

The following proposition is an easy consequence of the construction of P^E:

Proposition 2 *Let E be an encounter of depth k and ω an outcome in E. Then I_ω is a model of P^E.*

It is easy to see that not every model M of a program P^E corresponds to an outcome of the corresponding encounter E. For example, take the tit-for-tat program discussed in Example 8. A possible model for this program is the model

$$M = \{state(i, <>, 0) \mid i = 1, 2\} \cup \\ \{state(i, X_t, t), does(i, c_i, t), does(i, d_i, t) : i = 1, 2,\ t \geq 1, X_t \in \{0, 1\}^t\}$$

Clearly, such a model does not satisfy the requirements we have for coherent outcomes of the corresponding encounter E: for every i and t there should occur a unique atom $does(i, a, t)$ and a unique atom $state(i, X, t)$ in M and both should be supported by a rule

$$state(i, X, t-1), \quad \varphi \rightarrow state(i, X', t), \quad does(i, a, t)$$

occurring in P^E. Therefore, arbitrary models of logic programs do not capture the idea of a (coherent) outcome.

Obviously, the reason for this lack of correspondence is that we did not include the second requirement for coherency: no consequent of a rule should be considered as true without being supported by some rule. Therefore, let us call an interpretation I of a program P^E derived from an encounter E *coherent* if the following conditions do hold:

1. I is a model of P^E;
2. if $c \in I$ then there is a reason for it, i.e. there exists a rule $\psi \rightarrow p_1, \ldots, p_k$ in P^E such that $c = p_j$ for some $j = 1, \ldots, k$ and $I \models \psi$.

We call such an interpretation a *coherent model* of P^E. To show that these intuitions behind coherency are justified, we now prove that there exists an exact correspondence between the set of coherent outcomes of an encounter E and the set of coherent models of the associated logic program P^E:

Theorem 3 *Let E be an encounter, and P^E its associated logic program. Then the following holds:*

[6] Such a set is known to be the Herbrand base of the program.

1. *if ω is a coherent outcome of E then I_ω is a coherent model of P^E.*
2. *if M is a coherent model of P^E, then there exists a coherent outcome ω such that $M = M_\omega$.*

Proof (1) Let ω be a coherent outcome for the strategy profile σ. We show that I_ω is a coherent model of P^E. First of all, since ω is an outcome, by Proposition 2, I_ω is a model of P^E. To show that I_ω is coherent, consider an arbitrary $i \in Ag$ and $0 < t < k$ and let $\omega_i[t-1] = \sigma_i^{t-1}$ and $\omega_i[t] = \sigma_i^t$. Let $L_i(\sigma_i^t) = a$ and $C_i(\sigma_i^{t-1}, \sigma_i^t) = \Phi$. Then, by construction of I_ω, there exist atoms $state(i, X_{t-1}, t-1)$ and $does(i, a, t), state(i, X_t, t) \in I_\omega$ and, since $E, \omega, t-1 \models \Phi$, by Lemma 1, we have $I_\omega \models t(t-1, \Phi)$. Hence, by construction of P^E, there exists a rule $state(i, X_{t-1}, t-1), t(t-1, \Phi) \rightarrow state(i, X_t, t), does(i, a, t)$ where $state(i, X_t, t), does(i, a, t) \in I_\omega$ and $I_\omega \models state(i, X_{t-1}, t-1) \wedge t(t-1, \Phi)$. Hence, I_ω is a coherent model of P^E.

(2) Assume that M is a coherent model of P^E. First, we will show that there exists an outcome ω of E such that $M = M_\omega$.

In order to represent an outcome ω, it should hold that, for every i, the set M_i satisfies

$$\begin{aligned} M_i &= \{state(i, X_m, m), does(i, a, m) \mid state(i, X_m, m), does(i, a, m) \in M\} \\ &= \{state(i, <>, 0), does(i, a_0, 0), state(i, X_1, 1), does(i, a_1, 1), \ldots, \\ &\quad state(i, X_{k-1}, k-1), does(i, a_{k-1}, k-1)\} \end{aligned}$$

That is, for every i, M_i represents a run $r_i(\omega_i)$ i.e., for every $1 \leq m < k$, $does(i, a_m, m)$, $does(i, a'_m, m) \in M_i$ implies $a_m = a'_m$ and $state(i, X_m, m), state(i, X'_m, m) \in M_i$ implies $X_m = X'_m$.

We prove by contradiction that for every i this property holds. So, assume that there exists an i such that M_i does not represent a k-run ω_i. Then (i) there exists a smallest $1 \leq m < k$ such that for some $a \neq b$, $does(i, a, m)$ and $does(i, b, m)$ occur in M_i or for some $X_m \neq Y_m$, $state(i, X_m, m)$ and $state(i, Y_m, m)$ occur in M_i or (ii) there exists a smallest $1 \leq m < k$ such that for no a, $does(i, a, m)$ occurs in M or for no $X \in \{0, 1\}^m$, $state(i, X, m)$ occurs in M.

Suppose (i) holds, and without loss of generality we assume that $state(i, X, m)$ and $state(i, Y, m)$ occur in M. Since M is a coherent model of P^E, it follows that $state(i, X, m), does(i, a, m)$ and $state(i, Y, m), does(i, a, m)$ are the heads of the unique rules $state(i, X', m-1), \varphi \rightarrow state(i, X, m), does(i, a, m)$ and $state(i, X'', m-1), \psi \rightarrow state(i, Y, m), does(i, a, m)$, respectively, in P^E such that $M \models \varphi$ and $M \models \psi$. But then, by construction of P^E, we must have $X' = X''$ and $\psi \equiv \neg\varphi$, implying that $M \models \varphi$ and $M \models \neg\varphi$; contradiction.

Therefore, (ii) must hold. Again, without loss of generality assume that for no a, $does(i, a, m)$ occurs in M. Then, by construction of P^E, for some $state(i, X, m-1) \in M$ and $a, b \in Ac_i$, there are two rules $state(i, X, m-1), \varphi \rightarrow state(i, X.0, m), does(i, a, m)$ and $state(i, X, m-1), \neg\varphi \rightarrow state(i, X.1, m), does(i, b, m)$ in P^E. Since M is a model, $M \models state(i, X, m-1)$ and either $M \models \varphi$ or $M \models \neg\varphi$, it follows that either $does(i, a, t)$ or $does(i, b, t)$ occurs in M; contradiction.

So neither (i) nor (ii) can hold and therefore no such an i can exist, implying that for every i, M_i represents a k-run ω_i. Hence, there exists an outcome ω of E such that $M = M_\omega$.

Finally, we have to show that the outcome ω is a coherent outcome. So suppose ω_i occurs in ω. Since $M = M_\omega$, $M_i = r_i(\omega_i)$ and for every $m < k$ there exists a unique pair of atoms $state(i, X_m, m), does(i, a, m) \in M$. By construction of P^E,

1. there exists a guard Φ in the strategy σ_i as a label for some (s_i^{m-1}, s_i^m) where s_i^m is of depth m and $L_i(s_i^m) = a$,
2. $state(i, X_{m-1}, m-1)$ represents σ_i^{m-1}, and $state(i, X_m, m)$ represents σ_i^m,
3. $state(i, X_{m-1}, m-1), \varphi \rightarrow state(i, X_m, m), does(i, a, m)$ occurs in P^E, and
4. $\varphi = t(m, \Phi)$.

We have to show that $E, \omega, m \models_{\mathcal{C}} \Phi$. Note that $state(i, X_{m-1}, m-1), \varphi \rightarrow state(i, X_m, m), does(i, a, m)$ is a unique rule in P^E supporting $state(i, X_m, m)$, $does(i, a, m)$. Hence, since M contains $state(i, X_m, m)$ as well as $does(i, a, m)$ and M is coherent, it follows that $M \models \varphi$. Hence, by Lemma 1, $E, \omega, m \models_{\mathcal{C}} \Phi$. Therefore, ω is a coherent outcome. □

Example 9 Consider the tit-for-tat program we discussed before in Example 8.

This program has a unique coherent model $\{state(i, <>, 0), state(i, 1.0^{t-1}, t), does(i, c_i, t) : i = 1, 2, \ t \geq 1\}$.

Example 10 Consider now two programs based on the "I will cooperate if he will cooperate" principle:

$$\top \rightarrow state(1, <>, 0) \tag{10}$$
$$state(1, X, t-1), does(2, c_2, t) \rightarrow state(1, X.0, t), does(1, c_1, t) \tag{11}$$
$$state(1, X, t-1), \neg does(2, c_2, t) \rightarrow state(1, X.1, t), does(1, d_1, t) \tag{12}$$
$$\top \rightarrow state(2, <>, 0) \tag{13}$$
$$state(2, X, t-1), does(1, c_1, t) \rightarrow state(2, X.0, t), does(2, c_2, t) \tag{14}$$
$$state(2, X, t-1), \neg does(1, c_1, t) \rightarrow state(2, X.1, t), does(2, d_2, t) \tag{15}$$

Here, $1 \leq t \leq k-1$. This program has 2^{k-1} supported models: For every $1 \leq t \leq k-1$, either both $does(1, c_1, t)$ and $does(2, c_2, t)$ or both $does(1, d_1, t)$ and $does(2, d_2, t)$ are contained in a coherent model M.

5.3 Properties of coherent models for programs

A common opinion in logic programming semantics (see e.g. Gelfond 2008) is that any acceptable model of a program P should (i) respect the rules of the program and (ii) should not consider anything true if it is not necessary to do so.

While (i) clearly implies that any acceptable model should be a model of the program, (ii) implies (among others) that an acceptable model should be a *minimal* model of P. That is, M is an acceptable model if there does not exist a model M' of P such that the set of positive literals occurring in M' is strictly contained in M.

This might imply that coherent models of P do not always constitute acceptable models of a program: Coherent models as we have defined them are known in the logic programming community as *supported models* (Fitting 2002) and, in general, supported models do not need to be minimal. For example, take the program $P = \{a \rightarrow b\}$. This program has two coherent models: $M_1 = \emptyset$ and $M_2 = \{a, b\}$. Clearly, M_2 is not minimal.

Although this property (supported models are not always minimal models) is true w.r.t. the class of all possible logic programs, we will show that, due to the special form of the programs P^E derived from encounters E, coherent models of programs P^E are always minimal models.

Lemma 2 *Let P^E be a program derived from an encounter E. Then every coherent model M of P^E is a minimal*[7] *model of P^E.*

Proof On the contrary, let us assume that M is a coherent model of P^E and M is non-minimal. Then there exists a model M' of P^E such that M' is strictly included in M. Hence, $M - M' \neq \emptyset$. Therefore, there exists a smallest[8] $t > 0$ and an i, such that $does(i, a, t) \notin M'$. The line of reasoning for $state(i, X, t)$ is completely analogous.

Since $does(i, x, t)$ occurs in M and M is coherent, there are rules[9] $state(i, X, t - 1), \varphi \rightarrow does(i, x, t)$ and $state(i, X, t - 1), \neg\varphi \rightarrow does(i, y, t)$ in P^E and $M \models \varphi$.

Since M' is a model, $state(i, X, t - 1) \in M'$ and $does(i, x, t) \notin M'$, $M' \not\models \varphi$. Hence, $M' \models \neg\varphi$. But that implies that $does(i, y, t) \in M' \subseteq M$ and therefore both $does(i, y, t)$ as well as $does(i, x, t)$ occur in M. But then $M \models \varphi$ and $M \models \neg\varphi$; hence M cannot be a model of P^E; contradiction[10]. Therefore, M must be a minimal model. □

Hence, every coherent outcome ω of an encounter E corresponds to a minimal and supported model M_ω of P^E.

5.4 Answer set semantics and coherent models

The answer set semantics of a logic program P can be considered as the most rigorous interpretation of the two rules underlying the notion of an acceptable interpretation of a program P: any such an interpretation should (i) respect the rules of the program and (ii) should not consider anything true if it is not necessary to do so. Similarly, $\neg\varphi$ is true in J iff φ is false in J, $\neg\varphi$ is false in J iff φ is true in J and $\neg\varphi$ is unknown in J iff φ is unknown in J. For disjunctions $\varphi = \varphi_1 \vee \varphi_2$ it holds that φ is true in J if at least one of φ_1 or φ_2 is true in J, φ is false in J if both φ_i are false in J, and φ is undefined, else. Finally, a partial interpretation J is said to satisfy a program rule $\varphi \rightarrow p$ if p is true in J or φ is not true in J.

[7] Models that are both supported and minimal also have been called positivistic models (Bidoit and Hull 1989).

[8] Note that for all i, $state(i, <>, 0)$ occurs in every model of P^E.

[9] Note that in the following φ might be equal to $\neg\psi$.

[10] Remember that every interpretation is a subset of positive literals.

Now a partial interpretation J is said to be an *answer set* of a logic program[11] P if J is an inclusion minimal interpretation satisfying the rules of P.

Example 11 Let's consider Example 10 once again. According to the answer set semantics, there is a unique answer set J for this program: $J = \{state(i, <>, 0) \mid i = 1, 2\}$, i.e. no positive literal over the atoms of the program is considered to be true, except the atoms indicating the initial states. The reader should be able to verify that indeed according to the definitions stated above, J satisfies every rule of this program and that J is minimal. As a reason it is argued that no rational person adhering to the principles for an acceptable interpretation of this program should have a necessary reason for taking $does(i, c_1, t)$ and $does(i, c_2, t)$ to be true or have a necessary reason for taking $\neg does(i, c_1, t)$ and $\neg does(i, c_2, t)$ to be true. Hence, all these literals are undefined. Therefore, no consequent can be assumed to be true nor false and hence, every literal is undefined, except for $state(i, <>, 0)$.

Clearly, the answer set interpretation does not meet our intuitive interpretation of the rules in the program rules derived from encounters. How can this mismatch be explained? The answer is that since partial interpretations are also taken into account, one essential assumption from encounters is lost: In encounters we assume that for every player i and for every $0 \leq t < k$, it holds that exactly one action a is chosen by i at time t. This assumption has to be explicitly encoded in a rule when dealing with the answer set semantics. Hence, we have to make sure that every (acceptable) interpretation of I, for every i and every t, satisfies exactly one element from the set $\{does(i, a, t) \mid a \in Ac_i\}$.

Such interpretations can be enforced when dealing with partial interpretations, if we add, for every player i, and for every $0 < t < k$, an exclusive disjunctive rule

$$\top \rightarrow does(i, a_1, t) \oplus does(i, a_2, t) \oplus \cdots \oplus does(i, a_n, t)$$

where $Ac_i = \{a_1, a_2, \ldots, a_n\}$. Let us call the resulting programs containing these exclusive disjunctions *extended programs*, denoted by P^{E+}.

Now it can be easily shown that answer sets for such *extended programs* P^{E+} one-to-one correspond to coherent models of P^{E+}:

Theorem 4 *Let P^{E+} be an extended program for an encounter E of depth k. Then ω is a coherent outcome of E iff I_ω is an answer set of P^{E+}.*

Proof By Theorem 3, ω is a coherent outcome of E iff I_ω is a coherent model of P^E. So we only need to prove that I_ω is a coherent model of P^E iff I_ω is an answer set of P^{E+}.

($\Rightarrow$). Let I_ω be a coherent model of P^E. Suppose that for some $1 \leq i \leq n$ there exists a smallest $1 \leq t \leq k - 1$ such that for no $a \in Ac_i$, $does(i, a, t)$ occurs in I_ω or for no $X \in \{0, 1\}^t$, $state(i, X, t)$ occurs in I_ω. By assumption,

[11] Strictly speaking, we present here the definition of an answer set for programs without default negation.

$state(i, X_{t-1}, t-1) \in I_\omega$ and, by construction of P^E, there are $a, b \in Ac_i$ such that $state(i, X_{t-1}, t-1), \varphi \rightarrow state(i, X_{t-1}.0, t), does(i, a, t)$ and $state(i, X_{t-1}, t-1), \neg\varphi \rightarrow state(i, X_{t-1}.1, t), does(i, b, t)$ occur in P^E. Since I_ω is a model, we must have $I_\omega \not\models \varphi$ and $I_\omega \not\models \neg\varphi$, contradicting the fact that I_ω is a complete interpretation. Hence, for every $1 \leq t \leq k-1$ and $1 \leq i \leq n$ there is at least one $a \in Ac_i$ such that $does(i, a, t)$ and at least one $X \in \{0, 1\}^t$ such that $state(i, X, t)$ occurs in I_ω. Since I_ω is coherent, it follows immediately that there is exactly one such an $a \in At_k^G$ such that $does(i, a, t)$ occurs in I_ω and the same holds for $state(i, X, t)$. Hence, I_ω satisfies the exclusive disjunction rules in P^{E^+}, so I_ω is a model of P^{E^+}. This model is minimal since it is a minimal model of the set of exclusive disjunctive rules of P^{E^+}. Therefore, I_ω is an answer set of P^{E^+}.

($\Leftarrow$). Suppose I_ω is an answer set of P^{E^+}. Consider I_ω as a complete interpretation. Since I_ω satisfies all exclusive disjunctive rules of P^{E^+}, I_ω cannot be empty. Suppose that I_ω contains an unsupported positive literal $does(i, a, t)$ or $state(i, X, t)$ for some smallest value of $1 \leq t < k$. This means that for the rule $state(i, X_{t-1}, t-1), \varphi \rightarrow state(i, X_t, t), does(i, a, t)$ occurring in P^E, $M \models state(i, X_{t-1}, t-1)$, but $M \not\models \varphi$. Hence $M \models \neg\varphi$ and M satisfies the antecedent of the rule $state(i, X_{t-1}, t-1), \neg\varphi \rightarrow state(i, X_t, t), does(i, b, t)$. But then $M \models does(i, b, t)$ and since this literal is supported by a rule whose antecedent is satisfied by M, $a \neq b$. But then I_ω violates an exclusive disjunction rule; contradiction. Hence, I_ω is a coherent model of P^E. □

5.5 Coherent outcomes and logic programming semantics: some conclusions

As we have seen, the coherent outcome semantics is based on the intuitive idea that an outcome of an individual player always is based on a (local) reason: the guard on the transition leading to the action is satisfied. Using a simple translation of encounters into logic programs enabled us to transfer this principle to a guiding principle for acceptable models for programs derived from such an encounter. As a consequence, we showed that coherent models of programs derived from encounters are based on an analogous principle: a model of a program P is coherent if it is a supported model of P.

This idea of coherence as supportedness lacks an important feature of acceptable models of logic programs: *minimality*, that is the idea that something is not considered to be true unless it is strictly necessary to do so. We showed however that this idea is somehow compiled into the form of the program P^E derived from an encounter E: as it turns out the special form of these programs guarantees that very supported (coherent) model is a minimal model as well.

One of our implicit objectives was to show that coherent models of encounters can be used to provide logic programs derived from encounters with an acceptable semantics. Knowing that the answer set semantics is generally considered to be the embodying of this intuitive idea, we finally compared our coherent models with answer sets. As it turned out, in order to establish this correspondence we have to encode an additional assumption taken for granted in our semantics of encounters that has to

be made explicit when using the answer set semantics: for every player at any time exactly one action should be executed.

Hence, we conclude that

- first of all, the simple principles underlying our coherent semantics of encounters can be used to provide logic programs associated with them with an intuitively acceptable meaning;
- secondly, in comparing our semantics of encounters with a semantics of logic programs, one has not only to pay attention to a careful translation of the syntactical elements, but also to the interplay between semantics and the translation;
- finally, in establishing a correspondence between semantics of systems in different domains, we have to pay attention to assumptions that might be taken for granted in the source domain but have to be made explicit in the target domain.

6 Related work and conclusions

As noted in the introduction, Howard was one of the first who considered the possibility of players being replaced by programs, and those programs inspecting each other's code. He in fact attributes the idea of games played by programs rather than people to Axelrod (1980). Howard also points at some philosophical issues that can be raised in such a setting. He argues that biologists have suggested that individuals may cooperate in situations of conflict if (a) they recognise that the other player is a close relative, or (b) if they recognise the other player as someone with whom they are playing a sequence of games. The second possibility explains the interest in for instance the repeated Prisoner's Dilemma (Axelrod 1984), while the first possibility inspired (Howard 1988; Tennenholtz 2004) and the current paper. However, both Howard (1988) and Tennenholtz (2004) interpret "close relative" in a very strict sense, namely as "identical". And as Howard rightly objects to this, "It might be objected that if the two players have the same program they are effectively the same individual, and there is really no proper play of the PD after all". We propose to replace the requirement of "identical" by "producing a certain behaviour, given my behaviour".

Several other authors have begun to consider aspects of program equilibria. Kalai et al. (2010) abstract away from programs completely, and assume that each player has a "mutually conditioned commitment device". In this setting, they prove a "commitment folk theorem", analogous to the Nash folk theorem in iterated games (Osborne and Rubinstein 1994, p. 143). Fortnow (2009) 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. Peters and Szentes (2008) 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".

One interesting conceptual point is the relationship between our notion of a coherent outcome, and the notion of a solution concept in game theory. Put crudely, a game theoretic solution concept identifies, for every game, a set of possible outcomes of the game; if we think of the solution concept as capturing some notion of rational choice, then the outcomes that a solution concept identifies in this way are the rational

outcomes of the game. A typical concern in solution concepts is to identify "fixed points" of the game: outcomes that are stable against rational defection. Thus, in non-cooperative game theory, Nash equilibrium captures stability against unilateral deviation, while in cooperative game theory, the core captures stability against coalitional deviation. In the present paper, our notion of a coherent outcome can also be understood as in one sense capturing a notion of stable outcome. However, our coherent outcomes are not linked in any way with utility; the coherent outcomes of an encounter are simply the outcomes that are mutually consistent with each player's program strategies. In this sense, our notion of a coherent outcome is quite distinct from, and prior to, the notion of a game theoretic solution concept.

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 coherent outcome is tractable.

Acknowledgments We thank the three reviewers of the *International Journal of Game Theory* for their very helpful comments and suggestions.

References

Apt KR (1990) Logic programming. In: van Leeuwen J. (ed) Handbook of theoretical computer science, vol B: formal models and semantics (B). Elsevier/MIT Press, Amsterdam/Cambridge, pp 493–574

Axelrod R (1980) Effective choice in the prisoner's dilemma. J Confl Resolut 24:3–25

Axelrod R (1984) The evolution of cooperation. Basic Books, New York

Bidoit N, Hull R (1989) Minimalism, justification and non-monotonicity in deductive databases. J Comput Syst Sci 38(2):290–325

Binmore K (1994) Game theory and the social contract, vol 1: playing fair. The MIT Press, Cambridge

Binmore K (1998) Game theory and the social contract, vol 2: just playing. The MIT Press, Cambridge

Clarke EM, Grumberg O, Peled DA (2000) Model checking. The MIT Press, Cambridge

Emerson EA (1990) Temporal and modal logic. In: van Leeuwen J (ed) Handbook of theoretical computer science, vol B: formal models and semantics. Elsevier, Amsterdam , pp 996–1072

Ferraris P, Lifschitz V (2005) Mathematical foundations of answer set programming. In: Artemov Sergei N, Barringer H, d'Avila Garcez AS, Lamb LC, Woods J (eds) We will show them! (1). College Publications, London , pp 615–664

Fitting M (2002) Fixpoint semantics for logic programming. a survey. Theor Comput Sci 278(1–2):25–51

Fortnow L (2009) Program equilibria and discounted computation time. In: Proceedings of the twelfth conference on theoretical aspects of rationality and knowledge (TARK-09), Palo Alto, CA

Gelfond M (2008) Answer sets. In: Lifschitz V, van Harmelen F, Porter B (eds) Handbook of knowledge representation, vol 3 of Foundations of artificial intelligence. Elsevier, Amsterdam , pp 285–316

Howard JV (1988) Cooperation in the prisoner's dilemma. Theory Decis 24(3):203–213

Kalai AT, Kalai E, Lehrer E, Samet D (2010) A commitment folk theorem. Games Econ Behav 69(1):127–137

Lloyd JW (1987) Foundations of logic programming, 2nd edn. Springer, Heidelberg

Osborne MJ, Rubinstein A (1994) A course in game theory. The MIT Press, Cambridge

Papadimitriou CH (1994) Computational complexity. Addison-Wesley, Reading

Peters M, Szentes B (2008) Definable and contractible contracts. Unpublished Working Paper

Tennenholtz M (2004) Program equilibrium. Games Econ Behav 49:363–373