# Partial-order Boolean games

**Synthese manuscript No.**
(will be inserted by the editor)

# Partial-Order Boolean Games: Informational Independence in a Logic-based Model of Strategic Interaction

**Julian Bradfield · Julian Gutierrez · Michael Wooldridge**

Received: date / Accepted: date

**Abstract** As they are conventionally formulated, Boolean games assume that players make their choices in ignorance of the choices being made by other players – they are games of simultaneous moves. For many settings, this is clearly unrealistic. In this paper, we show how Boolean games can be enriched by *dependency graphs* which explicitly represent the informational dependencies between variables in a game. More precisely, dependency graphs play two roles. First, when we say that variable $x$ depends on variable $y$, then we mean that when a strategy assigns a value to variable $x$, it can be informed by the value that has been assigned to $y$. Second, and as a consequence of the first property, they capture a richer and more plausible model of *concurrency* than the simultaneous-action model implicit in conventional Boolean games. Dependency graphs implicitly define a partial ordering of the run-time events in a game: if $x$ is dependent on $y$, then the assignment of a value to $y$ must precede the assignment of a value to $x$; if $x$ and $y$ are independent, however, then we can say nothing about the ordering of assignments to these variables—the assignments may occur concurrently. We refer to Boolean games with dependency graphs as *partial-order Boolean games*. After motivating and presenting the partial-order Boolean games 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 concurrency in partial-order Boolean games can be modelled using a closure-operator semantics, and conclude by considering the relationship of our model to Independence-Friendly (IF) logic.

**Keywords** Boolean games · Foundations of games · Concurrency theory · Logic

## 1 Introduction

*Boolean games* are a family of logic-based games which have a natural interpretation with respect to multi-agent systems—see, e.g., [20,5,35,17,12,27]. 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. In addition to being an interesting game-theoretic model in their own right, it has been

Julian Bradfield
University of Edinburgh
E-mail: jcb@inf.ed.ac.uk

Julian Gutierrez
University of Oxford
E-mail: julian.gutierrez@cs.ox.ac.uk

Michael Wooldridge
University of Oxford
E-mail: mjw@cs.ox.ac.uk

argued that Boolean games are a natural abstract model for studying strategic behaviour in multi-agent systems: the use of logical goals is commonplace in the multi-agent systems community, and the fact that players act by assigning values to Boolean variables naturally models the execution of computer programs.

However, if one aims to use Boolean games as a model for 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 implicitly 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 (a software 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 large research area, somewhat tangential to game-theoretic concerns [29]; 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 realistic 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 (being 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*. When these strategies are executed, 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 on $p$, then the run-time event corresponding to assigning a value to $p$ must *precede* the run-time event corresponding to assigning a value for $q$. However, if $p$ and $q$ are independent, then we can say nothing about the ordering of assignments to these variables: they may be concurrent.

**Structure of the paper.** The remainder of this paper is structured as follows.

- We begin by motivating our work, and in particular discussing how our work relates to conventional strategic and extensive game models.
- We then define the formal framework of 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 go on to discuss how our work relates to work in concurrency. We first define a closure-operator semantics for partial-order Boolean games, which more explicitly reflects the causal, temporal

dependencies in our framework. We then 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.
– We conclude with a discussion of related and future work.

## 2 Motivation

To understand the motivation for our work, let us begin by recalling some standard game-theoretic ideas and the assumptions underpinning them. Probably the most widely studied and widely applied game-theoretic model is that of *strategic games* [30, p.11]. A strategic game is given by a structure $\langle N, (S_i)_{i\in N}, (u_i)_{i\in N}\rangle$, where $N = \{1, \ldots, n\}$ is a finite set of players, $S_i$ is the set of strategies or actions for player $i \in N$, and $u_i \colon S_1 \times \cdots \times S_n \to \mathbb{R}$ is the utility function for player $i \in N$, which defines the utility that player $i$ would obtain for every possible combination of strategies that might be chosen by players. Such a game is played by each player $i$ choosing a element of $S_i$; the outcome of the game is then a strategy profile $\mathbf{s} = \langle s_1, \ldots, s_n\rangle$, and each player $i$ receives a payoff of $u_i(\mathbf{s})$. It is assumed that players will seek to choose an element of $S_i$ so as to maximise the payoff they receive in the outcome $\mathbf{s}$, and that they will take into account the fact that other players are seeking to do likewise.

There are many assumptions implicit in this model and the description we give above, of which one of the most important is that players make their choices in ignorance of the choices of other players. This assumption is often called the assumption of *simultaneous moves*. This terminology suggests that the assumption relates to the *ordering of events* in the game (i.e., that everything happens simultaneously), but in fact it is an *informational* assumption, intended to capture the idea that players make their choices in ignorance of the choices of others. This assumption is easily motivated in standard game-theoretic settings, where the main issues of consideration relate to what decisions will be made by players in the game, rather than on the temporal ordering of events within the game.

In many settings, of course, the assumption that players act in complete ignorance of the choices made by others is not appropriate. The standard game-theoretic model used to capture such settings is that of *extensive form games of imperfect information* [30, p.199]. These game models incorporate two key elements not found in strategic games: first, they consider games played over a period of time, with players alternating their moves; and second, they make use of *information sets* to explicitly capture the information available to players as they make their moves—in particular, they allow us to specify what each player knows about the moves previously made in the game.

Extensive form games are of particular interest in computer science because they resemble labelled transition systems, or Kripke structures, which are widely used to model concurrently executing computer programs (see, e.g., [13]). A labelled transition system is a directed graph, in which edges correspond to the execution of atomic program instructions, and nodes correspond to states of the concurrent system: executing an instruction by a process within the system causes a transition from one state to another. Edges are labelled, typically either with the instruction being executed, or with the name of the process executing the instruction. However, it is important to note that although extensive games and concurrent programs appear to bear a close resemblance to one another, extensive form games are not intended to model concurrent programs, and have substantial limitations if we aim to use them for this purpose. To better understand this point, let us now describe our main interest, which is to reason about the behaviour of game-like concurrent computer programs, or multi-agent systems.

The key idea in the multi-agent systems domain is that we construct computer programs—software agents—that will act on our behalf in strategic settings. The strategic settings under consideration are usually assumed to involve interaction with other software agents, acting on behalf of other individuals, with preferences that may not be aligned with our own. Thus, in multi-agent systems research, the strategies selected by players are in fact computer programs, which are then executed concurrently with one another; the result of this concurrent computation is the "outcome" of the strategic scenario.

Observe that this view permits us to distinguish between the selection of a strategy (i.e., a program that will act on our behalf in the strategic scenario) and the "run-time behaviour" of the programs that are selected by players to act on their behalf. While game theory is concerned with how to make choices in strategic settings, it is not concerned with modelling and understanding the behaviour of concurrently

executing programs—this is, however, an important topic in computer science research, and seems an important issue if we are interested in understanding the behaviour of multi-agent systems (i.e., how those systems will interact with and transform their environment as they operate on our behalf). Now, we might naturally think of using extensive form games to model the run-time behaviour of agents, but from the perspective of concurrency, extensive form games have an important limitation: they assume an *ordering* of events in a game. In concurrency theory, this assumption corresponds to what is known as *interleaving semantics*, because the actions of each agent are interleaved with one another to generate a run-time behaviour. While for many purposes interleaving semantics are adequate, they are nevertheless accepted to be a simplification of *true concurrency*, where events are *partially ordered*. Crudely, partial-order models of concurrency allow us to explicitly represent the fact that events are, or are not, independent. Now, we can always "flatten" a partial-order model of events into a sequential/interleaved model (or, in game-theoretic terms, an extensive form game), by picking a total order of events that is consistent with the partial order. However, any such flattening operation will represent a simplification of the partial order model, which may imply an ordering of events that is not present in the partial-order model. One might consider the complete set of total orders that are consistent with the partial order, but such a set represents the partial order *implicitly*, while a partial order representation captures it *explicitly*.

In this sense, partial-order models provide a semantically more faithful representation of concurrency and, therefore, allows one to study concurrency at a more fundamental semantic level. Partial-order semantics are, however, more complex than interleaving ones. In fact, any nondeterministic sequential process or concurrent system with an interleaving semantics can be understood as a special case of the partial-order model. Then, one may expect to pay a price for having such a greater amount of information in partial-order models, and indeed, several decision problems that are tractable (to some degree) in the interleaving world can become rather difficult, or even algorithmically unsolvable, in a partial order setting. For example, a system with $n$ parallel $k$-state automata explodes to a $k^n$-state automaton when it is converted into an interleaving model, making most verification problems exponential in the original system size (although by working directly on the concurrent model, some problems can be solved in polynomial time [16]). Even more dramatically, a natural (albeit very strong) notion of equivalence on partial-order models, hereditary history-preserving bisimulation, is undecidable even on the class of finite models [23].

In this paper, we follow a different modelling approach where we explicitly consider a dependence relation (instead of an independence relation) to be able to express that the execution of different events in a given system can be done in an independent way—that is, whenever, they are not related by the dependence relation on events. Because we follow this approach to model dependence and independence in our model, an *empty* dependency relation will represent full concurrency/independence rather than sequential behaviour. The model we present, *partial-order Boolean games*, builds on the increasingly popular Boolean games model [20, 35, 17, 12, 27]. A Boolean game is a game of simultaneous moves, in which each player exercises unique control over a finite set of Boolean variables; the actions available to a player corresponds to the set of all possible assignments of truth or falsity that can be made for these variables. In Boolean games, preferences arise from the fact that each player has a goal, specified as a formula of propositional logic over the total set of Boolean variables in the game. When all players have made their choices (i.e., selected an assignment of truth or falsity to the variables they control), then the result is an overall truth assignment for the variables in the game, which will either satisfy or fail to satisfy each player's goal. A player will prefer outcomes that satisfy their goal over outcomes that do not satisfy it, but is indifferent between two outcomes that satisfy their goal, and is indifferent between outcomes that do not satisfy it.

Boolean games are an important model for multi-agent systems research for several reasons:

- First, strategies for players in Boolean games can naturally be understood as non-deterministic computer programs. All a strategy in a Boolean game does is assign values to a collection of Boolean variables—and this is, of course, exactly what computer programs do as they execute.
- Second, the goals that players have can be understood as the *specification* of the program: the use of logical specifications for the desirable behaviour of computer programs is standard in computer science (see, e.g., [26]) and in AI planning research [15].

However, as they are conventionally formulated, Boolean games are strategic games, and hence games of simultaneous moves, which poses a number of limitations from a semantic point of view. Now, it would

be technically straightforward to adapt the basic Boolean-games model to extensive form, but this would result in an overly simplified model of concurrent behaviour, for the reasons set out above.

We address these issues in partial-order Boolean games by enriching the basic Boolean-games model with a directed acyclic graph over the set of variables in the game, which we refer to as the game's *dependency graph*. This dependency graph captures both the information flow between events, and explicitly represents concurrency in the game. In partial-order Boolean games, when we say that variable $p$ is dependent on variable $q$, then this means that:

1. The strategy for the player controlling variable $p$ can be informed by the value that was chosen for $q$.
2. As a consequence of (1), the assignment of a value to $q$ must precede the assignment of a value for $p$.
3. Finally, the fact that two variables are *not* dependent upon each other means that the assignment of values to these variables may take place *concurrently*.

*Related (and unrelated) work.*Using interleaving or partial-order semantics in games can have important consequences—both from theoretical and practical viewpoints—as making such a choice can raise different issues, some of which we investigate here.

More specifically, we will present examples and complexity results which show that games played directly on a structure with an explicit notion of (in)dependence are radically different. For instance, checking their equilibrium properties is a computationally more complex problem (e.g., checking the existence of pure Nash equilibria is NEXPTIME-complete for partial-order Boolean games and only in $\Sigma_2^{\mathrm{P}}$ for conventional Boolean games); more importantly, disregarding the relations in the dependency graph of a game can lead to the construction of games—and hence of systems—with different sets of equilibrium strategy profiles.

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* [6]. 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) [24]. 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.

## 3 Partial-Order Boolean Games

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 $\Psi$, 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 $\Psi$, a *$\Psi$-valuation* is a total function $v\colon \Psi \rightarrow \mathbb{B}$, assigning truth or falsity to the variables in $\Psi$; we let $\mathcal{V}(\Psi)$ denote the set of all $\Psi$-valuations. Given a formula $\varphi \in \mathcal{L}_\Psi$ and a $\Psi$-valuation $v\colon \Psi \rightarrow \mathbb{B}$, we write $v \models \varphi$ to mean that $\varphi$ is true under the $\Psi$-valuation $v$, where the satisfaction relation "$\models$" is defined in the conventional way for propositional logic (so, for example, $v \models \varphi \vee \psi$ iff either $v \models \varphi$ or $v \models \psi$ or both).

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 $\gamma_i$ that $i$ desires to have satisfied. Each player $i \in N$ *controls* a (possibly empty) subset $\Phi_i$ of the overall set of Boolean variables $\Phi$. 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, 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.

Partial-order Boolean games augment conventional Boolean games with *dependency graphs*. Formally, a dependency graph over a set $\Phi$ 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 $p$.

Notice that we do *not* require dependency graphs to be transitive. We denote the transitive closure of $D$ by $D^+$, and where $q \in \Phi$, we define $D[q] = \{\, p \in \Phi \mid D(p,q)\}$, and $D^+[q]$ similarly. Thus, for any variable $p$, $D[p]$ is the total set of variables whose value can be taken into consideration when choosing a value for $p$.

The motivation for the requirement that dependency graphs are acyclic should be immediately obvious: 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 independently executing concurrent processes. The use of the dependency graph makes independence explicit and transparent (cf. [29]).

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 $\Phi$.

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 $v_i \colon \Phi_i \to \mathbb{B}$ for their Boolean variables $\Phi_i$. 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

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

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

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

(Recall that $\mathcal{V}(D[p])$ is the set of valuations over the variables in $D[p]$; such a function tells the relevant player what value to assign to $p$ given the values of the variables upon which $p$ depends.)

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 $\varphi$ 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 $p = \top$ or $p = \bot$. This is because, if $D[p] = \emptyset$ then $p$ does not depend on any variable. Hence, the choice of a value for this variable is equivalent to simply assigning directly either $\top$ or $\bot$ to $p$.

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 $k$ [7].

A *strategy* for player $i$, denoted by $\sigma_i$, is then a set of choice equations, one for each variable controlled by $i$. As usual, a *strategy profile* $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ 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:

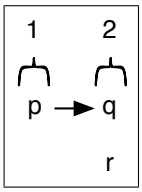


**Fig. 1** Dependency graph for Example 1.

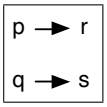


**Fig. 2** Dependency graph for Example 2.

**Lemma 1** *Let $G = \langle N, \Phi, (\Phi_i)_{i\in N}, (\gamma_i)_{i\in N}, D\rangle$ be a game and let $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ 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 $v(\boldsymbol{\sigma})$, (and thus we have that $v(\boldsymbol{\sigma})\colon \Phi \to \mathbb{B}$), and which we refer to as the* valuation induced by $\boldsymbol{\sigma}$. *Moreover, $v(\boldsymbol{\sigma})$ can be computed in time polynomial in the size of $\boldsymbol{\sigma}$.*

*Proof* Let $\sigma = \sigma_1 \cup \cdots \cup \sigma_n$. Now, consider any ordering of the equations in $\sigma$ which satisfies the requirement that all variables appearing in the right hand side of an equation must have previously appeared on the left hand side of an equation. The requirement that $D$ is acyclic ensures that such an ordering exists (although it will not in general be unique). Then such an ordering defines a simple straight line program (in fact, a greatly simplified form of straight line Boolean program [11]), which can be executed in polynomial time; and the resulting assignments to variables in $\Phi$ is the valuation $v(\boldsymbol{\sigma})$. Notice that *any* ordering of the equations satisfying the above requirement will yield the same valuation. □

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

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

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

Let us consider an informal 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. In particular, in the following example, player 2 will represent the captain, and variables $q, r$ will play the roles of the two platoons.

*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 given 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 $\gamma_2$, there is no strategy that guarantees to make both conjuncts true simultaneously. This is because $r$ is independent of $p$ and $q$: a choice for this value must be made independently of $p$ and $q$. As can be seen, the only two choice equations available for variable $r$ are as follows: $r = \top$ and $r = \bot$.

Note that since player 1 always gets its goal achieved, player 1 is happy (and free) to choose any value for $p$, and so player 2 cannot rely on the selection of any particular values for $p$ and $q$ to satisfy $\gamma_2$.

In summary, player 2 can choose the strategies that will select values for the variables $q$ and $r$ that he or she controls; but because these variables are independent, the strategies that choose values for $q$ and $r$ cannot communicate or coordinate with each other while they are being executed.

Note that 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 illustrates why not.

*Example 2* Suppose we have a game with:

$N = \{1, 2\}$,
$\Phi_1 = \{p, q\}$,
$\Phi_2 = \{r, s\}$,
$\gamma_1 = \top$, 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 $s$ takes the opposite value of $q$.

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—call them agents 3 and 4—the former controlling variable $r$ and the latter controlling variable $s$, each new agent having the same goal ($\gamma_2$, above), and with the same dependency graph in Figure 2. Then, the two strategy sets above induce pure 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. □

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:

$N = \{1, 2\}$,

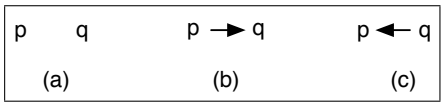


**Fig. 3** The only three possible dependency graphs for Example 3.

$\Phi_1 = \{p\}$,
$\Phi_2 = \{q\}$,
$\gamma_1 = p \leftrightarrow q$, 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 $q$.

We consider each possibility in turn.

- First consider (a). It should be easy to see that there is no pure Nash equilibrium for the game with dependency graph (a). Consider the following strategy profile: $(\sigma_1)\ p = \bot$ and $(\sigma_2)\ q = \bot$.
  With this outcome, $\gamma_1$ is satisfied, while $\gamma_2$ is not. In this case, player 2 could deviate with strategy $q = \top$, resulting in the satisfaction of his 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 pure 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 pure Nash equilibrium: as player 2 has his goal achieved, he cannot beneficially deviate, and no deviation on the part of player 1 could improve her position.
- Finally, 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 $q$, 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 her goal achieved by using the strategy $p = q$.

□

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 $D = \emptyset$, and so $D[p] = \emptyset$ 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 pure Nash equilibria of a game, an arguably desirable property from a game-theoretic perspective.

## 4 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 pure Nash equilibrium. The decision question is stated as follows.

IS-NE:
*Given*: Partial-order Boolean game $G$, strategy profile $\boldsymbol{\sigma}$.
*Question*: Is it the case that $\boldsymbol{\sigma} \in NE(G)$?

We emphasise that the representation of strategies we consider in this problem is formally given as a set of choice equations, as described in the preceding section.

The corresponding problem for conventional Boolean games is co-NP-complete [5], 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 $\boldsymbol{\sigma}$ be a strategy profile for this game. Then, if $i$ has a beneficial deviation from $\boldsymbol{\sigma}$, there is a beneficial deviation $\sigma_i'$ for player $i$ which is of size polynomial in $|\Phi_i|$.*

*Proof* Suppose $i$ has a beneficial deviation from $\boldsymbol{\sigma}$; let $\sigma_i''$ be this deviation. Let $\sigma_i'$ be the following strategy for $i$:

$$\sigma_i' = \{x = v(\boldsymbol{\sigma}_{-i}, \sigma_i'')(x) \mid x \in \Phi_i\}.$$

Clearly $\sigma_i'$ is of size polynomial in $|\Phi_i|$, and since $\sigma_i'$ simply copies whatever $\sigma_i''$ does, $v(\boldsymbol{\sigma}_{-i}, \sigma_i'') = v(\boldsymbol{\sigma}_{-i}, \sigma_i')$; as a consequence, the strategy $\sigma_i'$ is a beneficial deviation for player $i$. □

So, if player $i$ has a beneficial deviation, then there is a *small certificate* to this effect. Expressed in different words, if a player can deviate, then it can deviate using a very simple kind of "uninformed" strategy, which can be represented very succinctly. This is 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.

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 IS-NE problem, Proposition 2 allows us to prove the following complexity result.

**Proposition 1** IS-NE *is co-NP-complete.*

*Proof* Hardness can be demonstrated using a construction that is essentially the same as that used to show co-NP-hardness of the corresponding problem for conventional Boolean games (see, for instance, [35, Proposition 1]). Alternatively, it can also be shown by letting the dependency graph be empty, that is, by considering the special case of conventional Boolean games. To establish that IS-NE is in co-NP, we will show that the complement problem is in NP. The complement problem involves checking whether any player has a beneficial deviation from a given strategy profile. See that, from Lemma 2, we can existentially guess a player $i$ and a "small" deviation $\sigma_i'$ for $i$, and then from Lemma 1, we can check in polynomial time that $v(\boldsymbol{\sigma}) \not\models \gamma_i$ while $v(\boldsymbol{\sigma}_{-i}, \sigma_i') \models \gamma_i$. □

The next problem to consider is whether a given game has any pure 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 $\Sigma_2^{\mathrm{P}}$-complete [5], and the fact that IS-NE 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.*

*Proof* The following NEXPTIME algorithm decides the problem:

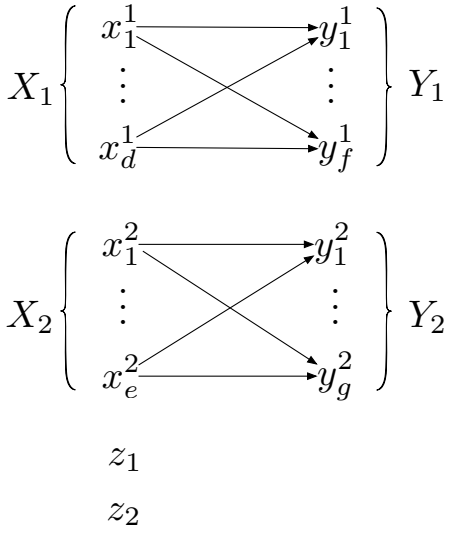


**Fig. 4** Dependency graph for the reduction in Proposition 2 ($X_1 = \{x_1^1, \ldots, x_d^1\}$, $X_2 = \{x_1^2, \ldots, x_e^2\}$, $Y_1 = \{y_1^1, \ldots, y_f^1\}$, and $Y_2 = \{y_1^2, \ldots, y_g^2\}$).

1. guess a strategy profile $\boldsymbol{\sigma}$;
2. verify that $\boldsymbol{\sigma} \in NE(G)$.

Step (1) is in NEXPTIME because any Boolean function on variables $\Phi$ can be represented by a propositional formula of size at most exponential in $|\Phi|$. For instance, for a Boolean function of $k$ arguments, the disjunction of all terms of $k$ literals on which the function is true, makes up such a formula. Guessing a strategy profile thus involves guessing at most $|\Phi|$ such formulae, one for each variable. Step (2) can then be carried out in deterministic time exponential in the number of variables (Lemma 1). The overall algorithm works in NEXPTIME.

For NEXPTIME hardness, we reduce the DEPENDENCY QUANTIFIER BOOLEAN FORMULA GAME (DQBFG) [21, p.87]. This problem relates to a 3 player game, containing players $B$ (Black) and $W_1, W_2$ (White 1, White 2). The white players form a team, attempting to beat the black player. The game is played on a Boolean formula $\varphi$ over variables $X_1 \cup X_2 \cup Y_1 \cup Y_2$. Player $W_i$ sees only variables $X_i \cup Y_i$. The game is played as follows:

- Player $B$ chooses an assignment for the variables $X_1 \cup X_2$;
- Player $W_1$ chooses an assignment for variables $Y_1$;
- Player $W_2$ chooses an assignment for variables $Y_2$;
- If the overall assignment for $X_1 \cup X_2 \cup Y_1 \cup Y_2$ satisfies $\varphi$ then black wins, otherwise team white wins.

The question is then whether there is a winning strategy for team white in this game. Our reduction produces a game with 3 players, $N = \{B, W_1, W_2\}$; as might be guessed, each player corresponds to the player with the same name in the DQBFG instance. We let $\Phi = X_1 \cup X_2 \cup Y_1 \cup Y_2 \cup \{z_1, z_2\}$ where $\{z_1, z_2\}$ are new variables. Goals and controlled variables for each player are defined as follows:

- $\gamma_B = \varphi$; $\Phi_B = X_1 \cup X_2$;
- $\gamma_{W_1} = (\neg\varphi) \vee (z_1 \leftrightarrow z_2)$; $\Phi_{W_1} = Y_1 \cup \{z_1\}$;
- $\gamma_{W_2} = (\neg\varphi) \vee \neg(z_1 \leftrightarrow z_2)$; $\Phi_{W_2} = Y_2 \cup \{z_2\}$.

The dependency graph for the game is as shown in Figure 4. Thus for each $y_i \in Y_i$ we have $D[y_i] = X_i$.

Now, let $G$ be the partial-order Boolean game defined by this reduction. We claim that team white has a win in the given DQBFG instance if and only if $NE(G) \neq \emptyset$.

($\Rightarrow$) In order to show this direction, suppose that team white has a win in the DQBFG instance. Consider the winning strategies for the DQBFG instance: they immediately define strategies for players $W_1$ and $W_2$ in

our game (observe that the dependency graph ensures that the information dependencies are satisfied). We claim that any strategy profile containing these strategies will be a pure Nash equilibrium. Since they are winning strategies for team white, they ensure that $\neg\varphi$ is satisfied, and therefore that players $W_1$ and $W_2$ have their goals achieved. The only possible deviation would come from player $B$. But since the strategies are winning for team white, then player $B$ can have no such possible beneficial deviation.

($\Leftarrow$) In order to show this direction, suppose that $NE(G) \neq \emptyset$ and consider an arbitrary strategy profile $\boldsymbol{\sigma} \in NE(G)$. We claim that $\boldsymbol{\sigma}$ defines a winning strategy for team white in the DQBFG instance. Because $\boldsymbol{\sigma}$ is a pure Nash equilibrium no player wants to deviate. In particular, this means that $\neg\varphi$ must be satisfied as otherwise either $W_1$ or $W_2$ would have a beneficial deviation, no matter the values of $z_1$ and $z_2$. Moreover, because $\boldsymbol{\sigma} \in NE(G)$, player $B$ cannot deviate either. This immediately implies that the strategies that $W_1$ and $W_2$ are using define a winning strategy for team white in the DQBFG instance.

□

Now, in light of Lemma 2, this result may seem surprising. However, there is a simple reason for the substantial blowup in complexity. Whereas in the former one can replace potential deviations 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 pure Nash equilibrium:

$$(\sigma_1)\ p = \bot$$
$$(\sigma_2)\ q = \neg p.$$

This is because, using $\sigma_2$, 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:

$$(\sigma_1)\ p = \bot$$
$$(\sigma_2')\ q = \top.$$

However, this strategy profile does *not* form a pure Nash equilibrium: player 1 could now beneficially deviate using $p = \top$. In fact, there is no pure Nash equilibrium of this game in which both players use strategies of the form $x = b$, where $b$ is a logical constant. □

### 4.1 Special Cases

Even though the complexity of NON-EMPTINESS is NEXPTIME-complete with respect to 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 $\Sigma_2^{\mathrm{P}}$-complete.*
2. *For partial-order Boolean games of the following form,* NON-EMPTINESS *is PSPACE-complete:*
   *We have $N = \{1, 2\}$ with $\Phi_1 = \{x_1, \ldots, x_k\}$ and $\Phi_2 = \{y_1, \ldots, y_k\}$. The dependency graph is defined by $D[x_i] = \{x_j, y_j \mid 1 \leq j < i\}$ and by $D[y_i] = \{x_i\} \cup \{x_j, y_j \mid 1 \leq j < i\}$.*

Observe that in case (1), we obtain conventional Boolean games, for which the NON-EMPTINESS problem is $\Sigma_2^{\mathrm{P}}$-complete. The second case essentially corresponds to QBF, which is PSPACE-complete. More informally, it corresponds to a sequential setting, a prominent special case that motivates the definition of such a dependency graph. Because the complexity in the general case, where concurrency is allowed, is NEXPTIME, and in the sequential case it goes down to PSPACE, it becomes clear that concurrent behaviour in our setting does make things computationally harder.

Another interesting case, which closely related to the behaviour that is usually seen in concurrent and distributed systems, is the one where a group of agents collaborate in order to achieve a common goal. Such a situation is naturally captured by a solution concept called *strong Nash equilibrium* where multiple players are allowed to deviate—and not only one as it is the case in the definition of pure Nash equilibria. Formally, we say that $\boldsymbol{\sigma}$ is a *strong Nash equilibrium* if there is no coalition $C \subseteq N$ satisfying that for every player $i \in C$ there is a strategy $\sigma_i'$ for player $i$, such that $u_i(\boldsymbol{\sigma}_{-C}, \sigma_C') > u_i(\boldsymbol{\sigma})$, where $\sigma_C'$ is the collection of strategies $\sigma_i'$ for all players $i \in C$. Thus, the strategy profile $\boldsymbol{\sigma}$ is a strong Nash equilibrium if no coalition of players could benefit by jointly deviating from $\boldsymbol{\sigma}$ to another collection of strategies $\sigma_C'$. Let $sNE(G)$ denote the set of strong Nash equilibria of $G$. Based on this definition, we can show that checking whether a strategy profile $\boldsymbol{\sigma}$ is a strong Nash equilibrium can be done in exponential time:

**Lemma 3** *Let $G$ be a partial-order Boolean game and $\boldsymbol{\sigma}$ be a strategy profile. Then, checking whether $\boldsymbol{\sigma} \in sNE(G)$ can be done in deterministic exponential time in the number of both the Boolean variables and players in the game.*

*Proof* The algorithm to check whether $\boldsymbol{\sigma} \in sNE(G)$ can be divided in two sub-steps, and solved in deterministic exponential time. First, using $\boldsymbol{\sigma}$, we can compute the set of players $L \subseteq N$ who do not get their goals achieved. This step of the algorithm can be done, for each player in the game, in deterministic exponential time in the number of variables—see the case for pure Nash equilibrium. We then check if any subset of players $C \subseteq L$ have a beneficial deviation. This can be done in deterministic exponential time in the number of players in the game: for each $C \subseteq L$, we check whether there is a strategy profile for $C$ such that using this strategy profile, with other players using their component of $\boldsymbol{\sigma}$ would result in the achievement of all the goals of players in $C$. The overall algorithm therefore runs in deterministic exponential time in both the number of Boolean variables and the number of players in the game. □

Using Lemma 3 we can, in addition, show that NON-EMPTINESS for partial-order Boolean games and strong Nash equilibrium is an NEXPTIME-complete problem, thus no harder than pure Nash equilibrium.

**Proposition 4** NON-EMPTINESS *for strong Nash equilibrium is NEXPTIME-complete.*

*Proof* The proof, and algorithm, is very similar to the case for pure Nash equilibrium. We use the same meta-algorithm, which decides the problem in NEXPTIME:

1. guess a strategy profile $\boldsymbol{\sigma}$;
2. verify that $\boldsymbol{\sigma} \in sNE(G)$.

As before, step (1) is in NEXPTIME because any Boolean function can be represented by a propositional formula of size at most exponential in $|\Phi|$. Moreover, step (2) can then be carried out in deterministic time exponential in both the number of variables and the number of players (Lemma 3). Then, the overall algorithm works in NEXPTIME. For hardness, observe that the reduction from DQBFG games used in Proposition 2 can be used here as well.[1] That is, it is also true that team white has a win in the given DQBFG instance if and only if $sNE(G) \neq \emptyset$. For the ($\Rightarrow$) direction note that the same reasoning applies because in the game $G$ only player $B$ does not have its goal achieved. Then, the only coalition of players who do not get their goal achieved is the *singleton* set $\{B\}$. On the other hand, for the ($\Leftarrow$) direction, note that if $\boldsymbol{\sigma} \in sNE(G)$ then, in particular, it is also the case that $\boldsymbol{\sigma} \in NE(G)$, and again the same reasoning can be used to show that team white has a winning strategy in the DQBFG instance. □

Another special case to be formally analysed is the one illustrated in Example 3, that is, informally, the case where a dependency graph is included in a more general one. As we show next, in such a case, the set of pure Nash equilibria of the system can only be made bigger. Formally, we have the following result.

**Proposition 5** *Let $G = \langle N, \Phi, (\Phi_i)_{i\in N}, (\gamma_i)_{i\in N}, D\rangle$ and $G' = \langle N, \Phi, (\Phi_i)_{i\in N}, (\gamma_i)_{i\in N}, D'\rangle$ be two games differing only in $D$, such that $D \subseteq D'$. Then*

$$NE(G) \subseteq NE(G').$$

[1] A key observation is that in our partial-order Boolean game every strategy profile always renders two winners and only one loser.

*Proof* Suppose $\boldsymbol{\sigma} \in NE(G)$. Since $D \subseteq D'$ all players who achieved their goals in $G$ can use the same strategies in $G'$. Then, the only players who will have an incentive to deviate are those whose goals were not achieved in $G$.

Suppose for a contradiction that player $i$ does not get is goal achieved in $G$, but can, in $G'$, unilaterally deviate and achieve its goal $\gamma_i$. If that is the case, then, due to (the proof of) Lemma 2, there is a strategy $\sigma_i'$ for $i$, which does not depend on $D'$, that is, a strategy that can be defined regardless of the dependencies in $D'$, and which is such that $(\boldsymbol{\sigma}_{-i}, \sigma_i') \models \gamma_i$. However, since $\sigma_i'$ depends neither on $D'$ nor on $D$, it is a strategy that $i$ could also use in $D$ to unilaterally deviate and get its goal $\gamma_i$ achieved. However, since $\boldsymbol{\sigma}$ is a pure Nash equilibrium, such a strategy $\sigma_i'$ cannot exist, and hence if $\boldsymbol{\sigma} \in NE(G)$ then $\boldsymbol{\sigma} \in NE(G')$ too, which is a sufficient condition to show that $NE(G) \subseteq NE(G')$. □

The strategy $\sigma_i'$ used in the proof of Lemma 2 and in the proof of the above proposition is a particular kind of "uninformed" strategy, so called because it does not depend on what the other players in the game may do, but rather on a fixed valuation for the variables controlled by player $i$. Note also that as illustrated in Example 3 the inclusion in the other direction may not hold in the general case. As a consequence, adding functional dependencies to the game can only either increase the number of pure Nash equilibria of the game or leave it unchanged.

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.

## 5 Concurrency in Partial-Order Boolean Games

As we noted earlier, in concurrency theory there are two main approaches to modelling concurrent behaviour: either using *interleaving* or using *partial-order* models for concurrency. Interleaving models represent concurrency as the nondeterministic combination of all possible sequential behaviours in the system. On the other hand, partial-order models represent concurrency explicitly, by means of an (in)dependence relation on the set of events in the system that can be executed concurrently.

Examples of interleaving models include Kripke structures, labelled transitions systems, infinite trees, Hoare languages, Moore and Mealy machines, and many more. Examples of partial-order models include Petri nets, event structures, asynchronous transition systems, Mazurkiewicz trace languages, Chu spaces, transition systems with independence (TSI), amongst others. A good introduction to various models for concurrency, both to interleaving and to partial-order ones, can be found in [29].

Because partial-order semantics represent concurrency explicitly (by means of an independence relation on the set of events that the system can execute in parallel), independence/concurrency is a *primitive* notion rather than a *derived* concept as is the case in the interleaving framework. In this sense, partial-order models provide a semantically more faithful representation of concurrency and, therefore, allows one to study concurrency at a more fundamental semantic level. In fact, any nondeterministic sequential process or concurrent system with an interleaving semantics can be seen, trivially, as a partial-order model with an empty independence relation.

However, because partial-order models may not be easy to deal with, much research has been done to find restricted, yet interesting and useful, classes of models where different aspects of concurrency can be captured. An approach that has delivered positive results from a semantic viewpoint is the one where *closure operators* are used to model *deterministic concurrent* behaviour. Due to its mathematical properties, this approach has found interesting applications and interpretations within games—cf., [1, 3, 18, 34]. In particular, in [34], strategies represented as closure operators are formally related to event structures [29], the canonical model of concurrency that was the inspiration for our model of Boolean games.

In this section, we study the connections between our model of games and event structures. We do so by investigating the relationship between the model of strategies associated with partial-order Boolean games and the way that *deterministic concurrency* can be captured using closure operators. In particular, we show that concurrency, as seen in partial-order Boolean games, can be mathematically represented using a closure operator semantics for players' strategies. This closure-operator semantics, in turn, gives a more explicit representation of the causal and temporal dependencies in our games framework.

Even though the temporal order of events in partial-order Boolean games is not completely determined by the strategies executed by the players (since *independent* events may occur in *parallel*, while *dependent*

events must occur *sequentially*), a given strategy profile completely determines the final outcome the game. This kind of deterministic, yet concurrent, behaviour can be mathematically represented by a closure-operator semantics of the game, which we formalise next.

Let $\leq$ denote the reflexive and transitive closure of $D$. Since $D$ is a DAG, the structure $(\Phi, \leq)$ is a partial order. Let $\mathcal{D}$ be the set of $\leq$-downclosed subsets of $\Phi$. Thus $(\mathcal{D}, \subseteq)$ is a partial order. Moreover, for each player $i$, let $D_{-i}[q]$, with $q \in \Phi_i$, be defined to be the set $(D^+[q] \setminus \Phi_i)$. Now, for each player $i$, define the function $\pi_i \colon \mathcal{D} \to \mathcal{D}$ by

$$\pi_i(X) = X \cup \{q \in \Phi_i \mid (D_{-i}[q]) \subseteq X\}.$$

The function $\pi_i$ adds to $X$ the variables to be played next by player $i$, that is, it adds those variables in $\Phi_i$ that can be given values provided that the variables in $X$ have been already set to some Boolean value. Note that $\pi_i(X)$ is also $\leq$-downclosed. The key observation in this section is that, in fact, over the partially ordered set $(\mathcal{D}, \subseteq)$, each function $\pi_i$ determines a stable closure operator, that is, a map being:

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

We then can show:

**Proposition 6** *Let $D$ be the dependency graph in a partial-order Boolean game where $N = \{1, \ldots, n\}$ and $\Phi = (\Phi_i)_{i \in N}$. Then, each function $\pi_i$ for every player $i$, as defined above, is a closure operator.*

*Proof* To show that each $\pi_i$ is a closure operator, we need to show that $\pi_i$ is extensive, idempotent, and monotone.

**(Extensive)** Follows directly from the definition.

**(Idempotent)** Take any $X \in \mathcal{D}$ and let $Y = \pi_i(X)$. Then, $Y = X \cup \{q \in \Phi_i \mid (D_{-i}[q]) \subseteq X\}$ and, since $(Y \setminus X) \subseteq \Phi_i$, we know that $D_{-i}[q] \subseteq X$ if and only if $D_{-i}[q] \subseteq Y$, for all $q$. Therefore,

$$\begin{aligned} \pi_i(X) &= X \cup \{q \in \Phi_i \mid (D_{-i}[q]) \subseteq X\} \\ &= Y \cup \{q \in \Phi_i \mid (D_{-i}[q]) \subseteq X\} \\ &= Y \cup \{q \in \Phi_i \mid (D_{-i}[q]) \subseteq Y\} \\ &= \pi_i(Y) \end{aligned}$$

Therefore, $\pi_i(X) = \pi_i(\pi_i(X))$, to obtain:

$$\forall X.\ \pi_i(X) = \pi_i(\pi_i(X)).$$

**(Monotone)** Take any $X, Y \in \mathcal{D}$ and let $X \subseteq Y$. By definition, we have

$$\pi_i(X) = X \cup \{q \in \Phi_i \mid (D_{-i}[q]) \subseteq X\}$$

and

$$\pi_i(Y) = Y \cup \{q \in \Phi_i \mid (D_{-i}[q]) \subseteq Y\}.$$

Since $X \subseteq Y$ then, for all $q' \in \Phi_i$,

$$\begin{aligned} &\text{if} \quad q' \in \{q \in \Phi_i \mid (D_{-i}[q]) \subseteq X\} \\ &\text{then } q' \in \{q \in \Phi_i \mid (D_{-i}[q]) \subseteq Y\} \end{aligned}$$

to obtain that

$$\forall X, Y.\ X \subseteq Y \Rightarrow \pi_i(X) \subseteq \pi_i(Y)$$

—as required. □

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.

Call $X \in \mathcal{D}$ a *history* of play: $X$ 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 $\tau_i$, defined below) based on both $\pi_i$ and the choice equations, denoted by $f_q$, for the variables $q \in \Phi_i$ they have control over, in the following way:

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

The first observation to make is that the original semantics of partial-order Boolean games, that is, the semantics given by $(\sigma_i)_{i\in N}$ representing sets of choice equations for each player $i$, can be easily recovered from $(\tau_i)_{i\in N}$, as follows for each player strategy: $\sigma_i = \bigcup_{p\in\Phi_i} \tau_i(D^+[p])$. The second observation is that even though $\tau_i$ is a strategy for player $i$, semantically, the map $\tau_i$ behaves differently from $\sigma_i$. The former takes histories of play and, based on that information, gives only a few necessary choice functions to be used next (a rather operational approach); the latter, instead, provides a player with all the choice equations that $i$ has available at once, with no information as to when during the game such equations should be used (a more denotational approach). Thus, we can think of $\tau_i$ as being more operationally informative when compared with $\sigma_i$, since some notion of "time" is implicitly considered. (However, in either case, that is whether using $(\sigma_i)_{i\in N}$ or $(\tau_i)_{i\in N}$, one needs to have access to the dependency graph to know when a particular choice function is going to be effectively used while playing the game.) Because of the informal descriptions given above, we will say that the map $\tau_i$ is a *temporal strategy* for player $i$. In fact, because each $\pi_i$ is a stable closure operator, a few useful additional observations also follow.

1. First, a temporal 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.
2. Second, the behaviour of different agents is independent on concurrent variables, i.e., such a kind of behaviour is oblivious to superfluous or unnecessary alternations between players' moves. Put another way, in the end, it is irrelevant who plays first in concurrent (unordered) events—of course this is not the case for sequential (ordered) events.
3. Finally, as dependency graphs induce a temporal (and causal) dependency relation on events, playing at a later "time unit" provides at least as much information as playing at a previous one; indeed, a temporal strategy preserves causal dependencies when playing the game.

Not all the above properties relate to concurrency. The first one simply ensures that a temporal strategy behaves as expected: providing a player with the choice decisions that such a player has to make give a particular state of play. The second one, on the other hand, is the most closely related to concepts in concurrency. Because in a concurrent setting independent (unordered) events should be treated *uniformly* regardless of the particular order in which they happen to be played, a strategy must formally capture such a uniformity requirement. Closure operators, and in particular the second property above, ensure exactly that. This, in turn, helps ensure that strategies can be composed in *any* order you like and the final result of the game would be the same; in other words, the problem of how to compose strategies in a concurrent setting vanishes once a closure-operator interpretation can be given. Extensive studies of this phenomena can be found, for instance, in [33] for programming languages and in [1] for logical systems. Finally, the third property above mentioned embodies an *efficiency* property from a concurrent viewpoint: it ensures that no player will be allowed to play fewer events than those it is required to. As with the second property, this property also relieves one from having to define how to interpret the execution of multiple events. For a more comprehensive discussion on the natural relationship between closure-operator semantics and concurrency, the reader is referred to [1,34], upon which our presentation builds.

One delicate point that should mentioned concerns dependencies between a player's own variables. The definition of $D_{-i}$ means that such dependencies are ignored in the computation of $\tau_i$. Thus in setting variables according to the choice functions in $\tau_i(X)$, it is still necessary to resolve any player-internal dependencies to constrain the execution order. However, this process is invisible to other players. In general, using player-internal dependencies can have rather counter-intuitive results, imposing a form of imperfect recall; in most modelling situations, there should be no internal dependencies.

As mentioned above, the idea of using closure operators to model concurrency has been successfully used in the past, e.g., see [1,3,18,33,34]. However, it is important to note that the closure-operators representation can only deal with deterministic concurrency. In a non-deterministic setting, more care has to be taken in order to ensure that the particular execution of unordered (concurrent, independent, etc.) events does not affect the outcome of the game. Deterministic concurrency—and therefore closure-operator semantics—is nevertheless a modelling framework powerful enough to represent the concurrent behaviour of many interesting classes of systems. To conclude this section, let us now see, by way of a simple example, what 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 $\tau_1$ and $\tau_2$ 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 with:

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

where, as defined above for any $\tau_i$, for each pair in $\tau_i$, the first component of such a pair is the history of play (that is, a set of Boolean variables) and the second component is what player $i$ is to play next given such a particular history of play (that is, a set of choice functions). □

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.

## 6 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, it 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 [31] and logics for independence and concurrency [8].

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).

Because of this way that IF formulae are interpreted, a particular IF formula can be either true, false, or undetermined. The first two cases have the same game-theoretic interpretation of first-order logic: using the standard evaluation game of a given formula $\varphi$ (with respect to a given model $M$), we would say that $\varphi$ is true if there is a winning strategy to show that $\varphi$ holds in $M$, whereas we would say that $\varphi$ is false if there

is a winning strategy to show that $\varphi$ does not hold in $M$. However, since in general in IF evaluation games winning strategies may not exist, an IF formula $\varphi$ will be defined to be undetermined if it is neither true nor false. Then, for instance, as shown by Proposition 7 below, the semantics of the formula shown above can by given by a partial-order Boolean game with $D = \{(x, y), (u, v)\}$. In general, these dependencies define a partial order which, under our semantics, is interpreted with the dependency graph $D$.

*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 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 7** *Let $\varphi = \forall \ldots \psi$, where $\psi$ 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 $\varphi$. Then*

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

*Proof* First, note that if $\varphi$ is IF-true, by the definition of the semantics of the logic, the pure Nash equilibria necessarily contain winning $\exists$ strategies, making $\psi$ true. Conversely, if a pure Nash equilibrium exists in which $\psi$ is true, the strategy in it is IF-winning, so $\varphi$ is IF-true. The case where $\varphi$ is IF-false is similar. Otherwise, either (i) no pure Nash equilibrium exists or (ii) $\varphi$ is not either true or false in all pure Nash equilibria of $G$. In either case, (i) or (ii), it implies that no player has a winning strategy to ensure that $\varphi$ is true/false regardless of the behaviour of the other player—hence, necessarily, the formula is IF-undetermined. □

Partial-order Boolean games may have many players, not merely two. In coalition logics [31], 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 $\Phi_1, \Phi_2$ and arbitrary goals $\gamma_1, \gamma_2$. In the case that the $\psi_i$ 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 $\psi_i$ 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 $\psi$ may contain conjunctions $\psi_1 \wedge \psi_2$, where $\forall$ player chooses the conjunct, acting as 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 illustrated next:

*Example 7* Let player 1 own variable $p$ and player 2 own variable $q$, with empty dependency and let both players have goal $p \wedge q$. 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 $\gamma_1 \to \gamma_2$, is more interesting. If player 1 can win, then player 2 wins also; but player 2 may try to win while falsifying $\gamma_1$. A pair of strategies that achieves $\gamma_1$ is in equilibrium; a pair of strategies that achieves $\gamma_2 \wedge \neg\gamma_1$ is only in equilibrium if there is no way for 1 to achieve $\gamma_1$. 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.

### 6.1 'Socially responsive' logics

In Section 5, we outlined a closure operator approach for *n*-player concurrent games; and, in this section, we encoded IF logic into our games. However, the Hintikka game understanding of IF logic is still a two player game—only the team approach makes some use of the many agents.

Abramsky [2] discusses a logic (call it $\mathcal{AL}$) generalizing the ideas of IF logic to the multi-player case. Given a set of *n* agents, he proposed using logical operators $\oplus_i$ and $\mathcal{Q}_i$, where $\varphi \oplus_i \psi$ represents agent *i* making a choice between playing in $\varphi$ or $\psi$, and $\mathcal{Q}_i x.\varphi$ represents agent *i* choosing a value for variable *x*. Thus, in the two-player case, $\oplus_1, \oplus_2, \mathcal{Q}_1, \mathcal{Q}_2$ would correspond to $\vee, \wedge, \exists, \forall$ in the normal FOL game. Independence of choices is achieved by explicit $\varphi \| \psi$ (parallel evaluation) and $\varphi.\psi$ (sequential evaluation) operations—and one can then sensibly view $\mathcal{Q}_i x$ as a 'particle' (a free-standing formula), rather than as a prefixing operator. Thus, for example, the IF formula $\forall x.\exists y/x$ corresponds to $(\mathcal{Q}_1 x)\|(\mathcal{Q}_2 y)$ (as also done in [8]). As a consequence, in the general *n*-agent case, quantifier particles can be combined with $\|$ and . to describe the dependency of choices, exactly as we did in the previous section (see example 5)—and indeed, Abramsky [2] also gives a semantics to its logic in terms of closure operators.

The $\oplus$ operators raise an interesting question about the formulation of partial-order Boolean games. In our formulation, the agents make a set of partially dependent choices of Boolean variables, aiming to achieve goals $\gamma_i$ expressed as Boolean formulas over the variables. These goals are evaluated purely truth-functionally at the end of the game. However, evaluation of a Boolean formula is also expressible in the standard way as a game between two players, 'verifier' and 'refuter', with verifier making $\vee$ choices and refuter the $\wedge$ choices, and so we can continue the original game into a second stage in which player *i* verifies that it has achieved its goal. Of course, the system has no agent corresponding to the 'refuter' in the verification game, but we can encode 'refuter of *i*' as an additional agent $\bar{\imath}$, with control of no variables, whose goal is $\neg\gamma_i$. This then allows the Boolean operators $\vee$ and $\wedge$ in $\gamma_i$ to be expressed as $\oplus_i$ and $\oplus_{\bar{\imath}}$, so integrating the goals into the $\mathcal{AL}$ formulation.

Such an $\mathcal{AL}$ style presentation allows a more refined view of shared and partially shared goals. Suppose there are agents 1 and 2, controlling independent variables *p* and *q* respectively, with goals $\gamma_1 = \gamma_2 = p \vee q$. An obvious pure Nash equilibrium strategy profile is the one where each player sets its variable true. Now, suppose instead that $\gamma_1 = p \vee \neg q$, and $\gamma_2 = \neg p \vee q$. There are two pure Nash equilibrium strategy profiles: one where both set true, and one where both set false. The latter strategy profile is interesting: although it is an equilibrium, with the same outcome as the 'both true' profile, it is clearly not one that a 'rational' player would adopt, as it relies on the other agent cooperating, whereas the 'both true' profile achieves the goal without relying on the other agent. By making the choice in the goals explicit as $\gamma_1 = p \oplus_1 \neg q$ and $\gamma_2 = \neg p \oplus_2 q$, the 'both true' profile has the characteristic that each agent has a strategy that always leads to a successful literal controlled by itself, or an *autonomously successful* strategy—a generalization to *n* players of the notion of winning strategy in two-player games. On the other hand, by putting the other agent's choices into the goals, as in $\gamma_1 = p \oplus_2 \neg q$ and $\gamma_2 = \neg p \oplus_1 q$, neither player can attain its goal without the cooperation of the other: the equilibrium strategy profiles are $(\{p = \top, \oplus_1 = q\}, \{q = \top, \oplus_2 = p\})$ and $(\{p = \bot, \oplus_1 = \neg p\}, \{q = \bot, \oplus_2 = \neg q\})$, but none of the strategies is autonomously successful.

Writing the final goal evaluation as a game also makes explicit the fact that it is really just an extension of the original game: a binary $\oplus_i$ is not morally different from a binary variable choice, but the dependency structure is such that all $\oplus$ choices depend on all variables, or at least on all variables mentioned in the relevant goal. Moreover, although $\mathcal{AL}$ does not have an analogue of (con/dis)-junctive normal form, our

Boolean goals of course do. This leads naturally to the idea of *simple conjunctive* games: games where the final goals are simply conjunctions of literals, and the choice of variables is followed by disjunctive choices $\oplus$, all of which are dependent on the variable choices. Clearly any partial-order Boolean game can be put into this form, as above; and if the $\oplus$s in $\gamma_i$ are all $\oplus_i$, then a simple conjunctive game is just a partial-order Boolean game. Translating a game with other agents' choices in the goals back to a partial-order Boolean game simply requires introducing new variables.

More precisely, a *simple conjunctive game* comprises $N$ agents controlling Boolean variables $B$ with a dependency order $D$ as for partial-order Boolean games. Each agent has a goal $\gamma_i$ which is a formula in the following language: literals are variables or negated variables, terms are conjunctions of literals, which are formulas, and if $\varphi$ and $\psi$ are formulas so is $\varphi \oplus_i \psi$ for each $i$. $D$ is extended thus: each occurrence of a $\oplus$ operator is dependent upon all variables and upon any other $\oplus$ operators that enclose it. Choice functions for agent $i$ are extended accordingly to make choices for every $\oplus_i$ operator; strategies and strategy profiles extend in the obvious way.

**Proposition 8** *Every simple conjunctive game $G$ has an isomorphic partial-order Boolean game $G'$, such that each strategy (profile) in $G$ has a strategy (profile) in $G'$ with the same resulting successes or failures for each agent, and vice versa.*

*Proof* Let $G$ be a simple conjunctive game as above. Let $G'$ contain the variables of $G$, together with one new variable $r_i^k$ for every occurrence $\varphi^k \oplus_i^k \psi^k$ of a choice operator. The control and dependencies of the $r_i^k$ are inherited from those of their choice operators. The goal $\gamma_i'$ of agent $i$ in $G'$ is obtained by recursively applying the following operation to $\gamma_i$: map $\varphi^k \oplus_j^k \psi^k$ to $(r_j^k \wedge \varphi^k) \vee (\neg r_j^k \wedge \psi^k)$. Then every choice in a $G$ strategy determines a choice in a $G'$ strategy, with the choice of $\varphi^k$ corresponding to setting $r_j^k = \top$, and by construction the outcomes correspond. □

## 7 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—i.e., a "true concurrency" framework. 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 [3], processes in concurrent constraint programs [33], or agents in asynchronous systems [28].

**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 [29]. 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 [9, 10] 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 [9,10] are two-player zero-sum games, while the games in this paper are, in general, $n$-player non-zero-sum games. Only recently [19], 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, with IF as a two-player fragment. This idea has also been explored using a categorical model [2]. However, there is little 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 is problematic. 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 with IF logic itself, our games may be equipped with rules adequate for practical reasoning—*cf.* [25].

In this article, we have made use of the DQBF formalism. This was introduced, and its complexity established, by [32] as a Henkin-quantifier [22] extension of QBF, independently of the earlier development of independence-friendly logic. DQBF is a restriction of IF logic over a Boolean domain. More recently, [4] have developed the links between DQBF and IF logic: their Skolem models and Herbrand counter-models for DQBF correspond to the teams for $\exists$ and $\forall$ in the symmetrical team or trump semantics for IF logic (see [25] for a survey of IF semantics). They then restrict the form of DQBFs to produce a class where synthesis of the models is easier, by limiting the dependencies between variables. It may be that similar techniques will apply to our more multi-player formalism. Finally, it is also worth noting that whereas in [22] there is an argument to use Henkin quantifiers, and therefore IF logic too, to reason about "infinitely-long" formulae, our setting only considers the finite case, both at the logical and game-theoretic level. As a consequence, our framework can be seen as a special case from that viewpoint. On the other hand, both in IF and in DQBF, the setting is naturally captured by a two-player strategic game, where we can allow for multi-player interactions. Then, in that sense, the reasoning framework we develop here is more general.

## 8 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 [14]. 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.

## References


1. Abramsky, S.: Sequentiality vs. concurrency in games and logic. Mathematical Structures in Computer Science **13**(4), 531–565 (2003)
2. Abramsky, S.: Socially responsive, environmentally friendly logic. Acta Philosophica Fennica **78** (2006). Truth and Games: Essays in Honour of Gabriel Sandu
3. Abramsky, S., Melliès, P.A.: Concurrent games and full completeness. In: LICS, pp. 431–442. IEEE Computer Society (1999)
4. Balabanov, V., Chiang, H.J., Jiang, J.H.: Henkin quantifers and Boolean formulae: A certification perspective of DQBF. Theoretical Computer Science **523**, 86–100 (2014)
5. Bonzon, E., Lagasquie, M., Lang, J., Zanuttini, B.: Boolean games revisited. In: ECAI (2006)
6. Bonzon, E., Lagasquie-Schiex, M.C., Lang, J.: Dependencies between players in Boolean games. International Journal of Approximate Reasoning **50**(6), 899–914 (2009)
7. Boppana, R.B., Sipser, M.: The complexity of finite functions. In: Handbook of Theoretical Computer Science Volume A: Algorithms and Complexity, pp. 757–804. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands (1990)
8. Bradfield, J.C.: Independence: logics and concurrency. Acta Philosophica Fennica **78**, 47–70 (2006). Truth and Games: Essays in Honour of Gabriel Sandu
9. Clairambault, P., Gutierrez, J., Winskel, G.: The winning ways of concurrent games. In: LICS, pp. 235–244. IEEE Computer Society (2012)
10. Clairambault, P., Gutierrez, J., Winskel, G.: Imperfect information in logic and concurrent games. In: Computation, Logic, Games, and Quantum Foundations, *LNCS*, vol. 7860, pp. 7–20. Springer (2013)
11. Cook, S., Soltys, M.: Boolean programs and quantified propositional proof systems. Bulletin of the Section of Logic **28**(3), 119–129 (1999)
12. Dunne, P.E., Kraus, S., van der Hoek, W., Wooldridge, M.: Cooperative Boolean games. In: AAMAS (2008)
13. Emerson, E.A.: Temporal and modal logic. In: J. van Leeuwen (ed.) Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics, pp. 996–1072. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands (1990)
14. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. The MIT Press: Cambridge, MA (1995)
15. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann Publishers: San Mateo, CA (2004)
16. Godefroid, P.: Partial-order methods for the verification of concurrent systems, *Lecture Notes in Computer Science*, vol. 1032. Springer-Verlag: New York (1996)
17. Grant, J., Kraus, S., Wooldridge, M., Zuckerman, I.: Manipulating Boolean games through communication. In: IJCAI (2011)
18. Gutierrez, J.: Concurrent logic games on partial orders. In: WoLLIC, *LNCS*, vol. 6642, pp. 146–160. Springer (2011)
19. Gutierrez, J., Wooldridge, M.: Equilibria of concurrent games on event structures. In: LICS. ACM Press (2014)
20. Harrenstein, P., van der Hoek, W., Meyer, J.J., Witteveen, C.: Boolean games. In: TARK, pp. 287–298 (2001)
21. Hearn, R.A., Demaine, E.D.: Games, Puzzles, & Computation. A. K. Peters, Ltd.: Wellesley, MA (2009)
22. Henkin, L.: Some Remarks on Infinitely Long Formulas. Journal of Symbolic Logic **30**(1), 167–183 (1961)
23. Jurdziński, M., Nielsen, M., Srba, J.: Undecidability of domino games and hhp-bisimilarity. Information and Computation **184**(2), 343–368 (2003)
24. Koller, D., Milch, B.: Multi-agent influence diagrams for representing and solving games. Games and Economic Behavior **45**(1), 181–221 (2003)
25. Mann, A.L., Sandu, G., Sevenster, M.: Independence-friendly logic. A game-theoretic approach., *LMS Lecture Note Series*, vol. 386. Cambridge University Press (2011)
26. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag: Berlin, Germany (1992)
27. Mavronicolas, M., Monien, B., Wagner, K.W.: Weighted Boolean formula games. In: WINE, pp. 469–481 (2007)
28. Melliès, P.A., Mimram, S.: Asynchronous games: Innocence without alternation. In: CONCUR, *LNCS*, vol. 4703, pp. 395–411. Springer (2007)
29. Nielsen, M., Winskel, G.: Models for concurrency. In: Handbook of Logic in Computer Science. Oxford University Press: Oxford, England (1995)
30. Osborne, M.J., Rubinstein, A.: A Course in Game Theory. The MIT Press: Cambridge, MA (1994)
31. Pauly, M.: Logic for social software. Ph.D. thesis, University of Amsterdam (2001)
32. Peterson, G.L., Reif, J.H., Azhar, S.: Lower bounds for multiplayer noncooperative games of incomplete information. Computers and Mathematics with Applications **41**, 957–992 (2001)
33. Saraswat, V.A., Rinard, M.C., Panangaden, P.: Semantic foundations of concurrent constraint programming. In: POPL, pp. 333–352. ACM Press (1991)
34. Winskel, G.: Deterministic concurrent strategies. Formal Aspects of Computing **24**(4-6), 647–660 (2012)
35. Wooldridge, M., Endriss, U., Kraus, S., Lang, J.: Incentive engineering for Boolean games. Artificial Intelligence **195**, 418–439 (2013)