Computing Extensive-Game Backward-Induction Strategies by Model-Checking a Confluence Modal Characterization of the Game Solution

Pedro Arturo Góngora David A. Rosenblueth

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
Universidad Nacional Autónoma de México
A.P. 20-726, C.P. 01000, México D.F., México
pedro.gongora@gmail.com, drosenbl@servidor.unam.mx

Abstract

Studying game-theoretic problems with modal logics is an important and promising approach for the mutual benefit of both these areas. Some of the existing modal characterizations of Nash equilibria have established that such equilibria obey a special case of modal confluence in finite games. These characterizations are based on validity, requiring that a property is preserved in every model. From a practical point of view, model checking appears as an attractive set of tools for reasoning about large non-deterministic systems. Model checking, however, verifies property satisfaction in a particular model. We propose using hybrid logic to take these validity-based characterizations to a model-checking setting. In addition, we present an example in hybrid PDL for finite, non-cooperative games.

1 Introduction

Part of the recent research into combining game theory and modal logics focuses on foundational topics, such as correspondence results between game frames and solutions (see [3], [8] and the survey [13]). It is, however, still a matter of current research whether problem solving in game theory, such as solution computation, may benefit from tools and techniques developed for modal logics. Verification of validity requires to prove that a property is preserved in every model. Model checking [4, 2], by contrast, is a technique for mechanically (and efficiently) verifying property satisfaction in a particular model. Harrenstein et al. [8] characterize game solutions as a special case of modal confluence in treelike models. These characterizations rely on validity, and cannot be used directly in a model-checking approach. Our proposal is to use hybrid-logic extensions [1] as a bridge between these foundational characterizations and practical model checking. Specifically, the purpose of this approach is to compute the solution of a game by model checking the validity-based characterizations of the solutions.

We present an example in Propositional Dynamic Logic (PDL) with hybrid-logic extensions. The method we present here requires building a game model consisting of the tree structure of the game and the players' preferences. We use PDL programs to recursively

define the relation between every non-final node and the best outcome reachable from such a non-final node. This relation is a special case of modal confluence, and we use a formula with hybrid-logic operators to characterize the states locally satisfying this property. Then, a model checker can traverse the model reconstructing the strategies of a solution.

The rest of the paper is organized as follows. In Sect. 2 we outline the concepts of extensive games, Nash equilibria and backward induction. Such concepts are sufficient for understanding the contents of the paper. In Sect. 3 we present the formal syntax and semantics of hybrid PDL. Section 4 is devoted to our method for computing backward-induction strategies with hybrid PDL. Section 5 finishes the paper with some final thoughts and conclusions.

2 Extensive-Form Games and Backward Induction

Games are formal descriptions of rational agents' interactions. Such interactions may be formalized as a set of sequences of events (extensive form). The nature of the game (e.g., of conflict or agreement) is described numerically, assigning utility units to each agent at every possible outcome of the game. Although there are several classes of games, in this paper we will deal only with finite, non-cooperative, complete and perfect information games in extensive form. We refer the reader to [9] for a more thorough introduction to the concepts presented in this section.

An extensive-form game can be represented as a tree. A non-final node or state represents the turn of one player, and has one successor for each possible action available to the player at that time. In finite games, the leaves represent the possible outcomes of the game. In a game tree, a player i's strategy is a function that chooses a single successor for every node representing i's turn. A strategy profile is a set consisting of one strategy for each player.

Definition 2.1 (Finite Extensive-Form Games, Strategies, and Strategy Profiles)**.** *A finite extensive-form game G is a tree defined as the tuple:*

$$G \stackrel{\text{def}}{=} \langle S, \prec, s_0, N, P, \{u_i\}_{i \in N} \rangle$$

where:

- $S \stackrel{\text{def}}{=} \{s_0, s_1, \ldots, s_n\}$ *is the set of nodes of the tree, with $Z \subseteq S$ denoting the leaves;*
- $\prec \subseteq (S \times S)$ *orders S into a tree with root $s_0 \in S$;*
- $N \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$ *is the set of players;*
- $P : (S - Z) \to N$ *is a move (or turn) function;*
- $u_i : Z \to \mathbb{R}$ *is the player i's utility function.*

A strategy a_i for player i, maps an element in $\{s \in S \mid P(s) = i\}$ with one of its $\prec$-successors. Let A_i denote the set of all strategies for player i. A strategy profile is a tuple $a \in \times_{\{i \in N\}} A_i$. Let A denote the set of all strategy profiles.

Example 2.2. Consider the game depicted in Fig. 1. The game has two players, 1 and 2. The results of P are depicted above the non-final nodes, and the results of the u_i functions

are depicted below the leaves. Player 1 moves first, then player 2 moves and the game ends. Suppose that 1 moves right and then 2 also moves right, then the agents will gain 4 and 2 utility units respectively. From the root node, this outcome is determined if all players follow the strategy profile emphasized with thick arrows in the figure.

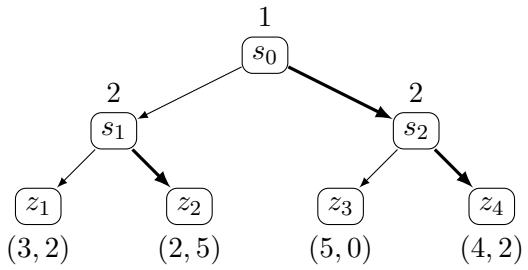

Figure 1: A two-player game and its backward-induction strategy profile.

A key feature of the game-theoretic approach to decision making is that all players are assumed rational. A player's rationality dictates her to always choose the actions maximizing her utility. Consequently, a solution for a game is a method for choosing strategies maximizing the utility of all players. A well-known solution concept is that of *Nash equilibria*. A strategy profile is a Nash equilibrium if every player cannot increase her utility by deviating unilaterally from the profile.

The *backward-induction* solution is a special case of Nash equilibria: the subgame-perfect Nash equilibria. A backward-induction algorithm follows from the inductive proof by Kuhn that every finite extensive game[1] is determined.

Backward induction can be explained as the counterfactual reasoning that a player may use for choosing her actions: *"If I chose this action, then the next player will choose his best action and my payoff will be ... "*. This counterfactual reasoning is applied recursively until a possible ending of the game is reached.

Example 2.3. Consider again the two-player game in Fig. 1. Player 1 has to decide whether to choose s_1 or s_2. If 1 chooses s_1, then 2 will surely choose z_2. If 1 chooses s_2, then 2 will surely choose z_4. Thus, 1 will choose s_2 for maximizing her payoff. The thick arrows show the result of this backward induction. Also, observe that this result may be identified with the set $\{s_2, z_2, z_4\}$, whose elements correspond to the best decision taken at a non-final state.

3 Propositional Dynamic Logic with Hybrid Extensions

Propositional Dynamic Logic (PDL) is an extension to multimodal logic. In PDL, we can define new modalities from a given basic set. We define this new modalities, called *programs*, by using an appropriate language. In this paper, we will use PDL with regular programs.

We first present the syntax and semantics of pure PDL, then we present an extension using a hybrid-logic operator. For more details on PDL and the hybrid-logic extensions to PDL presented here, we refer the reader to [7] and the original papers [10] and [5].

[1] Kuhn's result applies only to games of perfect and complete information.

Definition 3.1 (PDL Syntax)**.** *Let $AtProp = \{p_0, p_1, \ldots\}$ be a countable set of atomic propositions and $\Pi_0 = \{a_0, a_1, \ldots\}$ be a countable set of atomic programs. The set of all formulas of PDL is the least set generated by the following grammar:*

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid [\pi]\varphi$$
$$\pi ::= a \mid (\pi;\pi) \mid (\pi \cup \pi) \mid (\pi^*) \mid \varphi?$$

where $p \in AtProp$ y $a \in \Pi_0$.

Definition 3.2 (PDL Models, Satisfaction, Program and Formula Denotation)**.** *A PDL model $\mathfrak{M}$ is the tuple:*

$$\mathfrak{M} \stackrel{\text{def}}{=} \langle S, \{R_a\}_{a\in\Pi_0}, \ell\rangle$$

where:

- $S \stackrel{\text{def}}{=} \{s_0, \ldots, s_n\}$ *is the set of states;*
- $R_a \subseteq (S \times S)$ *is an accessibility relation for the atomic program $a \in \Pi_0$;*
- $\ell : S \to 2^{AtProp}$ *is a state-labeling function.*

Given some state s of a model $\mathfrak{M}$ we will call the pair $(\mathfrak{M}, s)$ a pointed model. We define the satisfaction relation $\models$ between pointed models and PDL formulas as the least relation such that:

1. $(\mathfrak{M}, s) \models p$ *iff* $p \in \ell(s)$;
2. $(\mathfrak{M}, s) \models \neg\varphi$ *iff* $(\mathfrak{M}, s) \not\models \varphi$;
3. $(\mathfrak{M}, s) \models (\varphi \wedge \psi)$ *iff* $(\mathfrak{M}, s) \models \varphi$ *and* $(\mathfrak{M}, s) \models \psi$;
4. $(\mathfrak{M}, s) \models [\pi]\varphi$ *iff for all* $s' \in W$, $(s, s') \in [\![\pi]\!]$ *implies* $(\mathfrak{M}, s') \models \varphi$.

Where the PDL program denotation relation $[\![\pi]\!] \subseteq (S \times S)$ is defined as follows:

$$[\![a]\!] \stackrel{\text{def}}{=} R_a$$
$$[\![\pi_1;\pi_2]\!] \stackrel{\text{def}}{=} [\![\pi_1]\!] \circ [\![\pi_2]\!]$$
$$[\![\pi_1 \cup \pi_2]\!] \stackrel{\text{def}}{=} [\![\pi_1]\!] \cup [\![\pi_2]\!]$$
$$[\![\pi^*]\!] \stackrel{\text{def}}{=} ([\![\pi]\!])^*$$
$$[\![\varphi?]\!] \stackrel{\text{def}}{=} \{(s,s) \mid (\mathfrak{M}, s) \models \varphi\}$$

Finally, we define the formula denotation $[\![\varphi]\!]$ as the set of states satisfying φ:

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \{s \in S \mid (\mathfrak{M}, s) \models \varphi\}$$

A model may be regarded as a labeled transition system (LTS), and programs as rules on how to traverse this transitions. An atomic program a specifies taking all transitions labeled with a. The construct ";" stands for *sequential composition*, "$\cup$" for *non-deterministic choice*,

and "*" for *non-deterministic iteration* (Kleene star). The program "$\varphi?$" is called a *test* of whether φ holds at the current state.

The intended reading of the formula $[\pi]\,\varphi$ is: "*φ necessarily holds after every possible execution of π*". We can also define the following dual modality: $\langle\pi\rangle\varphi \stackrel{\text{def}}{=} \neg\,[\pi]\,\neg\varphi$. Thus, we can read the formula $\langle\pi\rangle\varphi$ as: "*there is a possible execution of π reaching a φ-state*". We will use π^n to denote the n-times iteration of the program π, thus $\pi^1 = \pi$, $\pi^2 = \pi;\pi$, and so on. The rest of the logical connectives ($\vee$, $\Rightarrow$ and $\Leftrightarrow$) are defined as usual in terms of $\neg$ and $\wedge$ (e.g., by using the De Morgan Laws). Also, we define the formula $\top \stackrel{\text{def}}{=} (p \vee \neg p)$, for some fixed atomic proposition p, and the formula $\bot \stackrel{\text{def}}{=} \neg\top$.

We now present the hybrid extensions to PDL that we will later use. Hybrid logics extend modal logics with several new operators. We will only use a subset of these extensions, specifically, we will use *nominals* and the *state-variable binder* $\downarrow$. For a detailed explanation on this extensions, and others as well, see the mentioned papers [10] and [5].

Definition 3.3 (Nominals, State Variables, and State-Variable Binder)**.** *Let Nom be a set of nominals such that each nominal identifies at least one state, and each state is identified by at least one nominal. For simplicity, we will identify the set of nominals with S, assuming that S and AtProp are disjoint sets. Then, for any nominal t, we define:*

- $(\mathfrak{M}, s) \models t$ *iff* $s = t$.

Let Var be a set of countably many state variables with σ, ρ, etc. ranging over Var. For any state variable σ and any formula φ, we add two formula constructs, variables and the state-variable binder:

- σ;
- $\downarrow\sigma.\,\varphi$.

We define the notions of free and bound variables using analogue rules as those from classical predicate logic. Then, we define:

- $(\mathfrak{M}, s) \models \downarrow\sigma.\,\varphi$ *iff* $(\mathfrak{M}, s) \models \varphi\{s/\sigma\}$.

Where $\varphi\{s/\sigma\}$ is the formula φ with all the free instances of variable σ substituted by the nominal s. For simplicity, we consider only formulas not having free variables.

4 Computing Backward-Induction Strategies

We can use modal logics for describing properties of transition systems. A modal logic formula may be satisfied at several states in such a transition system or model. The problem of deciding whether a formula φ is satisfied at a state s of a model $\mathfrak{M}$ (viz. $(\mathfrak{M}, s) \models \varphi$) is called the *model-checking problem*. The *state-labeling* algorithm for model checking computes the set $[\![\varphi]\!]$ of all states satisfying a given modal formula φ. The algorithm makes this computation by inductively extending the labeling function ℓ from atomic propositions to all formulas.

Our goal is computing backward-induction strategies by using model checking. A possible method is defining a modal formula characterizing all the states representing a best decision taken at non-final nodes. For example, see the game in Fig. 1. The expected result would

be the set $\{s_2, z_2, z_4\}$. We first present an intuitive explanation of our method by using this example.

We proceed inductively on the depth of the game tree. For the base case, consider the subtree with root s_2. Following the algorithm, player 2 chooses z_4 because it is *the best among the siblings*: the utility associated with z_4 is greater than or equal to the utility associated with the other siblings.

Let $\xrightarrow{sib}$ relate a leaf with itself and its siblings. Let $\xrightarrow{2}$ represent an order on leaves such that $z \xrightarrow{2} z'$ iff $u_2(z) \leq u_2(z')$. Figure 2 shows subsets of these two relations: in panel (a) for the analysis of z_3, and in panel (b) for the analysis of z_4. Observe that only for the best sibling z_4, every path leaving z_4 with $\xrightarrow{sib}$ has a possible return with $\xrightarrow{2}$. Restricting the attention to these portions of the relations, only for z_4 it holds that $\xrightarrow{sib}$ is contained in the converse of $\xrightarrow{2}$.

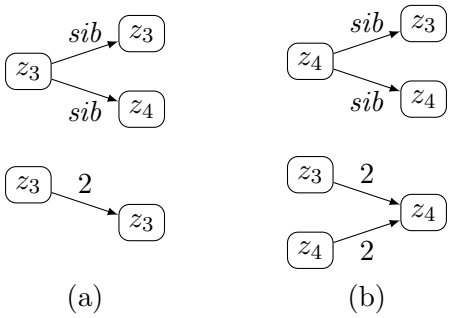

Figure 2: (a) $\xrightarrow{sib}$ is not contained in the converse of $\xrightarrow{2}$, (b) for the best sibling z_4, $\xrightarrow{sib}$ is contained in the converse of $\xrightarrow{2}$.

Recall that the multimodal logic axiom scheme:

$$\varphi \rightarrow [a]\langle b\rangle\varphi \tag{1}$$

characterizes all multimodal frames such that the relation $\xrightarrow{a}$ is contained in the converse of the relation $\xrightarrow{b}$ (cf. [11], chapters 5 and 6). Such a characterization relies on validity: it is necessary to prove that (1) holds for every φ and for every possible valuation of atomic propositions. Thus, (1) cannot be used directly in a model-checking approach that relies on satisfaction on a particular model.

Note, however, that in our example, if there were a formula σ identifying exactly the desired leaf, it would suffice to verify that (1) holds for $\varphi = \sigma$. This is the reason that motivates using hybrid-logic extensions.

For the base case in our example, by using the hybrid-logic binding operator $\downarrow$, a first approach to the desired characterization is as follows:

$$\downarrow\sigma.\ [sib]\langle 2\rangle\sigma \tag{2}$$

Thus (2) characterizes all the states σ locally satisfying this desired property.

For the inductive case we use a similar reasoning. In the example, at the root of the tree 1 has to choose between s_1 and s_2. In fact, the decision is choosing between z_2 and z_4. This

is because it is assumed that the algorithm is recursively applied at s_1 and at s_2. Thus, we say that z_2 and z_4 are *virtual siblings*, and use the same method as before.

We now present the formal definitions of a game model and the best sibling characterization.

Definition 4.1 (Game Models)**.** *Let $G = \langle S, \prec, s_0, N, P, \{u_i\}_{i\in N}\rangle$ be a game as defined in Def. 2.1. A game model $\mathfrak{M}_G$ is the tuple:*

$$\mathfrak{M}_G \stackrel{\text{def}}{=} \langle S, \{R_a\}_{a\in\Pi_0}, \ell\rangle$$

where:

- $\Pi_0 \stackrel{\text{def}}{=} \{pred, succ\} \cup N$*;*
- $(s, s') \in R_{pred}$ *iff s is an immediate $\prec$-predecessor of s';*
- $(s, s') \in R_{succ}$ *iff s' is an immediate $\prec$-successor of s;*
- $(z, z') \in R_i$ *iff $z, z' \in Z$, $i \in N$, and $u_i(z) \leq u_i(z')$;*
- *for all $i \in N$, there are atomic propositions of the form $turn_i$, such that $turn_i \in \ell(s)$ iff $s \in (S - Z)$ and $P(s) = i$.*

A game model consist of the following components. First, the model describes the structure of the game tree by using successor and predecessor relations between the nodes. Secondly, the model encodes the players' turn information with propositional variables. Finally, the model includes the preference orders on the leaves induced by the player's utility functions.

Definition 4.2 (Best Sibling Characterization)**.** *We define the formula $BS(h)$ characterizing the best siblings in a game tree of depth h:*

$$BS(1) \stackrel{\text{def}}{=} \bigvee_{i\in N} \downarrow\sigma.\ (\langle pred\rangle turn_i \wedge \langle pred\rangle maxDepth(1) \wedge [sib]\langle i\rangle\sigma)$$

$$BS(h{+}1) \stackrel{\text{def}}{=} \bigvee_{i\in N} \downarrow\sigma.\ (\langle pred\rangle turn_i \wedge \langle pred\rangle maxDepth(h{+}1) \wedge [toBest(h)]\downarrow\rho.\ [vsib(\sigma, h)]\langle i\rangle\rho)$$

where:

$$maxDepth(h) \stackrel{\text{def}}{=} \langle succ^h\rangle[succ]\bot \wedge [succ^{h+1}]\bot$$

$$sib \stackrel{\text{def}}{=} pred; succ$$

$$vsib(\sigma, h) \stackrel{\text{def}}{=} toAncestor(\sigma); sib; toBest(h)$$

$$toAncestor(\sigma) \stackrel{\text{def}}{=} pred^*; \sigma?$$

$$toBest(h) \stackrel{\text{def}}{=} (leaf?) \cup \bigcup_{1\leq j\leq h} ((succ; BS(j)?); \cdots; (succ; BS(1)?))$$

The best-sibling characterization depends on the depth of the game tree, and we provide two formulas: one for trees of depth 1 (base case), and another for trees of arbitrary depth (inductive case).

The beginning of the characterization is the same for both formulas. We use the binding operator $\downarrow$ to anchor the evaluation to some node s. The outer disjunction and the formula $\langle pred \rangle turn_i$ state that s is a decision taken by some player i. The formula $\langle pred \rangle maxDepth(\cdot)$ assures that the game tree under consideration is of the desired depth.

For the base case, the rest of the characterization is as described before. Any leaf reachable from s by using the program sib, has a possible returning transition by using the program i. This states that s is as desirable as any other s-sibling.

For the inductive case, recall that the preference relation is defined only for the final nodes. For this reason, we use the PDL program *toBest* for relating the non-final node s with a leaf z, such that z represents the outcome of the sub-game with root r (predecessor of s). We use again the operator $\downarrow$ to anchor the evaluation to the leaf z, and then employ the same characterization but comparing z with its virtual siblings.

The recursive nature of the backward-induction algorithm is embodied in the programs *vsib* and *toBest*. The program *vsib* is straightforward: this program relates the leaf z with its virtual siblings by returning to the root r, and applying the program *toBest* to all the r successors. If the depth of the tree from r is $h+1$, then the program *toBest* reconstructs the path of best decisions for the sub-games of depth $j < h+1$ (the tree does not have to be balanced).

Finally, we characterize the set of nodes representing the best decisions taken at non-final nodes in a game-tree of depth h as the following set:

$$NE(h) \stackrel{\text{def}}{=} \bigcup_{1 \leq j \leq h} [\![BS(j)]\!]$$

5 Conclusions

In [3], Bonanno proposes to use temporal logic to formalize the backward-induction solutions of games. In [8], Harrenstein et al. provide correspondence results between game frames, encoding backward-induction solutions, and a special case of modal confluence. Both [3] and [8] are validity-based characterizations. There are other satisfaction-based characterizations. Van der Hoek et al. [15] use a variation of alternating-time logic. Troquard et al. [12] use a preference logic with hybrid extensions. Góngora and Rosenblueth [6] use probabilistic temporal logic for characterizing solutions with mixed strategies.

Van Benthem et al. [14] use a *ceteris paribus* preference logic for characterizing Nash equilibria. This characterization is also local, and a model checker for that logic may also compute the solutions of a game. The difference with our approach is that in [14] the characterization is for normal-form games. Thus, the approach in [14] includes all possible solutions, while backward-induction computes only sub-game perfect solutions.

Passy and Tinchev [10] introduce for the first time PDL with nominals and other extensions. Franceschet and de Rijke [5] introduce PDL with various hybrid extensions, and also present model-checking complexity results for these logics. Here, we only use a subset of the languages presented in [5].

In this paper, we propose using hybrid-logic extensions to bridge validity-based Nash equilibria characterizations and practical model checking. By using such extensions, we show that a model checker can compute the equilibria, besides just characterizing them. We provide an example of this computation by using hybrid PDL. We use a PDL formula, corresponding

to a special case of confluence, for characterizing the best decision taken at non-final nodes in a game tree. A model checker can compute the set of all nodes characterized by this formula, and thus reconstruct the strategy profile of a backward-induction solution.

We conclude by pointing out some directions for further research that we consider desirable. For the example presented here, it is important to investigate what kind of optimizations are possible in practical implementations. When using the binding operator $\downarrow$, the complexity of model checking becomes exponential in the depth of the occurrences of $\downarrow$ in the formula. In our definitions, however, we can reuse results and keep the operator $\downarrow$ at low depths. Also, by rewriting some formulas in a guarded-command fashion, it is possible to avoid many computations (e.g., in the *toBest* program). Another direction of research is exploring the possibility of using logics more common in the model-checking community, for example, Computation Tree Logic (CTL). Finally, it would be interesting to study whether this approach can be applied to games with only mixed-strategy solutions.

References

[1] Areces, C., and ten Cate, B. Hybrid logics. In *Handbook of Modal Logic*, P. Blackburn, J. van Benthem, and F. Wolter, Eds. Elsevier, 2007.

[2] Baier, C., and Katoen, J.-P. *Principles of Model Checking*. The MIT Press, Cambridge, Massachusetts, 2008.

[3] Bonanno, G. Branching time, perfect information games, and backward induction. *Games and Economic Behavior 36* (2001), 57–73.

[4] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems 8*, 2 (Apr. 1986), 244–263.

[5] Franceschet, M., and de Rijke, M. Model checking hybrid logics (with an application to semistructured data). *Journal of Applied Logic 4*, 3 (2006), 279–304.

[6] Góngora, P. A., and Rosenblueth, D. A. A characterization of mixed-strategy Nash equilibria in PCTL augmented with a cost quantifier. In *Proceedings of the 10th International Workshop on Computational Logic in Multi-Agent Systems* (Germany, 2009), Institut für Informatik, Technische Universität Clausthal, pp. 139–155.

[7] Harel, D., Kozen, D., and Tiuryn, J. *Dynamic Logic*. Foundations of Computing. MIT Press, 2000.

[8] Harrenstein, P., van der Hoek, W., Meyer, J.-J. C., and Witteveen, C. A modal characterization of nash equilibrium. *Fundamenta Informaticae 57*, 2-4 (2003), 281–321.

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

[10] Passy, S., and Tinchev, T. An essay in combinatory dynamic logic. *Information and Computation 93* (1991), 263–332.

[11] Popkorn, S. *First Steps in Modal Logic*. Cambridge University Press, Cambridge, England, 1994.

[12] Troquard, N., van der Hoek, W., and Wooldridge, M. Model checking strategic equilibria. In *MoChArt* (2008), D. Peled and M. Wooldridge, Eds., vol. 5348 of *Lecture Notes in Computer Science*, Springer, pp. 166–188.

[13] van Benthem, J. In praise of strategies. ILLC, University of Amsterdam., 2007. To appear in J. van Eijck & R. Verbrugge, eds., Games, Actions, and Social Software, College Publications, London.

[14] van Benthem, J., Girard, P., and Roy, O. Everything else being equal: A modal logic for ceteris paribus preferences. *Journal of Philosophical Logic 38*, 1 (2008), 83–125.

[15] van der Hoek, W., Jamroga, W., and Wooldridge, M. A logic for strategic reasoning. In *AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems* (New York, NY, USA, 2005), ACM, pp. 157–164.