ON THE STRUCTURE OF EVENTS IN BOOLEAN GAMES

Julian Bradfield[†] **Julian Gutierrez**[‡] **Michael Wooldridge**[‡]

[†]University of Edinburgh [‡]University of Oxford

Abstract

Conventional Boolean games embody two important assumptions: (*i*) all events in the game occur simultaneously; and (*ii*) assignments to variables must be made in complete ignorance of the values assigned to other variables. For many settings, these assumptions represent gross simplifications. In this paper, we show how Boolean games can be enriched by *dependency graphs*, which allow us to generalise the conventional Boolean games framework. A dependency graph specifies for every variable in the game what information will be available when a value is assigned to that variable. More precisely, a dependency graph is a directed acyclic graph over the set of game variables, where an edge from p to q means that the value assigned to variable p can be taken into account when choosing a value for variable q. We refer to games played with dependency graphs as *partial order Boolean games*. In partial order Boolean games, players simultaneously choose strategies that define how values will be assigned to variables; these strategies must take into account information dependencies as specified in the dependency graph. Once chosen, strategies are executed, and generate a run-time sequence of events corresponding to the assignment of values to variables as defined by the strategies. The dependency graph induces a partial order model of concurrency for run-time behaviour: if a variable q depends on p, then at run-time the assignment of a value to p must precede the assignment of a value for q. Partial order Boolean games thus distinguish between the event corresponding to the selection of a strategy and the events that arise from executing that strategy (*i.e.*, the assignment of values to variables). They thus more closely resemble the reality of multi-agent systems, in which multiple programs (representing player strategies) are concurrently executed to generate run-time behaviour. After motivating and presenting the game model, we explore its properties. We show that while some problems associated with our new games have the same complexity as in conventional Boolean games, for others the complexity blows up dramatically. We also show that the concurrent behaviour of partial order Boolean games can be represented using a closure operator semantics, and conclude by considering the relationship of our model to Independence Friendly (IF) logic.

1 Introduction

Boolean games are a compact and expressive family of logic-based games, which have a natural interpretation with respect to multi-agent systems (see, *e.g.*, [14, 22, 11, 9, 17]). A Boolean game is played over a set of Boolean variables. Each player desires the satisfaction of a goal, specified as a logical formula over the overall set of variables, and is assumed to control a subset of the variables: the choices available to a player correspond to the assignments that can be made to the variables controlled by that player. Players *simultaneously* choose valuations for the variables they control, and a player is satisfied if their goal is made true by the resulting overall valuation. Apart from being an interesting game theoretic model in their own right, it has been argued that Boolean games are a natural abstract model for studying strategic behaviour in multi-agent systems: the use of logically-specified goals is commonplace in the multi-agent systems community, and the fact that players act by assigning values to Boolean variables models the execution of finite-state computer programs.

However, if we really aim to use Boolean games as an abstract model of multi-agent systems, then existing Boolean game models embody some arguably rather extreme assumptions. First, in all studies of Boolean games that we are aware of, it is assumed that *all events in the game occur simultaneously*. There are two types of events in a Boolean game: the selection of strategies by players, and the assignments of values to variables according to these strategies. If we consider a multi-agent system setting, then it should be clear that assuming all such events occur simultaneously is unrealistic. In a multi-agent system, participants select their strategies by writing a computer program (*i.e.*, an agent) to act on their behalf, and then these various agents are concurrently executed, generating a "run-time" history of events. The actual computational histories that may be generated at run-time will depend on the model of concurrency underpinning the run-time behaviour of the agents. Concurrency is of course a huge research area, somewhat tangential to game theoretic concerns [19]; but nevertheless, the assumption that all events occur simultaneously is perhaps an abstraction too far if we want to interpret Boolean games as a model of multi-agent systems. A second difficulty in existing models of Boolean games is that players act in ignorance of all the choices made by other players. In many settings, players will have some information about the choices of others once the game progresses in time, and to faithfully capture such settings, a more realistic game model should reflect this.

In this paper we develop *partial order Boolean games*, a new model for concurrent and multi-agent systems which resolves these two difficulties. This model is somewhat closer to the concept of multi-agent systems that we described above: partial order Boolean games are games of simultaneous moves, but the moves that players make correspond to choosing a "program" (actually, a collection of Boolean functions) to play the game on their behalf. The programs then interact to actually play the game and generate run-time behaviour according to a partial-order model of concurrency. We thus distinguish between the two types of events that we mentioned above: events corresponding to the selection of strategies (these events occur simultaneously) and the run-time behaviour of these strategies as they are executed. To capture incomplete information and concurrency, we use *dependency graphs*. A dependency graph is a directed acyclic graph over the set of game variables: an edge from variable p to variable q means that when the strategy that chooses a value for q makes its assignment to this variable, it can take into account the value of p. The relationship between p and q is thus one of *functional dependence*. The total set of variables upon which q depends represents all the information that is available for a strategy assigning a value to q: if q is not dependent on a variable r, for example, then the choice of value for q cannot be informed by the value of r. While strategies in conventional Boolean games are very simple (simply an assignment of values to variables), in partial order Boolean games, strategies are more complex: *a strategy for a variable x is a Boolean function that assigns a value to x for every possible valuation of variables on which x depends*. We can think of such a function as the program written by the player to play the game on their behalf. When these strategies are executed, then they generate a run-time history of events, which correspond to the variables in the game being assigned values according to the strategies. The possible set of run-time histories that may be generated will be determined by the dependency graph, which implicitly defines a partial temporal ordering of run-time events. That is, if q depends upon p, then the run-time event corresponding to assigning a value to p must *precede* the run-time event corresponding to assigning a value for q.

Dependency in (Boolean) games. We would like to remark that we use the term "dependence" to refer to the idea that the selection of a value for one variable can take into account the value that has been assigned to another variable. The term "dependence" has been used in several other ways in game theory, some of which are related to our usage, some of which are not. One usage is to say that agent i is dependent on agent j if the choice made by j can potentially influence the utility that agent i obtains. In the context of Boolean games, this idea was investigated by Bonzon *et al* [4]. This is clearly distinct from our usage of the term. Relatedly, our model of games is also somewhat reminiscent of the idea of *multi-agent influence diagrams* (MAIDs) [15]. MAIDs are a compact representation model for extensive form games, which exploit the idea that we can reduce the state space required for representing a game by only recording information about the interactions between players where one player's choice affects another. Although partial order Boolean games are also based on a graph representation, this graph captures informational dependencies between variables. Moreover, as Bonzon *et al* point out, although MAIDs frequently provide a compact representation for games, Boolean games are in some cases exponentially more succinct than MAIDs.

Structure of the paper. The remainder of this paper is structured as follows. We begin by defining partial order Boolean games, developing a concrete model for player strategies, and investigating the computational complexity of decision problems for such games. We show that, for some problems, the complexity of the decision problem is essentially the same as the corresponding problem for conventional Boolean games, but for others, the complexity is much higher. We then define a closure operator semantics for partial order Boolean games, which more explicitly reflects the causal, temporal dependencies in our games framework. We also show a natural connection between partial order Boolean games and Independence-Friendly (IF) first-order logic—a logical formalism already studied in the context of logics for concurrency, independence, and incomplete information.

2 Partial Order Boolean Games

We build upon the Boolean games framework; see, for instance, [14, 3, 22, 11, 9, 17] for discussions and more details on Boolean games, as well as some of its applications in AI and multi-agent systems.

Let $\mathbb{B} = \{\top, \bot\}$ be the set of Boolean truth values, with "$\top$" being truth and "$\bot$" being falsity; we use $\top$ and $\bot$ to denote both the syntactic constants for truth and falsity, respectively, as well as their semantic counterparts. Let $\Phi = \{p, q, \ldots\}$ be a finite, fixed, non-empty vocabulary of Boolean variables. For every subset $\Psi \subseteq \Phi$ of Boolean variables, we denote by $\mathcal{L}_\Psi$ the set of (well-formed) formulae of propositional logic over the set Ψ, constructed using the conventional Boolean operators ("$\wedge$", "$\vee$", "$\rightarrow$", "$\leftrightarrow$", and "$\neg$"), as well as the truth constants "$\top$" and "$\bot$". For every subset of Boolean variables Ψ, a Ψ-*valuation* is a total function $v : \Psi \to \mathbb{B}$, assigning truth or falsity to the variables in Ψ; we let $\mathcal{V}(\Psi)$ denote the set of all Ψ-valuations. Also, given a formula $\varphi \in \mathcal{L}_\Psi$ and a Ψ-valuation $v : \Psi \to \mathbb{B}$, we write $v \models \varphi$ to mean that φ is true under the Ψ-valuation v, where the satisfaction relation "$\models$" is defined in the conventional way.

Our games are populated by a set $N = \{1, \ldots, n\}$ of *agents*—the players. Each agent i is assumed to have a *goal*, which is represented by an $\mathcal{L}_\Phi$-formula γ_i that i desires to have satisfied. Each player $i \in N$ *controls* a (possibly empty) subset Φ_i of the overall set of Boolean variables Φ. By "control", we mean that i has the unique ability within the game to set the value ($\top$ or $\bot$) of each variable $p \in \Phi_i$. We require that each variable is controlled by exactly one agent, that is, *i.e.*, we have $\Phi = (\Phi_1 \cup \cdots \cup \Phi_n)$ and $\Phi_i \cap \Phi_j = \emptyset$, for all $i \neq j$. Then, a *Boolean game* is given by a structure $\langle N, \Phi, (\Phi_i)_{i \in N}(\gamma_i)_{i \in N} \rangle$, where each component is formally defined as described above. Such a game is played by each player i simultaneously choosing values for their variables Φ_i in an attempt to satisfy their goal γ_i; the collection of choices made by all players defines a valuation for the variables Φ, and this valuation is the outcome of the game: under this valuation, a player's goal is either satisfied or not satisfied.

Notice that, in this model, all events in the game (the selection of strategies by players *and* the execution of these strategies, *i.e.*, the events corresponding to the assignment of values to variables) are implicitly assumed to occur *simultaneously*. Also, note that each player must choose values for all its variables in complete ignorance of the values selected by other players for their variables. We now show how to extend the Boolean games model to overcome these two limitations.

The basic idea is to augment Boolean games with a graph that we refer to as a *dependency graph*. This graph plays a dual role. First, it specifies the *functional dependencies* between variables in the game: it specifies for each variable what information is available when a value for that is assigned. Second, it implicitly specifies a partial temporal order for run-time events: if a variable q is dependent upon variable p, as specified in the dependency graph, then this means that, at run-time, variable p must be assigned a value before variable q is assigned a value.

Formally, a dependency graph over a set Φ is simply a directed acyclic graph $D \subseteq \Phi \times \Phi$; we will write $D(p,q)$ to mean $(p,q) \in D$. We read $D(p,q)$ as "q depends on p"; more precisely:

> $D(p,q)$ means that the choice of a value for the Boolean variable q by the player who controls q can be informed by the value that was assigned to the Boolean variable p.

We denote the transitive closure of D by D^*, and where $p \in \Phi$, we define $D[p] = \{q \in \Phi \mid D(q,p)\}$, and $D^*[p]$ similarly. Thus, $D[p]$ is the total set of variables whose value can be taken into consideration when choosing a value for variable p.

Note that the graph D induces a *partial temporal ordering* on run-time events in the game. That is, if $q \in D[p]$ then this implies that the assignment of a value to q must *precede* the run-time assignment of a value to p. However, in general, the dependency graph does not *totally* order choices, as we will see shortly. The motivation for the requirement that dependency graphs are acyclic should now be clear: it ensures that we will not have situations in which a variable is "waiting" on itself ($p \in D^*[p]$).

The dependency graph structure makes it explicit which choices are, or can be regarded to be, *independent*. Suppose D is a dependency graph such that $p \notin D[q]$ and $q \notin D[p]$. That is, the choice of p does not depend on q, and the choice of q does not depend on p. Then we say p and q are *independent*. More generally, where we have two connected components in the dependency graph D such that the components are not connected to each other, then these components are independent of each other; we can interpret such independent structures as concurrent processes. The dependency graph makes independence explicit and transparent, as desired in some models of concurrency [19].

Formally, a *partial order Boolean game* is simply a Boolean game together with a dependency graph, *i.e.*, a structure $G = \langle N, \Phi, (\Phi_i)_{i\in N}, (\gamma_i)_{i\in N}, D\rangle$ where $\langle N, \Phi, (\Phi_i)_{i\in N}, (\gamma_i)_{i\in N}\rangle$ is a Boolean game as defined earlier, and $D \subseteq \Phi \times \Phi$ is a dependency graph over the set of Boolean variables Φ.

We can now explain how these new games are played. In conventional Boolean games, a game is played by every player i simultaneously picking a valuation $v_i : \Phi_i \to \mathbb{B}$ for their Boolean variables Φ_i. Clearly, the choices for players in partial order Boolean games are more complex: the value chosen for a variable p can depend on the values assigned to the variables in the set $D[p]$. It therefore makes sense to model a choice for a variable p as a function that maps a valuation for the variables $D[p]$ to a value for variable p, *i.e.*, a function with the signature

$$f : \mathcal{V}(D[p]) \to \mathbb{B}$$

or, making clear that f is a second-order function:

$$f : (D[p] \to \mathbb{B}) \to \mathbb{B}$$

(Recall that $\mathcal{V}(D[p]) = \{f : D[p] \to \mathbb{B}\}$ is the set of valuations over the variables $D[p]$, and so such a function maps a valuation to the variables on which p depends to a chosen value for p.)

We represent such individual choices as equations of the form:

$$p = f(q, \ldots, r)$$

where f is a Boolean valued function, defining a value for p for every possible valuation to the variables $D[p] = \{q, \ldots, r\}$. We refer to $p = f(q, \ldots, r)$ as a *choice equation*.

To keep things mathematically simple, we will assume that the Boolean function f representing the strategy for choosing a value for p is given as a propositional formula φ over the variables on which p depends, that is, a formula $\varphi \in \mathcal{L}_{D[p]}$. Thus, the choice equations that we will consider take the form $p = \varphi$, where $p \in \Phi$ and $\varphi \in \mathcal{L}_{D[p]}$. Note that when $D[p] = \emptyset$ (in other words, the value of variable p does not depend on the value of any other variables), then the choice equation for p can be assumed to take the form $p = b$ where $b \in \mathbb{B}$ is a logical constant.

We emphasise that representing Boolean functions by propositional formulae is not a limitation: any Boolean function f of k variables can be represented by a propositional formula over these variables, although in the worst case, the smallest formula representing f may be exponential in k [5].

A *strategy* for player i, denoted by σ_i, is then a set of choice equations, one for each variable controlled by i. As usual, a *strategy profile* $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ is a collection of strategies, one for each player in the game. Since we require that dependency graphs D are acyclic, we have the following:

Lemma 1 *Let $G = \langle N, \Phi, (\Phi_i)_{i \in N}, (\gamma_i)_{i \in N}, D \rangle$ be a game and let $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ be a strategy profile for G. Then there is a unique solution to the set of equations $\sigma_1 \cup \cdots \cup \sigma_n$, which we denote by $v(\vec{\sigma})$, (and thus we have that $v(\vec{\sigma}) : \Phi \to \mathbb{B}$), and which we refer to as the* valuation induced by $\vec{\sigma}$. *Moreover, $v(\vec{\sigma})$ can be computed in time polynomial in the size of $\vec{\sigma}$.*

We can define utility functions with respect to strategy profiles in essentially the same way for conventional Boolean games:

$$u_i(\vec{\sigma}) = \begin{cases} 1 & \text{if } v(\vec{\sigma}) \models \gamma_i \\ 0 & \text{otherwise.} \end{cases}$$

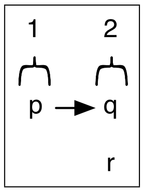

Figure 1: Dependency graph for Example 1.

Where $\vec{\sigma} = (\sigma_1, \ldots, \sigma_i, \ldots, \sigma_n)$ is a strategy profile and σ'_i is a strategy for player i, we denote by $(\vec{\sigma}_{-i}, \sigma'_i)$ the strategy profile obtained from $\vec{\sigma}$ by replacing the i component with σ'_i. We then say that $\vec{\sigma}$ is a (pure strategy) *Nash equilibrium* if we have that there is no player $i \in N$ and strategy σ'_i for player i such that $u_i(\vec{\sigma}_{-i}, \sigma'_i) > u_i(\vec{\sigma})$. Thus, the strategy profile $\vec{\sigma}$ is a Nash equilibrium if no player could benefit by unilaterally deviating from $\vec{\sigma}$ to another strategy. Let $NE(G)$ denote the Nash equilibria of G.

Let us consider a small example, to further illustrate the notion of independence between variables. Imagine a captain who controls two army platoons that must engage in two separate battles, where there is no way of the platoons communicating with each other. Then, the captain has to design strategies that will be executed by the platoons *independently*: that is, even though the captain can choose strategies for both platoons, he must choose strategies so that *no communication or coordination between the platoons takes place while they execute their respective strategies*. The following formal representation of this example makes this point explicit.

Example 1 *Consider a partial order Boolean game where $N = \{1, 2\}$, $\Phi = \{p, q, r\}$, $\Phi_1 = \{p\}$, $\Phi_2 = \{q, r\}$, $\gamma_1 = \top$, and $\gamma_2 = (p \leftrightarrow q) \wedge (r \leftrightarrow \neg q)$. The dependency graph is as in Figure 1. Now, player 2 has a choice for variable q that is guaranteed to make the left conjunct of his goal true—the required choice equation is simply: $q = p$.*

But although player 2 controls both r and q in the right conjunct of γ_2, there is no strategy that guarantees to make both conjuncts true simultaneously. This is precisely because r is independent of q: a choice for this value must be made independently of q. As can be seen, the only two choice equations available for variable r are as follows: $r = \top$ and $r = \bot$.

Note that in the preceding example, player 2 is perfectly able to coordinate the selection of strategies for variables q and r, but at run-time, these strategies cannot communicate with each other, and in particular, the strategy for variable r must assign a value to this variable without knowing anything about the values of p or q. We emphasise that this is not a "problem" in our model: it accurately reflects real-world situations in which parts of a strategy must be executed even though those parts of the strategy cannot communicate or coordinate with each other *while* they are being executed. One might wonder whether it would be simpler in this case to treat independent variables as separate players. We argue that this might not make sense. The next example hopefully illustrates why not.

Example 2 *Suppose we have a game where $N = \{1,2\}$, $\Phi_1 = \{p,q\}$, $\Phi_2 = \{r,s\}$, $\gamma_1 = \top$, and $\gamma_2 = ((r \leftrightarrow p) \wedge (s \leftrightarrow q)) \vee (\neg(r \leftrightarrow p) \wedge \neg(s \leftrightarrow q))$. Thus, player 1 is indifferent about how to assign values to his variables, while player 2 will be satisfied if either:*

- *r takes the same value as p and s takes the same value as q; or else*
- *r takes the opposite value to p and q takes the opposite value of s.*

The dependency graph for this example is illustrated in Figure 2. Now, player 2 clearly has choice equations that will guarantee the achievement of his goal. Consider the choice equation set

$$\sigma_2^1 = \{r = p, s = q\}$$

and the set

$$\sigma_2^2 = \{r = \neg p, s = \neg q\}.$$

Since either of these strategy sets guarantees to achieve player 2's goal, they each represent a dominant strategy for player 2. Now, suppose we treated our current agent 2 as two separate agents, say agents 3 and 4, the former controlling variable r and the latter controlling variable s, each new agent with the same goal (γ_2, above), and with the same dependency graph in Figure 2. Then, the two strategy sets above induce Nash equilibria (for instance, letting $\sigma_3^1 = \{r = p\}$ and $\sigma_4^1 = \{s = q\}$), but such new strategy sets do not *yield dominant strategies for the two new players of the game. Thus, we argue, in attempting to treat the independent variables r and s as being controlled by different players, we fundamentally change the character of the game.*

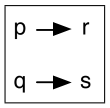

Figure 2: Dependency graph for Example 2.

Let us now consider another example, to further illustrate the distinction between our new model and conventional Boolean games.

Example 3 *We will explore some variations of the non-cooperative game of* matching pennies. *Suppose we have $N = \{1,2\}$, $\Phi_1 = \{p\}$, $\Phi_2 = \{q\}$, $\gamma_1 = p \leftrightarrow q$, and $\gamma_2 = \neg(p \leftrightarrow q)$. Thus, player 1 controls variable p, and player 2 controls variable q; player 1 wants both variables to take the same value, while player 2 wants them to take different values.*

Now, consider the three different dependency graphs for this game shown in Figure 3:

- *In the dependency graph shown in Figure 3(a), variables p and q are independent: the choice for p is made in ignorance of the choice for q, and vice-versa.*
- *In Figure 3(b), q is dependent on p, and so the choice of value for q takes place after the choice of value for p is made; moreover, the choice of value for q is informed by the choice for p.*
- *In Figure 3(c), p is dependent on q. Here, the choice of value for p takes place after the choice of value for q, and the choice of value for p is informed by the choice of value for q.*

Let us consider (a) first. It should be easy to see that there is no Nash equilibrium for the game with dependency graph (a). Consider the following strategy profile: (σ_1) $p = \bot$ *and* (σ_2) $q = \bot$.

With this outcome, γ_1 *is satisfied, while* γ_2 *is not. In this case, player* 2 *could deviate with strategy* $q = \top$*, resulting in the satisfaction of player* 2*'s goal. In fact, it is easy to see that, for every possible strategy, one of the players will have a beneficial deviation, and so this game has no Nash equilibria.*

Now consider the dependency graph in Figure 3(b). Player 2 *can see the value of p before assigning a value to q, and has four options available with respect to his strategy:* $q = \top$, $q = \bot$, $q = p$, *and* $q = \neg p$. *With the final option,* $q = \neg p$, *player* 2 *guarantees to get his goal achieved (hence this strategy* weakly dominates *all of the other strategies for player* 2*). Observe that any strategy profile in which player* 2 *uses this strategy is a Nash equilibrium: as player* 2 *has his goal achieved, he cannot beneficially deviate, and no deviation on the part of player* 1 *could improve his position.*

For the dependency graph in Figure 3(c), the situation is reversed: p is dependent on q. In this case, player 2 *must choose a value for p, and player* 1 *is then able to choose a value for p while knowing which value was assigned to q. In this case,* 1 *can guarantee to get his goal achieved by using the strategy* $p = q$.

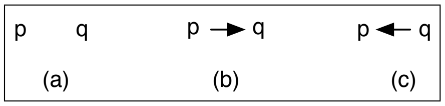

Figure 3: The only three possible dependency graphs for Example 3.

As this example shows, partial order Boolean games strictly generalise conventional Boolean games: the implicit choice structure of a conventional Boolean game (where every player simultaneously and independently chooses a valuation for their variables) is obtained in a partial order Boolean game in which all variables are made independent of each other, that is, letting $D[p] = \emptyset$ for every $p \in \Phi$.

The example above also illustrates a more useful fact about the implications on the sets of equilibria when allowing dependencies. That is, that having at hand the additional strategic power of using functional dependencies between variables may increase the set of Nash equilibria of a game, an arguably desirable property from a game-theoretic perspective.

3 Complexity of Partial Order Boolean Games

Let us now consider some questions relating to the complexity of partial order Boolean games. The most obvious question to ask relates to verifying whether a particular strategy profile is a Nash equilibrium. The decision question is stated as follows.

VERIFICATION:
Given: Partial Order Boolean game G, strategy profile $\vec{\sigma}$.
Question: Is it the case that $\vec{\sigma} \in NE(G)$?

We emphasise that the representation of strategies we consider in this problem is that based on propositional formulae, as a set of choice equations as described in the preceding section.

The corresponding problem for conventional Boolean games is co-NP-complete [3], and it might be natural to suppose that the problem would be harder for partial order Boolean games, particularly given that strategies are more complex objects than in conventional Boolean games. In fact, this is not the case: the complexity is no worse than that of conventional Boolean games. The key fact is:

Lemma 2 *Let G be a partial order Boolean game containing a player i and let $\vec{\sigma}$ be a strategy profile for this game. Then, if i has a beneficial deviation from $\vec{\sigma}$, there is a beneficial deviation σ_i' for player i which is of size polynomial in $|\Phi_i|$.*

So, if player *i* has a beneficial deviation, then there is, in the terminology of computational complexity theory, a *small certificate* to this effect. In light of results that will follow later, this result is perhaps surprising; we will postpone discussion of this point for a moment. Returning to the VERIFICATION problem, Lemma 2 allows us to prove the following complexity result.

Proposition 1 VERIFICATION *is co-NP-complete; moreover, it can be solved in time exponential in the number of Boolean variables Φ, but polynomial in the size of the given strategy profile.*

The next problem to consider is whether a given game has any Nash equilibria. The decision question is stated as follows.

NON-EMPTINESS:
Given: Partial Order Boolean game *G*.
Question: Is it the case that $NE(G) \neq \emptyset$?

The NON-EMPTINESS problem for conventional Boolean games is Σ_2^p-complete [3], and the fact that VERIFICATION is no harder for partial order Boolean games might lead one to hypothesise that the NON-EMPTINESS problem is also no harder. In fact, the complexity is dramatically different:

Proposition 2 NON-EMPTINESS *is NEXPTIME-complete.*

Now, in light of Lemma 2, which demonstrated that if a player has a beneficial deviation then there is a small certificate to this effect, this result may seem surprising. However, there is a good reason for the for the difference in complexity reported in Lemma 2 and Proposition 2. Whereas in the former one can replace potentially large strategies with the values those strategies choose, in the latter this cannot be done. The following example illustrates why not.

Example 4 *Recall Example 3, and the dependency graphs in Figure 3. In particular, take the graph in Figure 3(b). Given this dependency graph, the following strategy profile forms a Nash equilibrium:*

$$\begin{array}{ll} (\sigma_1) & p = \bot \\ (\sigma_2) & q = \neg p. \end{array}$$

This is because, using σ_2, player 2 guarantees to get his goal achieved, and there is no deviation for player 1 that would achieve his goal: whatever choice player 1 makes for p, player 2 simply negates it. Now, let us consider what happens if we try to apply this same trick (used in Lemma 2), namely, replacing each strategy by the value that the strategy chooses. This would result in the strategy profile:

$$\begin{array}{ll} (\sigma_1) & p = \bot \\ (\sigma_2') & q = \top. \end{array}$$

However, this strategy profile does not *form a Nash equilibrium: player* 1 *could now beneficially deviate using* $p = \top$. *In fact, there is no Nash equilibrium of this game in which both players use strategies of the form* $x = b$, *where* b *is a logical constant. The trick works in Lemma 2 because when considering whether a player has a beneficial deviation, we do not need to take into account how other players would respond to our deviation: we only need to know whether such a beneficial deviation is possible against a profile of strategies for other players that can be assumed to be fixed.*

Even though the complexity of NON-EMPTINESS is NEXPTIME-complete w.r.t. unrestricted dependency graphs, the problem is easier when restricted to some important classes of graphs:

Proposition 3

1. *For games in which* $D = \emptyset$, *the* NON-EMPTINESS *problem is* Σ_2^p*-complete.*
2. *For partial order Boolean games of the following form,* NON-EMPTINESS *is PSPACE-complete: We have* $N = \{1, 2\}$ *with* $\Phi_1 = \{x_1, \dots, x_k\}$ *and* $\Phi_2 = \{y_1, \dots, y_k\}$. *The dependency graph is defined by* $D[x_i] = \{x_j, y_j \mid 1 \le j < i\}$ *and by* $D[y_i] = \{x_i\} \cup \{x_j, y_j \mid 1 \le j < i\}$.

Observe that in case (1), we obtain conventional Boolean games, for which the NON-EMPTINESS problem is Σ_2^p-complete. The second case essentially corresponds to QBF, which is PSPACE-complete.

Let us now turn our attention to semantic features of partial order Boolean games. In particular, we are now going to study how concurrent behaviour is modelled by our partial order Boolean games.

4 Concurrency in Partial Order Boolean Games

As discussed before, one important feature of partial order Boolean games is that they provide a game-based model of concurrency. In this section, we study this feature in more detail, and show that the kind of concurrency seen in partial order Boolean games can be mathematically represented using a *closure operator semantics* for players' strategies. This closure operator semantics, in turn, will give a more explicit representation of the causal and temporal dependences in our games framework.

In particular, even though the temporal order in which events happen in partial order Boolean games is not fully known only by knowing the strategies to be played—because unordered events *may* happen in *parallel*, whereas ordered events *must* happen *sequentially*—it is, nonetheless, the case that once each player chooses a strategy, regardless of how concurrent (unordered) events are executed, the system (game) always progresses towards the same final outcome: the valuation $v(\vec{\sigma})$ induced by the strategy profile $\vec{\sigma}$ under consideration. This kind of *deterministic*, yet concurrent, behaviour can be mathematically represented by a closure operator semantics of the game, which we formalise next.

Let $\le$ be the reflexive and transitive closure of D. Since D is a DAG, the structure $(\Phi, \le)$ is a partial order. Where $p, q \in \Phi$, let $D[p] \le_D D[q]$ whenever $p \le q$. Thus, $(2^\Phi, \le_D)$ is also a partial order, where $\le_D$ is just the lifting of $\le$ from elements in Φ to $\le$-downclosed sets in 2^Φ. Thus, if $X \le_D Y$ then $X \subseteq Y$.

Now, for each player i, define the function $\pi_i : 2^\Phi \to 2^\Phi$, where $X \in 2^\Phi$, as follows:

$$\pi_i(X) = X \cup \{q \in \Phi_i \mid (D[q] \cap (\Phi \setminus \Phi_i)) \subseteq X\}.$$

The function π_i indicates, for each player i, which variables are to be played next, that is, to which variables give values provided that the variables in X have been already set to some Boolean value.

The key observation in this section is that, in fact, over the partial order set $(2^\Phi, \le_D)$, the function π_i determines a stable closure operator, that is, a map being:

1. **(Extensive):** $\forall X.\, X \leq_D \pi_i(X)$,
2. **(Idempotent):** $\forall X.\, \pi_i(X) = \pi_i(\pi_i(X))$, and
3. **(Monotone):** $\forall X, Y.\, X \leq_D Y \Rightarrow \pi_i(X) \leq_D \pi_i(Y)$.

We then can show:

Proposition 4 *Let D be the dependency graph in a partial order Boolean game where $N = \{1, \dots, n\}$ and $\Phi = (\Phi_i)_{i \in N}$. Then, each function π_i for every player i, as defined above, is a closure operator.*

Based on $(\pi_i)_{i \in N}$ we can define a semantics which gives a more operationally informative explanation of how concurrent behaviour in a multi-agent system is modelled in partial order Boolean games. What we are to define next is a closure operator semantics of concurrency in our framework.

Let $X \in 2^{\Phi}$ be a *history* of play if X is $\leq_D$-downclosed, that is, X indicates which variables have been set to a fixed Boolean value so far. Given a history of play X, each agent i knows what to play next (using the map τ_i, defined below) based on both π_i and the choice equations, denoted by f_q, for the variables $q \in \Phi_i$ they have control over, in the following way:

$$\tau_i(X) = \{q = f_q \mid q \in \pi_i(X)\}.$$

Then, note that even though τ_i can also be regarded as a strategy for player i, semantically, the map τ_i behaves differently from σ_i. The former takes histories of play and, based on that, gives only a few necessary choice functions to be used next (a rather operational approach); the latter, instead, provides a player with all choice equations that i has available at once, with no information as to when exactly such equations should be used (a more denotational approach). Thus, we can think of τ_i as being more operationally informative when compared with σ_i, since "time" is implicitly considered. Due to the informal description given above, we say that the map τ_i is a *temporal strategy* for player i. In fact, because each π_i is a stable closure operator, a few simple observations also follow.

Firstly, since a strategy can only modify the state of play by setting a value for a variable not played yet, and once such a value has been determined it cannot be modified, the strategy must naturally be an *extensive* operator. Secondly, because the behaviour of different agents must be independent on concurrent variables, such a kind of behaviour must be oblivious to superfluous or unnecessary alternations between players' moves. In other words, in the end, it has to be irrelevant who plays first in concurrent (unordered) events—of course this is not the case for sequential (ordered) events; thus, the strategy must also be *idempotent* to ensure this kind of behaviour. Finally, as dependency graphs induce a temporal (and causal) dependency relation on events, playing at a later "time unit" must provide at least as much information as playing at a previous one; in other words, a temporal strategy must be *monotone* so that causal dependencies are preserved when playing the game.

The idea of using closure operators to model deterministic concurrency has been successfully used in the past, *e.g.*, see [2, 12, 21]. This representation is very appealing because it ensures—due to the mathematical properties of closure operators—that the particular execution of unordered (concurrent, independent, etc.) events does not affect the outcome of the game. Let us now see, with a simple example, how a temporal strategy constructed with respect to a stable closure operator looks like.

Example 5 *Take again the game in Example 1. There, the game can proceed in five different ways in terms of how/when the events are played. We will write $p \parallel r$ and $q \parallel r$ for the parallel execution of two concurrent/independent events. Then, one of the following behaviours can be observed (assuming that executing atomic events takes no time and that "." is sequential composition):*

$$p.q.r;\ p.r.q;\ p.(q \parallel r);\ r.p.q;\ \text{or}\ (r \parallel p).q.$$

The temporal strategies τ_1 and τ_2 for players 1 and 2 must be defined for all those states of play when each player is to make a move. Moreover, whenever both players are allowed to play simultaneously, the result of playing their strategies must deliver the same result. This is the case since:

- $\tau_1 = \{(\emptyset, \{p = f_p\}), (\{p\}, \emptyset), (\{r\}, \{p = f_p\}), (\{p,q\}, \emptyset), (\{p,r\}, \emptyset), (\{p,q,r\}, \emptyset)\}$*; and*
- $\tau_2 = \{(\emptyset, \{r = f_r\}), (\{p\}, \{r = f_r, q = f_q\}), (\{r\}, \emptyset), (\{p,q\}, \{r = f_r\}), (\{p,r\}, \{q = f_q\}), (\{p,q,r\}, \emptyset)\}$.

As we can see, the two strategies in Example 5 seem to be bigger than needed. For instance, a more succinct representation could be:

- $\tau_1 = \{(\emptyset, \{p = f_p\}), (\{r\}, \{p = f_p\})\}$; and
- $\tau_2 = \{(\emptyset, \{r = f_r\}), (\{p\}, \{r = f_r, q = f_q\}), (\{p,q\}, \{r = f_r\}), (\{p,r\}, \{q = f_q\})\}$.

as we know that in all other cases the strategies map to $\emptyset$. Clearly, this is a valid succinct representation since it allows all possible five interleavings of events given before. In general, this kind of succinct representation is *always possible* since any τ_i will always map to $\emptyset$ in those states of play X that correspond to the *closed* sets (the fixed points) of the closure operators π_i w.r.t. which they are defined.

5 A Logic with Structured Choice

The concurrency features of partial order Boolean games also find applications to logic. Specifically, partial order Boolean games give a model for Sandu and Hintikka's Independence-Friendly (IF) logic. IF logic is an extension of first-order logic where quantifiers are partially instead of totally ordered. This gives a considerable increase in expressivity. IF logic (without negation) is equivalent to existential second-order logic—in complexity terms captures NP properties of finite models. Well known applications in, for instance, computer science, artificial intelligence, and multi-agent systems include logics for social software [20] and logics for independence and concurrency [6].

The syntax and semantics of IF logic is as follows. The syntax of IF logic is that of classical first-order logic with quantifiers extended to *slashed quantifiers* $\exists x/y,z,\ldots$, $\forall x/y,z,\ldots$, whose intended interpretation is that when choosing the witness/counter-example x, we may not know the values of $y,z,\ldots$ (which are presumed to be bound earlier in the formula). For example, in

$$\forall x.\exists y.\forall u/x,y.\exists v/x,y.\varphi(x,y,u,v)$$

variable v depends only on u, not on x or y (and y depends only on x as u,v have not been mentioned).

In other words, the semantics is given by a partial order Boolean game where $\{(x,y),(u,v)\} \subseteq D$. In general, these dependencies define a partial order which, under our semantics, is interpreted with the dependency graph D. We should note that our games are not quite IF, as we require dependency to be transitive. The standard definition of IF allows for non-transitive dependencies between quantifiers, which does not increase the expressive power of the logic, but does result in rather pathological properties of formulae. For this reason, some IF work imposes transitivity as an additional requirement.

Example 6 *Recall Example 3. In the usual presentation of the game, the dependency graph is that in Figure 3(a): the variables are independent. So, player 1 can achieve her goal $p = q$ iff $\forall q.\exists p/q.p = q$ and player 2 can achieve his goal $p \neq q$ iff $\forall p.\exists q/p.p \neq q$. Clearly, no player has a winning strategy. In fact, no strategy profile forms a pure-strategy Nash equilibrium, as shown in Example 3.*

IF logic has a prenex normal form, though it is substantially more complex to generate than for first-order logic when non-transitive quantifier dependencies are allowed.

Proposition 5 *Let $\varphi = \forall \ldots \psi$, where ψ is a Boolean formula, be a transitively-dependent IF formula in prenex normal form where all existentially quantified variables are in $\Phi_\exists$ and all universally quantified variables are in $\Phi_\forall$. Let G be the game $G = (\{\exists, \forall\}, \Phi, \Phi_\exists, \Phi_\forall, \psi, \neg\psi, D)$ with D given by the variables and quantification order in φ. Then*

- *φ is true (in IF logic) iff ψ is true in every Nash equilibrium of game G, and such an equilibrium exists (in which case $\exists$ has a winning strategy);*
- *φ is false (in IF logic) iff ψ is false in every Nash equilibrium of game G, and such an equilibrium exists (in which case $\forall$ has a winning strategy);*
- *φ is undetermined otherwise.*

Partial Order Boolean games may have many players, not merely two. In coalition logics [20], a team of players competes against the others to achieve a common goal. We can generalize the previous result by considering the coalition as an $\exists$ team, and all other players as a $\forall$ team.

An IF formula may be expressed as a partial order Boolean game, but partial order Boolean games may have many players with different goals, rather than just two players with complementary goals. In the remainder of this section, we analyse further the relationship between IF and partial order Boolean games. Consider first two-player games with variable ownerships Φ_1, Φ_2 and arbitrary goals γ_1, γ_2. In the case that the ψ_i are contradictory, the game corresponds to an IF game, with one player taking the role of $\forall$, the other being $\exists$, and quantifier dependencies given by dependency graph D.

However, if the ψ_i are equivalent, and there is a common goal, the best strategy for the players is to cooperate, and they should both take the role of $\exists$ in an IF game, forming a single team of two players. In such a game, there are no universal quantifiers, but there is still a role for $\forall$, as the goal ψ may contain conjuncts—$\forall$ is played by the malevolent environment (demonic non-determinacy in the terminology of concurrency theory). Thus, the IF quantifier prefix is entirely existential, with slashes given by D. As with classic coordination games, a pair of strategies is in equilibrium if it achieves the common goal; but a pair of strategies that does not achieve the goal may also be in equilibrium, if neither player can unilaterally change strategy to achieve the goal. An example of such a situation is:

Example 7 *Let player 1 own variable p and player 2 own variable q, with empty dependency and let both players have goal $p \wedge q$. In IF terms, this is expressed by the formula $\exists p.\exists x/p.\ p \wedge q$. Consider the strategy profile $((p = \bot), (q = \bot))$. This strategy profile loses—that is, no player gets its goal achieved—but neither player can unilaterally deviate to win the game.*

The IF formulation points up an issue already mentioned before: the existential team has communication barriers between its players, not just lack of knowledge of the opponents' moves. In fact, Hintikka's original formulation of IF did not allow slashing with previous existential variables, only with previous universal variables. The case where one goal is stronger, say $\gamma_1 \rightarrow \gamma_2$, is more interesting. If player 1 can win, then player 2 wins also; but player 2 may try to win while falsifying γ_1. A pair of strategies that achieves γ_1 is in equilibrium; a pair of strategies that achieves $\gamma_2 \setminus \gamma_1$ is only in equilibrium if there is no way for 1 to achieve γ_1. If we move further to incomparable goals, then the situation becomes more complex still, and the connection to IF logic becomes harder to establish. This leads to the notion of extensions of IF logic, which we will discuss in the following section.

6 Related Work

Semantics. Partial order Boolean games form a game-based model of multi-agent and distributed systems with the power to *explicitly* represent partial order concurrent behaviour—the sometimes called "true concurrency" framework within the logic and semantics of concurrency communities. This is a feature that has attracted much attention from a semantics viewpoint, in particular, because it can be used to give semantically finer representations/models of concurrent behaviour.

Closer to our work are the mathematical presentations of player strategies using *closure operators*, a semantic approach not usually associated with true concurrency concepts but which has very natural connections to it. Specifically, from the different uses of closure operators in the literature of semantics and concurrency theory, the following works are mathematically similar to the presentation we gave in this paper: closure operator semantics used to provide models for fragments of linear logic [2], processes in concurrent constraint programs [21], or agents in asynchronous systems [18].

Concurrency. Our model of partial order Boolean games has been partly inspired by a model of concurrency, a relationship on which we expand next. The dependency graphs defined for our game model are related to *event structures*, a canonical model of the partial order behaviour of concurrent systems [19]. Event structures are certain partial orders of events which can explicitly model the causal dependencies between the events that a computing system performs. In event structures, concurrency (independence of events) and nondeterminism (conflicts between events) are naturally captured.

Our model of partial order Boolean games relates to a class of event structures called *conflict-free*. Conflict-free event structures are the sub-model of event structures where incomparable events in the partial order are necessarily concurrent. Such a sub-class of event structures contains all *effective* computations of a system, that is, those that can actually happen as all conflicts between events are avoided. These structures are also related to some, so-called, domains of information—certain ordered structures where the notion of concurrent computation is naturally captured.

Game models of event structures [7, 8] are, however, different from partial order Boolean games. Three main differences are: (*i*) whereas in the former game model the execution of events is asynchronous, in the latter it is synchronous; (*ii*) moreover, players in games on event structures are allowed, at any time, to stop playing the game, whereas in our game model all players must play until every variable has been given a value; (*iii*) finally, games on event structures as in [7, 8] are two-player zero-sum games, while the games in this paper are, in general, n-player non-zero-sum games. Only recently [13], concurrent games on event structures have been extended to the non-zero-sum setting.

Logic. IF logic is inherently a two-player logic—the (partial order) games induced by IF logic formulae are played by two players, $\exists$ and $\forall$. Partial order Boolean games suggest that we could define a natural n-player logic of imperfect information, which would have IF as a two-player fragment. This idea has also been explored using a categorical model [1]. However, there is not much we can positively say about a logic of imperfect information of this kind, given the limitations that IF imposes. In particular, from a logical and proof theoretic viewpoint, the issue of providing a sound and complete axiomatisation for validity. Basically, since IF does not contain classical negation, (bi-)implication is not part of the logic, and so logical equivalence must be defined and axiomatised in its own right. Owing to its partial second-order power, IF is too expressive for either equivalence or validity to have a recursively complete axiomatisation. Thus, since IF logic is encodable into partial order Boolean games, any logic fully capturing partial order Boolean games will not be completely axiomatisable. However, as IF logic itself, may be equipped with rules adequate for practical reasoning—*cf.* [16].

7 Conclusions and Future Work

By extending Boolean games in a simple and intuitive way, *i.e.*, by adding a dependency graph that indicates the dependencies between variables in a game, we are able to dramatically increase the expressive power of conventional Boolean games. Of particular importance, we can now more easily represent the behaviour of concurrent systems and provide to them a game semantics where concurrency can be modelled using a closure operator semantics. Our model is closely related to work on IF logics, which are increasingly used in mathematical logic, game theory and the theory of concurrency.

It would be interesting to examine several aspects of partial order Boolean games in more detail. First, it should be clear from our comments on event structures above that there are very close links between partial order Boolean games and models of concurrency. It would be interesting to study these manifest similarities in more detail, perhaps by looking at Boolean game-like refinements of event structures and related models of, so-called, true concurrency. While event structures have been studied from a game theoretic perspective before, our work may give a new perspective on this work. Another aspect of our model that deserves further study is the link to games of incomplete information, and, perhaps even more relevantly from a computer science perspective, work on epistemic logics [10]. While the way in which our model captures the limited information available to players is transparent, it would be interesting to try to link this more precisely to the standard models of knowledge used in computer science (*i.e.*, Kripke structures/possible worlds semantics). Finally, it would be interesting to take a closer look at restrictions on dependency graphs, and the complexity classes of the corresponding decision problems in partial order Boolean games.

Acknowledgements. Gutierrez and Wooldridge acknowledge with gratitude the financial support of the ERC Advanced Investigator Grant 291528 ("RACE") at Oxford.

References

[1] S. Abramsky. Socially responsive, environmentally friendly logic. *Acta Philosophica Fennica*, 78, 2006. Truth and Games: Essays in Honour of Gabriel Sandu.

[2] S. Abramsky and P.-A. Melliès. Concurrent games and full completeness. In *LICS*, pages 431–442. IEEE Computer Society, 1999.

[3] E. Bonzon, M. Lagasquie, J. Lang, and B. Zanuttini. Boolean games revisited. In *ECAI*, 2006.

[4] E. Bonzon, M.-C. Lagasquie-Schiex, and J. Lang. Dependencies between players in Boolean games. *International Journal of Approximate Reasoning*, 50(6):899–914, 2009.

[5] R. B. Boppana and M. Sipser. The complexity of finite functions. In *Handbook of Theoretical Computer Science Volume A: Algorithms and Complexity*, pages 757–804. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.

[6] J. C. Bradfield. Independence: logics and concurrency. *Acta Philosophica Fennica*, 78:47–70, 2006. Truth and Games: Essays in Honour of Gabriel Sandu.

[7] P. Clairambault, J. Gutierrez, and G. Winskel. The winning ways of concurrent games. In *LICS*, pages 235–244. IEEE Computer Society, 2012.

[8] P. Clairambault, J. Gutierrez, and G. Winskel. Imperfect information in logic and concurrent games. In *Computation, Logic, Games, and Quantum Foundations*, volume 7860 of *LNCS*, pages 7–20. Springer, 2013.

[9] P. E. Dunne, S. Kraus, W. van der Hoek, and M. Wooldridge. Cooperative Boolean games. In *AAMAS*, 2008.

[10] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. The MIT Press: Cambridge, MA, 1995.

[11] J. Grant, S. Kraus, M. Wooldridge, and I. Zuckerman. Manipulating Boolean games through communication. In *IJCAI*, 2011.

[12] J. Gutierrez. Concurrent logic games on partial orders. In *WoLLIC*, volume 6642 of *LNCS*, pages 146–160. Springer, 2011.

[13] J. Gutierrez and M. Wooldridge. Equilibria of concurrent games on event structures. In *LICS*. ACM Press, 2014. To appear. DOI:10.1145/2603088.2603145.

[14] P. Harrenstein, W. van der Hoek, J.-J. Meyer, and C. Witteveen. Boolean games. In *TARK*, pages 287–298, 2001.

[15] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior*, 45(1):181–221, 2003.

[16] A. L. Mann, G. Sandu, and M. Sevenster. *Independence-friendly logic. A game-theoretic approach.*, volume 386 of *LMS Lecture Note Series*. Cambridge University Press, 2011.

[17] M. Mavronicolas, B. Monien, and K. W. Wagner. Weighted Boolean formula games. In *WINE*, pages 469–481, 2007.

[18] P.-A. Melliès and S. Mimram. Asynchronous games: Innocence without alternation. In *CONCUR*, volume 4703 of *LNCS*, pages 395–411. Springer, 2007.

[19] M. Nielsen and G. Winskel. Models for concurrency. In *Handbook of Logic in Computer Science*. Oxford University Press: Oxford, England, 1995.

[20] M. Pauly. *Logic for Social Software*. PhD thesis, University of Amsterdam, 2001.

[21] V. A. Saraswat, M. C. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *POPL*, pages 333–352. ACM Press, 1991.

[22] M. Wooldridge, U. Endriss, S. Kraus, and J. Lang. Incentive engineering for Boolean games. *Artificial Intelligence*, 195:418–439, 2013.