Artificial Intelligence 248 (2017) 123–157

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

# From model checking to equilibrium checking: Reactive modules for rational verification

Julian Gutierrez, Paul Harrenstein, Michael Wooldridge *

*Department of Computer Science, University of Oxford, United Kingdom*

ARTICLE INFO

*Article history:*
Received 2 July 2015
Received in revised form 2 April 2017
Accepted 8 April 2017
Available online 12 April 2017

*Keywords:*
Complexity of equilibria
Reactive modules
Temporal logic

ABSTRACT

Model checking is the best-known and most successful approach to formally verifying that systems satisfy specifications, expressed as temporal logic formulae. In this article, we develop the theory of *equilibrium checking*, a related but distinct problem. Equilibrium checking is relevant for multi-agent systems in which system components (agents) are assumed to be acting rationally in pursuit of delegated goals, and is concerned with understanding what temporal properties hold of such systems under the assumption that agents select strategies in equilibrium. The formal framework we use to study this problem assumes agents are modelled using Reactive Modules, a system modelling language that is used in a range of practical model checking systems. Each agent (or *player*) in a Reactive Modules *game* is specified as a nondeterministic guarded command program, and each player's goal is specified with a temporal logic formula that the player desires to see satisfied. A strategy for a player in a Reactive Modules game defines how that player selects enabled guarded commands for execution over successive rounds of the game. For this general setting, we investigate games in which players have goals specified in Linear Temporal Logic (in which case it is assumed that players choose deterministic strategies) and in Computation Tree Logic (in which case players select nondeterministic strategies). For each of these cases, after formally defining the game setting, we characterise the complexity of a range of problems relating to Nash equilibria (e.g., the computation or the verification of existence of a Nash equilibrium or checking whether a given temporal formula is satisfied on some Nash equilibrium). We then go on to show how the model we present can be used to encode, for example, games in which the choices available to players are specified using STRIPS planning operators.



## 1. Introduction

Our main interest in this paper is in the analysis of concurrent systems composed of multiple non-deterministic computer programs, in which at run-time each program resolves its non-determinism rationally and strategically in pursuit of an individual goal, specified as a formula of temporal logic. Since the programs are assumed to be acting strategically, game theory provides a natural collection of analytical concepts for such systems [53]. If we apply game-theoretic analysis to such systems, then the main questions to be answered about such systems are not just "what computations might the system produce?", but rather, "what computations might the system produce if the constituent programs *act rationally*?" If we interpret acting rationally to mean choosing strategies (for resolving non-determinism) that are in Nash equilibrium, then

* Corresponding author.
*E-mail address:* mjw@cs.ox.ac.uk (M. Wooldridge).

http://dx.doi.org/10.1016/j.artint.2017.04.003
0004-3702/

this question amounts to asking "*which of the possible computations of the system will be produced in equilibrium*?" Further, if we use temporal logic as the language for expressing properties of our multi-agent and concurrent system (as is standard in the computer aided verification community [20]), then we can also interpret this question as "*which temporal logic formulae are satisfied by computations arising from the selection of strategies in equilibrium*?" We refer to this general problem as *equilibrium checking* [75].

Related questions have previously been considered within computer science and artificial intelligence – see e.g., [14,23, 29,30,51,13,8]. However, a common feature in this previous work is that the computational models used as the basis for analysis are highly abstract, and in particular are not directly based on real-world programming models or languages. For example, in [29] the authors define and investigate iterated Boolean games (iBG), a generalisation of Boolean games [36,37], in which each agent exercises unique control over a set of Boolean variables, and system execution proceeds in an infinite sequence of rounds, with each agent selecting a valuation for the variables under their control in each round. Each player has a goal, specified as a formula of Linear Temporal Logic (LTL), which it desires to see achieved. The iterated Boolean games model is simple and natural, and provides a compelling framework with which to pose questions relating to strategic multi-agent interaction in settings where agents have goals specified as logical formulae. However, this model is arguably rather abstract, and is some distance from realistic programming languages and system modelling languages; we discuss such work in more detail in the related work section towards the end of this article.

In brief, our main aim is to study a framework without these limitations. Specifically, we study game-like systems in which players are specified using (a subset of) the Reactive Modules language [2], which is widely used as a system modelling language in practical model checking systems such as mocha [4] and Prism [43]. Reactive Modules is intended to support the succinct, high-level specification of concurrent and multi-agent systems. As we will see, Reactive Modules can readily be used to encode other frameworks for modelling multi-agent systems (such as multi-agent STRIPS planning systems [10]).

The remainder of the article is structured as follows:

- We begin in the following section by motivating our work in detail, in particular by arguing that the classical notion of system correctness is of limited value in multi-agent systems, and introducing the idea of equilibrium checking as representing a more appropriate framework through which to understand the behaviour of such systems.
- We then survey the logics LTL and CTL, and their semantic basis on Kripke structures, present srml – a sublanguage of Reactive Modules that we use throughout the article – and then develop a formal semantics for it.
- We then introduce Reactive Modules games, in which the structure of the game (what we call the "arena") is specified using Reactive Modules, and the preferences of players are specified by associating a temporal (LTL or CTL) goal formula with each player, which defines runs or computation trees that would satisfy the player's goal.
- We then investigate the complexity of various game-theoretic questions in Reactive Modules games, for both the LTL and the CTL settings, and conclude by discussing the complexity and expressiveness of our new framework against the most relevant related work. Table 2 at the end of the paper summarises our findings.
- Finally, to demonstrate the wider applicability of our framework, we show how it can be used to capture propositional STRIPS games (cf. [22,12,25]), such as the MA-STRIPS model of Brafman and Domshlak [10].

Although largely self-contained, our technical presentation is necessarily terse, and readers may find it useful to have some familiarity with temporal logics [20,18], model checking [16], complexity theory [54], and basic concepts of non-cooperative game theory [53].

## 2. Motivation

Our aim in this section is to motivate and introduce the idea of equilibrium checking as a multi-agent systems counterpart to the standard notion of verification and model checking. (Many readers will be familiar with much of this material – we beg their indulgence so that we can tell the story in its entirety.)

*Correctness and formal verification* The *correctness problem* has been one of the most widely studied problems in computer science over the past fifty years, and remains a topic of fundamental concern to the present day [9]. Broadly speaking, the correctness problem is concerned with checking that computer systems behave as their designer intends. Probably the most important problem studied within the correctness domain is that of *formal verification*. Formal verification is the problem of checking that a given computer program or system $P$ is correct with respect to a given formal (i.e., mathematical) specification $\varphi$. We understand $\varphi$ as a description of system behaviours that the designer judges to be acceptable – a program that guarantees to generate a behaviour as described in $\varphi$ is deemed to correctly implement the specification $\varphi$.

A key insight, due to Amir Pnueli, is that *temporal logic* can be a useful language with which to express formal specifications of system behaviour [56]. Pnueli proposed the use of Linear Temporal Logic (LTL) for expressing desirable properties of computations. LTL extends classical logic with tense operators **X** ("in the next state…"), **F** ("eventually…"), **G** ("always…"), and **U** ("…until …") [20]. For example, the requirement that a system never enters a "crash" state can naturally be expressed in LTL by a formula $\mathbf{G}\neg crash$. If we let $[\![P]\!]$ denote the set of all possible computations that may be produced by the program $P$, and let $[\![\varphi]\!]$ denote the set of state sequences that satisfy the LTL formula $\varphi$, then verification of LTL properties

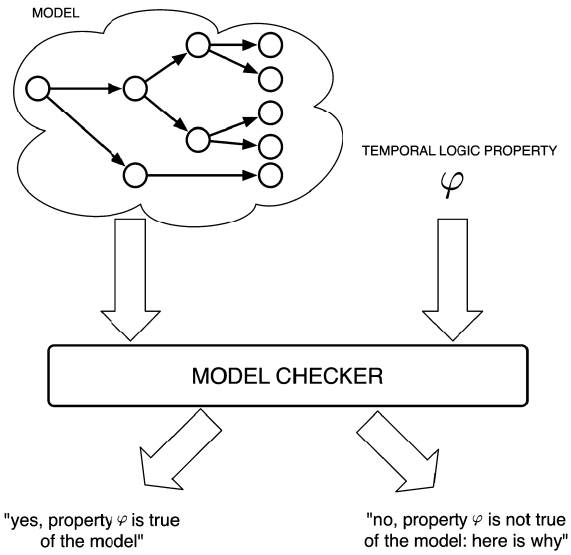



**Fig. 1.** Model checking. A model checker takes as input a model, representing a finite state abstraction of a system, together with a claim about the system behaviour, expressed in temporal logic. It then determines whether or not the claim is true of the model or not; most practical model checkers will provide a counter example if not.

reduces to the problem of checking whether $[\![P]\!] \subseteq [\![\varphi]\!]$. Another key temporal formalism is Computation Tree Logic (CTL), which modifies LTL by prefixing tense operators with *path quantifiers* **A** ("on all paths…") and **E** ("on some path…"). While LTL is suited to reasoning about runs or computational histories, CTL is suited to reasoning about transition systems that encode all possible system behaviours.

*Model checking* The most successful approach to verification using temporal logic specifications is *model checking* [16]. Model checking starts from the idea that the behaviour of a finite state program $P$ can be represented as a Kripke structure, or transition system $K_P$. Now, Kripke structures can be interpreted as models for temporal logic. So, for example, checking whether a program $P$ satisfies an LTL property $\varphi$ reduces to the problem of checking whether $\varphi$ is satisfied on every path through $K_P$. Checking a CTL specification $\varphi$ is even simpler: the Kripke structure $K_P$ is a CTL model, so we simply need to check whether $K_P \models \varphi$ holds. For an illustration see Fig. 1. These checks can be efficiently automated for many cases of interest. In the case of CTL, for example, checking whether $K_P \models \varphi$ holds can be solved in time $O(|K_P| \cdot |\varphi|)$ [15,20]; for LTL, the problem is more complex (PSPACE-complete [20]), but using automata theoretic techniques it can be solved in time $O(|K_P| \cdot 2^{|\varphi|})$ [69], the latter result indicating that such an approach is feasible for small specifications. Since the model checking paradigm was first proposed in 1981, huge progress has been made on extending the range of systems amenable to verification by model checking, and to extending the range of properties that might be checked [16].

*Multi-agent systems* We now turn the class of systems that we will be concerned with in the present paper. The field of *multi-agent systems* is concerned with the theory and practice of systems containing multiple interacting semi-autonomous software components known as *agents* [72,62]. Multi-agent systems are generally understood as distinct from conventional distributed or concurrent systems in several respects, but the most important distinction for our purposes is that different agents are assumed to be operating on behalf of different external principals, who delegate their preferences or goals to their agent. Because different agents are "owned" by different principals, there is no assumption that agents will have preferences that are aligned with each other.

*Correctness in multi-agent systems* Now, consider the following question:

*How should we interpret correctness and formal verification in the context of multi-agent systems*?

In one sense, this question is easily answered: We can certainly think of a multi-agent system as a collection of interacting non-deterministic computer programs, with non-determinism representing the idea that agents have choices available to them; we can model such a system using any number of readily available model checking systems, which would then allow us to start reasoning about the possible computational behaviours that the system might in principle exhibit. But while such an analysis is entirely legitimate, and might well yield important insights, it nevertheless misses a key part of the story that is relevant in order to understand a multi-agent system. This is because *it ignores the fact that agents are assumed to pursue their preferences rationally and strategically*. Thus, certain system behaviours that might be possible *in principle* will never arise *in practice* because they could not arise from rational choices by agents within the system.

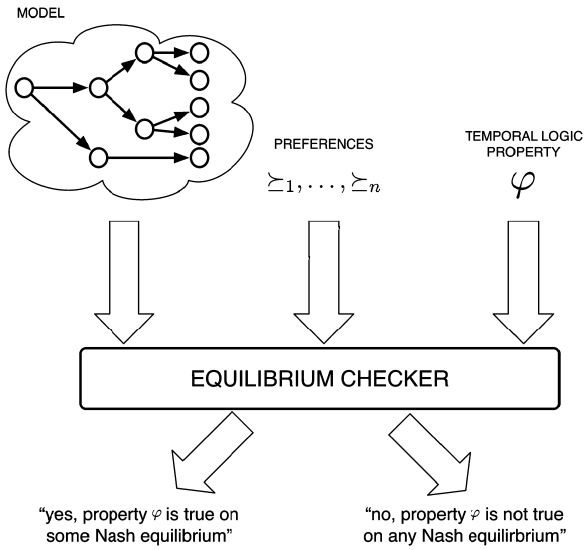



**Fig. 2.** Equilibrium checking. The key difference to model checking is that we also take as input the preferences of each of the system components, and the key question asked is whether or not the temporal property $\varphi$ holds on some/all equilibria of the system.

The classical formulation of correctness does not naturally match the multi-agent system setting because there is no single specification $\varphi$ against which the correctness of a multi-agent system is judged. Instead, *the agents within such a system each carry their own specification*: an agent is judged to be correct if it acts rationally to achieve its delegated preferences or goals. There is no single standard of correctness by which the system as a whole can be judged, and attempting to apply such a standard does not help to understand the behaviour of the system. So, what should replace the classical notion of correctness in the context of multi-agent systems? We will now argue for the concept of *rational verification* and *equilibrium checking*.

*Rational verification and equilibrium checking* We believe (as do many other researchers [52,62]) that *game theory* provides an appropriate analytical framework for the analysis of multi-agent systems. Originating within economics, game theory is essentially the theory of strategic interaction between self-interested entities [49]. While the mathematical framework of game theory was not developed specifically to study computational settings, it nevertheless seems that the toolkit of analytical concepts it provides can be adapted and applied to multi-agent settings. A game in the sense of game theory is usually understood as an abstract mathematical model of a situation in which self-interested players must make decisions. A game specifies the decision-makers in the game – the players – and the choices available to them – their strategies. For every combination of possible choices by players, the game also specifies what outcome will result, and each player has their own preferences over possible outcomes. A key concern in game theory is to try to understand what the outcomes of a game can or should be, under the assumption that the players within it choose rationally in pursuit of their preferences. To this end, many *solution concepts* have been proposed, of which *Nash equilibrium* is probably the best-known. A Nash equilibrium is a collection of choices, one for each participant in the game, such that no player can benefit by unilaterally deviating from this combination of choices. Nash equilibria seem like reasonable candidates for the outcome of a game because to unilaterally move away from a Nash equilibrium would result in that player being worse off – which would clearly not be rational. In general, it could be the case that a given game has no Nash equilibrium, or multiple Nash equilibria.

It should be easy to see how this general setup maps to the multi-agent systems setting: players map to the agents within the system, and each player's preferences are defined as their delegated goals; the choices available to each player correspond to the possible courses of action that may be taken by each agent in the system. Outcomes will correspond to the computations or runs of the system, and agents will have preferences over these runs; they act to try and bring about their most preferred runs.

With this in mind, we believe it is natural to think of the following problem as a counterpart to model checking and classical verification. We are given a multi-agent system, and a temporal logic formula $\varphi$ representing a property of interest. We then ask *whether $\varphi$ would be satisfied in some run that would arise from a Nash equilibrium collection of choices by agents within the system*. We call this *equilibrium checking*, and refer to the general paradigm as *rational verification*. For an illustration see Fig. 2. This idea is captured in the following decision problem:

E-NASH:
*Given*: Multi-agent system $M$; temporal formula $\varphi$.
*Question*: Is it the case that $\varphi$ holds on *some* computation of $M$ that could arise through the agents in $M$ choosing strategies that form a Nash equilibrium?

The obvious counterpart of this decision problem is A-Nash, which asks whether a temporal formula $\varphi$ is satisfied on *all* computations that could arise as a result of agents choosing strategies that form a Nash equilibrium.

A higher-level question is simply whether a system has *some* Nash equilibria:

Non-Emptiness:
*Given*: Multi-agent system $M$.
*Question*: Does $M$ have some Nash equilibria?

A system without any Nash equilibria is inherently *unstable*: whatever collection of choices we might consider for the agents within it, some player would have preferred to make an alternative choice. Notice that an efficient algorithm for solving E-Nash would imply an efficient algorithm for Non-Emptiness.

Finally, we might consider the question of verifying whether a given strategy profile represents a Nash equilibrium:

NE-Membership:
*Given*: Multi-agent system $M$, strategy profile $\vec{\sigma}$.
*Question*: Is $\vec{\sigma}$ a Nash equilibrium?

The aim of the present article is to formulate and investigate the main computational questions relating to equilibrium checking. In order to be able to study these questions, we need to fix on an appropriate model of multi-agent systems, and also some way of defining preferences for this model. As we noted above, the model of multi-agent systems that we adopt is that of Reactive Modules [2]. Our argument for studying this framework is that it is a well-known, widely studied, and most importantly, *widely used* framework for modelling concurrent and multi-agent systems [4,67,43]. In short, Reactive Modules represents a *realistic* computational framework within which to study the questions of interest to us. This leaves the question of how to define preferences. Our approach here is very natural: we assume that preferences are defined by giving each agent a temporal logic *goal*, which the agent is required to try to accomplish. The use of logically-specified goals is commonplace in AI, and temporal logic itself is widely used as a goal specification language within the AI planning community [25]. Moreover, this approach also fits well with the classic view of correctness, as discussed above, wherein a temporal logic goal can be interpreted as a *specification* that the agent should satisfy.

## 3. Preliminaries

We will be dealing with logics that extend propositional logic. Thus, these logics are based on a finite set $\Phi$ of Boolean variables, and contain the classical connectives "$\wedge$" (and), "$\vee$" (or), "$\neg$" (not), "$\rightarrow$" (implies), and "$\leftrightarrow$" (iff), as well as the Boolean constants "$\top$" (truth) and "$\bot$" (falsity). A *valuation* for propositional logic is given by a subset $v \subseteq \Phi$, with the intended interpretation that $x \in v$ means that $x$ is true under valuation $v$, while $x \notin v$ means that $x$ is false under $v$. For propositional formulae $\varphi$ we write $v \models \varphi$ to mean that $\varphi$ is satisfied by $v$. Let $V(\Phi) = 2^{\Phi}$ be the set of all valuations for variables $\Phi$; where $\Phi$ is clear, we omit reference to it and write $V$. Let $v_{\bot} = \emptyset$ be the valuation under which all variables are false. Where $v \in V$, we let $\chi_v$ denote the characteristic formula of $v$, which is satisfied only by the valuation $v$, that is,

$$\chi_v = \bigwedge_{x \in v} x \wedge \bigwedge_{x \notin v} \neg x \,.$$

### 3.1. Kripke structures

We use *Kripke structures* [16, p. 14] to model the dynamics of our systems. A Kripke structure $K$ over $\Phi$ is given by a tuple $K = (S, S^0, R, \pi)$, where $S$ is a finite non-empty set of *states* with typical element $s$ and $R \subseteq S \times S$ is a left total *transition relation* on $S$. Left totality means that for every state $s \in S$, there is another state $s' \in S$ such that $(s, s') \in R$. Moreover, $S^0 \subseteq S$ is the set of *initial states* and $\pi : S \rightarrow V$ is a *valuation function*, assigning a valuation $\pi(s)$ to every $s \in S$. See Fig. 3 for an illustration of a Kripke structure over $\Phi = \{x, y\}$. Where $K = (S, S^0, R, \pi)$ is a Kripke structure over $\Phi$, and $\Psi \subseteq \Phi$, then we denote the *restriction of $K$ to $\Psi$* by $K|_{\Psi}$, where $K|_{\Psi} = (S, S^0, R, \pi|_{\Psi})$ is the same as $K$ except that the valuation function $\pi|_{\Psi}$ is defined as follows: $\pi|_{\Psi}(s) = \pi(s) \cap \Psi$. We say that the *size* of a Kripke structure $K$, denoted by $|K|$, is $|S| + |R| + |\pi|$.

$$\boldsymbol{\rho} = \pi(\rho[0]), \pi(\rho[1]), \pi(\rho[2]), \ldots,$$

### 3.2. Computation runs

A *run* of a Kripke structure $K$ is a sequence $\rho = s_0, s_1, s_2, \ldots$ of states where for all $t \in \mathbb{N}$ we have $(s_t, s_{t+1}) \in R$. Using square brackets around parameters referring to time points, we let $\rho[t]$ denote the state assigned to time point $t \in \mathbb{N}$ by run $\rho$. We say $\rho$ is an *$s$-run* if $\rho[0] = s$. A run $\rho$ of $K$ where $\rho[0] \in S^0$ is referred to as an *initial* run. Thus, in our example in Fig. 3, $s_0, s_0, s_2, s_3, s_3, s_3 \ldots$ and $s_2, s_3, s_1, s_0, s_1, s_0, \ldots$ are both runs but only the former is an initial run. Let $runs(K, s)$

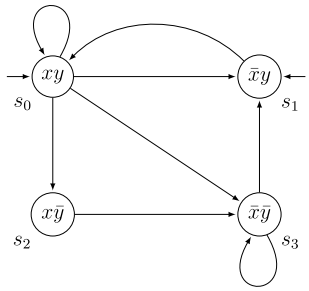



**Fig. 3.** A Kripke structure with four states, $s_0$, $s_1$, $s_2$, and $s_3$, of which the former two are initial states. An arrow from state $s$ to state $s'$ indicates that $(s, s') \in R$. The annotations inside the vertices indicate which propositional variables are set to true at the respective state and which are set to false, where, e.g., $\bar{x}$ indicates that variable $x$ is set to false.

be the set of $s$-runs of $K$, and let $runs(K)$ be the set of initial runs of $K$. Notice that a run $\rho \in runs(K)$ is a sequence of *states*. Every such sequence induces an infinite sequence $\boldsymbol{\rho} \in V^\omega$ of *valuations*, given by

which we also refer to as a *computation run*. In our example in Fig. 3, we thus find that $\{x, y\}, \{x, y\}, \{x\}, \emptyset, \emptyset, \emptyset, \ldots$ is a computation run, whereas, for instance, $\{y\}, \{x\}, \emptyset, \{x, y\}, \{x, y\}, \ldots$ is not. The set of these sequences, (i.e., sequences of valuations corresponding to runs in $runs(K)$) we denote by $\mathbf{runs}(K)$.

Given $\Psi \subseteq \Phi$ and $\boldsymbol{\rho} : \mathbb{N} \to V(\Phi)$, we denote the restriction of $\boldsymbol{\rho}$ to $\Psi$ by $\boldsymbol{\rho}|_\Psi$, i.e., $\boldsymbol{\rho}|_\Psi[t] = \boldsymbol{\rho}[t] \cap \Psi$ for each $t \in \mathbb{N}$. We refer to a finite prefix of a run as a *partial run*, and denote partial runs by $\varrho, \varrho', \ldots$ etc. We extend the notations for runs, restrictions of runs, etc., to initial runs, restrictions of partial runs, etc., in the obvious way.

### 3.3. Linear temporal logic

We use the well-known framework of *Linear Temporal Logic* (LTL) to express properties of runs of our systems [20,46,47]. Formulae of LTL are essentially predicates over infinite sequences of states. LTL extends propositional logic with two modal tense operators, namely, **X** ("next") and **U** ("until"). Formally, the syntax of LTL is defined with respect to a set $\Phi$ of Boolean variables as follows:

$$\varphi ::= x \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\, \mathbf{U}\, \varphi$$

where $x \in \Phi$. The remaining classical logical connectives are defined in terms of $\neg$ and $\vee$ in the usual way. Two key derived LTL operators are **F** ("eventually") and **G** ("always"), which are defined in terms of **U** as follows: $\mathbf{F}\varphi = \top\, \mathbf{U}\, \varphi$, and $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$. Given a set of variables $\Psi$, let $\textsc{Ltl}(\Psi)$ be the set of LTL formulae over $\Psi$; where the variable set is clear from the context, we write $\textsc{Ltl}$.

We interpret formulae of LTL with respect to pairs $(\boldsymbol{\rho}, t)$ where $\boldsymbol{\rho} \in V^\omega$ is a run and $t \in \mathbb{N}$ is a temporal index into $\boldsymbol{\rho}$. Any given LTL formula may be true at none or multiple time points on a run; for example, a formula $\mathbf{X}p$ will be true at a time point $t \in \mathbb{N}$ on a run $\boldsymbol{\rho}$ if $p$ is true on run $\boldsymbol{\rho}$ at time $t + 1$. We will write $(\boldsymbol{\rho}, t) \models \varphi$ to mean that $\varphi \in \textsc{Ltl}$ is true at time $t \in \mathbb{N}$ on run $\boldsymbol{\rho}$. The rules defining when formulae are true (i.e., the semantics of LTL) are defined as follows, where $x \in \Phi$:

$$\begin{aligned}
(\boldsymbol{\rho}, t) &\models x && \text{iff} && x \in \boldsymbol{\rho}[t] \\
(\boldsymbol{\rho}, t) &\models \neg\varphi && \text{iff} && \text{it is not the case that } (\boldsymbol{\rho}, t) \models \varphi \\
(\boldsymbol{\rho}, t) &\models \varphi \vee \psi && \text{iff} && (\boldsymbol{\rho}, t) \models \varphi \text{ or } (\boldsymbol{\rho}, t) \models \psi \\
(\boldsymbol{\rho}, t) &\models \mathbf{X}\varphi && \text{iff} && (\boldsymbol{\rho}, t + 1) \models \varphi \\
(\boldsymbol{\rho}, t) &\models \varphi\, \mathbf{U}\, \psi && \text{iff} && \text{for some } t' \geq t : \text{both } (\boldsymbol{\rho}, t') \models \psi \text{ and} \\
& && && \text{for all } t \leq t'' < t' : (\boldsymbol{\rho}, t'') \models \varphi
\end{aligned}$$

If $(\boldsymbol{\rho}, 0) \models \varphi$, we write $\boldsymbol{\rho} \models \varphi$ and say that $\boldsymbol{\rho}$ *satisfies* $\varphi$. We say that $\varphi$ and $\psi$ are *equivalent* if for all $\boldsymbol{\rho}$ we have $\boldsymbol{\rho} \models \varphi$ if and only if $\boldsymbol{\rho} \models \psi$. A formula $\varphi \in \textsc{Ltl}$ is *satisfiable* if there is some run satisfying $\varphi$; moreover, it is satisfied by a Kripke structure $K$ if it is satisfied by *all* its initial runs, that is, if, and only if, it is satisfied by all $\boldsymbol{\rho} \in \mathbf{runs}(K)$. As usual, we say that the *size* of an LTL formula $\varphi$, denoted by $|\varphi|$, is its number of subformulae.

### 3.4. Computation tree logic

In order to express branching-time properties, we use *Computation Tree Logic* (CTL), a branching temporal logic that extends propositional logic with *tense modalities* and *path quantifiers* [20]. Specifically, CTL combines the tense operators **X**

and **U** with the path quantifiers **A** and **E**; the path quantifier **A** means "on all paths…", while **E** means "on some path…". Formally, given a set $\Phi$ of Boolean variables, the syntax of CTL is defined as follows:

$$\varphi ::= x \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{AX}\varphi \mid \mathbf{E}(\varphi\,\mathbf{U}\,\varphi) \mid \mathbf{A}(\varphi\,\mathbf{U}\,\varphi)$$

where $x \in \Phi$. Assuming the remaining classical connectives have been defined as usual, we can define the remaining CTL operators as follows:

$$\begin{array}{lcllcl} \mathbf{EF}\varphi & = & \mathbf{E}(\top\,\mathbf{U}\,\varphi) & \mathbf{AG}\varphi & = & \neg\mathbf{EF}\neg\varphi \\ \mathbf{AF}\varphi & = & \mathbf{A}(\top\,\mathbf{U}\,\varphi) & \mathbf{EG}\varphi & = & \neg\mathbf{AF}\neg\varphi \\ \mathbf{EX}\varphi & = & \neg\mathbf{AX}\neg\varphi & & & \end{array}$$

Let $\textsc{Ctl}(\Phi)$ denote the set of CTL formulae over $\Phi$; when $\Phi$ is clear from the context, we write $\textsc{Ctl}$. CTL formulae are interpreted with respect to pairs $(K, s)$, where $K$ is a Kripke structure and $s$ is a state in $K$. We write $(K, s) \models \varphi$ to mean that the CTL formula $\varphi \in \textsc{Ctl}$ is satisfied in state $s$ of $K$. Formally, for Kripke structures $K$, states $s$ in $K$, and $x \in \Phi$, we have:

$$\begin{array}{lll} (K,s) \models x & \text{iff} & x \in \pi(s) \\ (K,s) \models \neg\varphi & \text{iff} & \text{it is not the case that } (K,s) \models \varphi \\ (K,s) \models \varphi \vee \psi & \text{iff} & (K,s) \models \varphi \text{ or } (K,s) \models \psi \\ (K,s) \models \mathbf{AX}\varphi & \text{iff} & \forall \rho \in \mathit{runs}(K,s) : (K, \rho[1]) \models \varphi \\ (K,s) \models \mathbf{E}(\varphi\,\mathbf{U}\,\psi) & \text{iff} & \exists \rho \in \mathit{runs}(K,s), \exists t \in \mathbb{N} \text{ such that } (K, \rho[t]) \models \psi \\ & & \text{and } \forall t' \in \mathbb{N} \text{ with } 0 \leq t' < t : (K, \rho[t']) \models \varphi \\ (K,s) \models \mathbf{A}(\varphi\,\mathbf{U}\,\psi) & \text{iff} & \forall \rho \in \mathit{runs}(K,s), \exists t \in \mathbb{N} \text{ such that } (K, \rho[t]) \models \psi \\ & & \text{and } \forall t' \in \mathbb{N} \text{ with } 0 \leq t' < t : (K, \rho[t]) \models \varphi \end{array}$$

If $(K, s_0) \models \varphi$ for all $s_0 \in S^0$, we write $K \models \varphi$ and say that $K$ *satisfies* $\varphi$. A CTL formula $\varphi$ is *satisfiable* if $K \models \varphi$ for some $K$. We say that $\varphi$ and $\psi$ are *equivalent* if $K \models \varphi$ iff $K \models \psi$, for all $K$. We also say that the *size* of a CTL formula $\varphi$, denoted by $|\varphi|$, is its number of subformulae.

### 3.5. Computation trees

We sometimes find it convenient to adopt an alternative view of Kripke structures, where we *unfold* the transition relation $R$ to obtain a *computation tree*. We now present the associated technical definitions for such unfoldings.

First, we assume the standard definitions of a word, string, and prefix of a string. Where $w$ is a finite word over some alphabet, we denote by $\mathit{prefix}(w)$ the set of non-empty prefixes of $w$. A *tree over an alphabet* $\Sigma$ we define by a non-empty set $T \subseteq \Sigma^+$ of non-empty strings over $\Sigma$, such that

(*i*) $T \cap \Sigma$ is a singleton (the root of the tree),
(*ii*) $T$ is closed under non-empty prefixes, i.e., $\mathit{prefix}(w) \subseteq T$ for every $w \in T$, and
(*iii*) $w \in T$ implies $wa \in T$ for some $a \in \Sigma$.

The set $T$ then fixes the set of the vertices and there is an edge between vertices $w, w' \in T$ iff there is an $a \in \Sigma$ such that $w' = wa$.

A *state-tree* for a Kripke structure $K = (S, S^0, R, \pi)$ is a tree $\kappa$ over $S$ such that $(s, s') \in R$ whenever $ws, wss' \in T$ for some $w \in S^*$. An *s-tree* is a state-tree $\kappa$ with root $s$ (i.e., with $s \in \kappa$). Observe that by condition (*i*) every state-tree is an $s$-tree for exactly one state $s$ in $S$. If $s \in S^0$, an $s$-tree is also called an *initial state-tree*. By $\mathit{trees}(K, s)$ we denote the set of $s$-trees and by $\mathit{trees}(K)$ the set of initial state-trees for the Kripke structure $K$.

By a *computation tree* we understand a tree on the set $V(\Phi)$ of valuations over $\Phi$. For $\Psi \subseteq \Phi$ we have $\boldsymbol{\kappa}|_\Psi$ denote the restriction of $\boldsymbol{\kappa}$ to $\Psi$, i.e., for every $u \in T$, $\boldsymbol{\kappa}|_\Psi(u) = \boldsymbol{\kappa}(u) \cap \Psi$. Notice that every state-tree $\kappa$ for a Kripke structure $K$ induces a computation tree $\boldsymbol{\kappa}$ such that

$$\boldsymbol{\kappa} = \{\pi(s_0) \ldots \pi(s_k) : s_1 \ldots s_k \in \kappa\}.$$

In such a case $\boldsymbol{\kappa}$ is said to be *a computation tree for* $K$. If $\kappa$ is an initial state-tree, the corresponding computation tree $\boldsymbol{\kappa}$ is said to be an *initial computation tree*. The set of all initial computation trees for $K$ we denote by $\mathbf{trees}(K)$. We refer to Fig. 4 for an illustration of this concept.

By an *unfolding* of $K = (S, S^0, R, \pi)$ we understand a maximal initial computation tree of $K$. Let $\kappa \in \mathit{trees}(K)$ be an initial state-tree for $K$ such that $s_0 \ldots s_k \in \kappa$ implies $s_0 \ldots s_k s_{k+1} \in \kappa$ for every state $s_{k+1}$ with $s_k \, R \, s_{k+1}$. Then, the corresponding computation tree $\boldsymbol{\kappa} : T \to V$ for $K$ is an *unfolding of* $K$. Each initial state $s \in S^0$ induces a unique unfolding. By $\mathbf{unfold}(K)$ we denote the set of unfoldings of $K$.

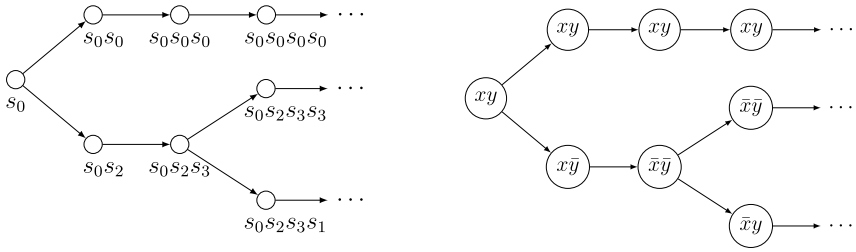



**Fig. 4.** The figure on the left is an (initial) state tree for the Kripke structure in Fig. 3. The figure on the right is the corresponding computation tree, where each vertex $v_1 \dots v_k$ is labelled with the last element $v_k$, e.g., vertex $\{x, y\}\{x\}\emptyset$ (or $xyx\bar{y}\bar{x}\bar{y}$) is labelled with $\emptyset$ (or $\bar{x}\bar{y}$).

## 4. The simple reactive modules language

In this section, we describe the language we use for modelling multi-agent systems: the *Simple Reactive Modules Language* (SRML). This language is a subset of the REACTIVE MODULES Language (RML [2]) that is widely used in model checkers such as MOCHA [4] and PRISM [43]. The fragment we make use of, known as SRML, was introduced by van der Hoek et al. [67] to study the complexity of model checking for the Alternating-time Temporal Logic (ATL) [3]. Our presentation of SRML largely follows that in [67], except that while a formal semantics for SRML was hinted at in [67], we here give a complete formal semantics for the language, showing how SRML-defined systems induce computational runs and Kripke structures.

The structures used to define agents in SRML are known as *modules*. An SRML module consists of:

(*i*) an *interface*, which defines the name of the module and lists the Boolean variables under the *control* of the module; and
(*ii*) a number of *guarded commands*, which define the choices available to the module at every state.

Guarded commands are of two kinds: those used for *initialising* the variables under the module's control (**init** guarded commands), and those for *updating* these variables subsequently (**update** guarded commands). A guarded command has two parts: a condition part (the "guard") and a corresponding action part, which defines how to update the value of (some of) the variables under the control of a module. The intuitive reading of a guarded command $\varphi \rightsquigarrow \alpha$ is "if the condition $\varphi$ is satisfied, then *one of the choices available to the module is to execute the action* $\alpha$". It is important to note that the truth of the guard $\varphi$ does not mean that $\alpha$ *will* be executed: only that it is *enabled* for execution – i.e., that it *may be chosen*.

An action $\alpha$ is a sequence of *assignment statements*. These statements define how (some subset of) the module's controlled variables should be updated if the guarded command is executed. If in some state no guarded command of a given module is enabled, then the values of the variables in that module are assumed to remain unchanged in the next state: the module has no choice.

Here is an example of a guarded command:

$$\underbrace{(x \wedge y)}_{\text{guard}} \rightsquigarrow \underbrace{x' := \bot; y' := \top}_{\text{action}}$$

The guard is the propositional logic formula $x \wedge y$, so this guarded command will be enabled in any system state where both variables $x$ and $y$ take the value "$\top$". If the guarded command is chosen for execution, then the effect is that in the next state of the system, the variable $x$ will take value $\bot$, while $y$ will take value $\top$. (The "prime" notation $x'$ means "the value of variable $x$ *after* the statement is executed".)

The following example illustrates the concrete syntax for modules:

**module** *toggle* **controls** $x$
**init**
:: $\top \rightsquigarrow x' := \top$
:: $\top \rightsquigarrow x' := \bot$
**update**
:: $x \rightsquigarrow x' := \bot$
:: $\neg x \rightsquigarrow x' := \top$

This module, named *toggle*, controls a single Boolean variable, $x$. There are two **init** guarded commands and two **update** guarded commands. (The symbol "::" is a syntactic separator.) The **init** guarded commands of *toggle* define two choices for the initialisation of this variable: assign it the value $\top$ or the value $\bot$. With respect to **update** guarded commands, the first command says that if $x$ has the value $\top$, then the corresponding choice is to assign it the value $\bot$, while the second command says that if $x$ has the value $\bot$, then it can subsequently be assigned the value $\top$. In other words, the module nondeterministically chooses a value for $x$ initially, and then on subsequent rounds toggles this value. Notice that in this example, the **init** commands are nondeterministic, while the **update** commands are deterministic.

### 4.1. Formal definition

We now give a formal definition of the semantics of SRML. Formally, a guarded command $g$ over Boolean variables $\Phi$ is an expression

$$\varphi \rightsquigarrow x'_1 := \psi_1; \cdots; x'_k := \psi_k$$

where $\varphi$ and every $\psi_i$ is a propositional logic formula over $\Phi$ and each $x_i$ is a member of $\Phi$. Let $guard(g)$ denote the *guard* of $g$. Thus, in the above rule, $guard(g) = \varphi$. We will require that no variable appears on the left hand side of two assignment statements in the same guarded command, that is, $x'_1$ through $x'_k$ are all distinct. The intended interpretation of $g$ is that if the propositional formula $\varphi$ evaluates to true against the valuation $v$ corresponding to the current state of the system, then the statement is enabled for execution; executing the statement means evaluating each $\psi_i$ against the current state $v$ of the system, and setting the corresponding variable $x_i$ to the truth value obtained from evaluating $\psi_i$ in this way. We say that $x_1, \ldots, x_k$ are the *controlled variables* of $g$, and denote this set by $ctr(g)$. If none of the guarded commands of a module is enabled, the values of all variables in $ctr(g)$ are left unchanged. In what follows, we will write **skip** as an abbreviation for the assignment that leaves the value of every variable controlled by a module unchanged.

Now, recall that an **init** guarded command is only used to initialise the values of variables when the system begins execution. Full RML allows for quite sophisticated initialisation schemes, but in SRML, it is assumed that the guards to **init** command are "$\top$", i.e., every **init** command is enabled for execution in the initialisation round of the system. We will also assume that in the assignment statements $x' := \psi$ on the right hand side of an **init** command, the expressions $\psi$ are simply Boolean constants, i.e., either $\top$ or $\bot$. Formally, an SRML module, $m_i$, is then defined as a triple:

$$m_i = (\Phi_i, I_i, U_i)$$

where:

- $\Phi_i \subseteq \Phi$ is the (finite) set of variables controlled by $m_i$;
- $I_i$ is a (finite) set of *initialisation* guarded commands, such that for all $g \in I_i$, we have $ctr(g) \subseteq \Phi_i$; and
- $U_i$ is a (finite) set of *update* guarded commands, such that for all $g \in U_i$, we have $ctr(g) \subseteq \Phi_i$.

An SRML *arena* $A$ is then given by an $(n+2)$-tuple:

$$A = (N, \Phi, m_1, \ldots, m_n),$$

where $N = \{1, \ldots, n\}$ is a set of agents, $\Phi$ is a set of Boolean variables, and for each $i \in N$, $m_i = (\Phi_i, I_i, U_i)$ is an SRML module over $\Phi$ that defines the choices available to agent $i$. We require that $\{\Phi_1, \ldots, \Phi_n\}$ forms a partition of $\Phi$ (so every variable in $\Phi$ is controlled by some agent, and no variable is controlled by more than one agent). For a propositional valuation $v \subseteq \Phi$, let $v_i$ denote $v \cap \Phi_i$.

The *size* of an SRML arena $A = (N, \Phi, m_1, \ldots, m_n)$, denoted by $|A|$, is defined to be $|m_1| + \cdots + |m_n|$, where the size of a module $m_i$, denoted by $|m_i|$, is given by $\Phi_i$ and the number of guards and assignment statements in such a module, which is polynomially bounded by the number of guarded commands multiplied by the number of variables controlled by $m_i$, that is, polynomially bounded by $(|I_i| + |U_i|) * |\Phi_i|$.

For technical reasons, we introduce for every module $m_i = (\Phi_i, I_i, U_i)$ an auxiliary guarded command, $g_i^{\mathbf{skip}}$ given by:

$$g_i^{\mathbf{skip}} = \bigwedge_{g \in U_i} \neg guard(g) \rightsquigarrow \mathbf{skip}$$

Thus, executing $g_i^{\mathbf{skip}}$ leaves all values for the variables in $\Phi_i$ unchanged.[1]

Given a module $m_i = (\Phi_i, I_i, U_i)$ and a valuation $v$, we define $enabled_i(v)$ as the set of update guarded commands that are enabled at $v$, under the proviso that, if none of the guarded commands in $U_i$ is enabled, $g_i^{\mathbf{skip}}$ will be. Formally,

$$enabled_i(v) = \{g \in U_i \cup \{g_i^{\mathbf{skip}}\} : v \models guard(g)\}.$$

Observe that, defined in this way, $enabled_i(v)$ is never empty, and will contain $g_i^{\mathbf{skip}}$ as the unique element if none of the guarded commands in $U_i$ is enabled.[2]

We now define a function that specifies the semantics of guarded commands. Let $g : \varphi \rightsquigarrow x'_1 := \psi_1; \cdots; x'_k := \psi_k$ be a guarded command in a module $m_i$ that controls the variables in $\Phi_i$. Then, $exec_i(g, v)$ denotes the propositional valuation for the variables $\Phi_i$ that would result from the execution of $g$ on $v$. Notice that $exec_i(\cdots)$ only gives a valuation for

[1] The command $g_i^{\mathbf{skip}}$ is introduced to facilitate a clear semantical definition of the behaviour of a reactive module if at some point of time none of its update guarded commands are enabled. There are, of course, other ways to resolve this issue. For example, one could also require that $U_i$ always contains $g_i^{\mathbf{skip}}$ (or one could give a considerably more complicated definition of $exec_i(g_i, v)$ than the one given in the main text).

[2] Recall that all initial guarded commands are enabled initially.

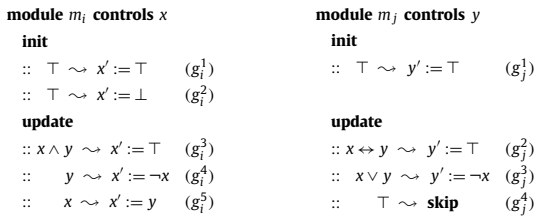



**Fig. 5.** The reactive modules $m_i$ and $m_j$ used in Example 1.

the variables $\Phi_i$ controlled by $m_i$; it does not specify the value of variables controlled by other modules. Formally, the function $exec_i(\cdots)$ is defined for guarded command $g = \varphi \rightsquigarrow x'_1 := \psi_1; \cdots; x'_k := \psi_k$ and valuation $v$ as follows:

$$exec_i(g, v) = (v_i \setminus ctr(g)) \cup \{x_i \in \{x_1, \ldots, x_k\} : v \models \psi_i\}.$$

The behaviour of an SRML arena is obtained by executing enabled guarded commands, one for each module, in a synchronous and concurrent way. A *joint guarded command* $J = (g_1, \ldots, g_n)$ is a profile of guarded commands, one for each module. We extend the notations from guarded commands to joint guarded commands in the obvious way. In particular we write

$$enabled(v) = enabled_1(v) \times \cdots \times enabled_n(v).$$

Moreover, we use

$$exec(J, v) = exec_1(g_1, v) \cup \cdots \cup exec_n(g_n, v)$$

to denote the execution of a joint guarded command $J = (g_1, \ldots, g_n)$ at valuation $v$. Then, the *deterministic execution* of an SRML arena proceeds in rounds. In the first round, the choices available to a module $m_i$ correspond to the initialisation guarded commands $I_i$ of $m_i$; the module selects one of these for execution; call this guarded command $g_i$. The result of the choice made by $m_i$ is defined to be $exec_i(g_i, v_\perp)$. The collection of choices made by all players then defines the initial valuation that appears in the run. In subsequent rounds, the choices available to a module $m_i$ correspond to the update guarded commands that are enabled for execution by the valuation that was produced on the previous round. Again, each player selects one such enabled guarded command for execution, and the collective result of these choices defines the next valuation that appears in the run, and so on. More formally, the deterministic execution of an SRML arena $A = (N, \Phi, m_1, \ldots, m_n)$ produces a run $\boldsymbol{\rho} : \mathbb{N} \to V$ such that, for some joint guarded command $J \in I_1 \times \cdots \times I_n$, we have $\boldsymbol{\rho}[0] = exec(J, v_\perp)$ and, for all $t > 0$, there is some joint command $J \in enabled(\boldsymbol{\rho}[t-1])$ with $\boldsymbol{\rho}[t] = exec(J, \boldsymbol{\rho}[t-1])$. An SRML arena $A$ may allow different runs, depending on which joint guarded commands are selected for execution, and the set of runs an arena $A$ permits we will denote by $\mathbf{runs}(A)$.

An SRML arena $A = (N, \Phi, m_1, \ldots, m_n)$ can also be executed *nondeterministically*, in which case it produces a computation tree $\boldsymbol{\kappa}$. For any such computation tree there is some joint initial guarded command $J \in I_1 \times \cdots \times I_n$, such that $exec(J, v_\perp) \in \boldsymbol{\kappa}$ and, for all $v_0 \ldots v_k$ and $v_0 \ldots v_k v_{k+1}$ in $\boldsymbol{\kappa}$, there is some enabled $J$ in $enabled(v_k)$ with $exec(J, v_k) = v_{k+1}$. An arena $A$ may allow multiple computation trees, and the set of computation trees an arena $A$ allows we denote by $\mathbf{trees}(A)$. Since bisimilar trees satisfy the same set of both LTL and CTL formulae, to simplify the presentation of our results we may allow $\mathbf{trees}(A)$ – and $\mathbf{trees}(K)$, with $K$ a Kripke structure – to be closed under bisimulation [38].

To illustrate these concepts, consider the following example of an SRML arena.

**Example 1.** Let $\Phi = \{x, y\}$ and consider the SRML arena

$$A = (\{i, j\}, \{x, y\}, m_i, m_j),$$

where agent $i$ controls $x$ and agent $j$ controls $y$ and $m_i$ and $m_j$ are further specified as in Fig. 5. Thus,

$$\begin{aligned}
&enabled_i(\{x, y\}) = U_i && enabled_j(\{x, y\}) = U_j \\
&enabled_i(\ \{x\}\ ) = \{g_i^5\} && enabled_j(\ \{x\}\ ) = \{g_j^3, g_j^4\} \\
&enabled_i(\ \{y\}\ ) = \{g_i^4\} && enabled_j(\ \{y\}\ ) = \{g_j^3, g_j^4\} \\
&enabled_i(\ \ \emptyset\ \ ) = \{g_i^{\mathbf{skip}}\} && enabled_j(\ \ \emptyset\ \ ) = \{g_j^2, g_j^4\}
\end{aligned}$$

Furthermore, observe that $exec((g_i^1, g_j^1), v_\perp) = \{x, y\}$, $exec((g_i^3, g_j^3), \{x, y\}) = \{x\}$, $exec((g_i^5, g_j^3), \{x\}) = \{x\}$, and $exec((g_i^{\mathbf{skip}}, g_j^4), \emptyset) = \emptyset$. It follows that $\{x, y\}, \{x\}, \emptyset, \emptyset, \ldots$ is a run in $\mathbf{runs}(A)$. By a similar argument, it can also readily be appreciated that the (partial) computation tree in Fig. 4 depicts the initial part of a computation tree contained in $\mathbf{trees}(A)$. Fig. 6 further illustrates the transitions between valuations that are induced by enabled guarded commands.

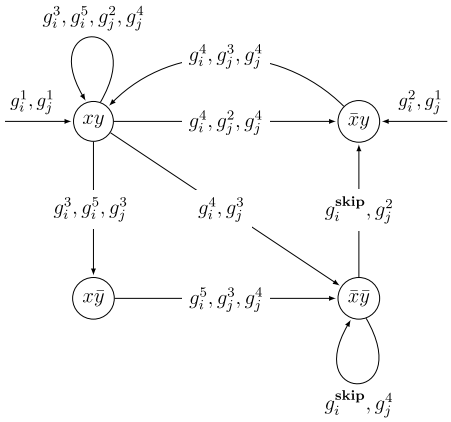



**Fig. 6.** Graphical depiction of the behaviour of the SRML Arena of Example 1. Here, e.g., $x\bar{y}$ represents $\{x\}$, the valuation that sets $x$ to true and $y$ to false. If an edge between valuations $v$ and $w$ is labelled with both a guarded command $g_i^k$ for player $i$ and a guarded command $g_j^m$ for player $j$, then $(g_i^k, g_j^m) \in \mathit{enabled}(v)$ and $\mathit{exec}_k(g_i^k, g_j^m, v) = w$.

### 4.2. Kripke-based and logic-based semantics

Above, we have presented a formal semantics for SRML, showing how arenas can be executed deterministically (leading to runs), or non-deterministically (leading to computation trees). We now present two further semantics, both of which are used extensively in what follows. In the first, we show how SRML arenas induce Kripke structures, while in the second, we see how the semantics of SRML arenas can be defined in terms of temporal logic formulae.

Whenever an enabled joint guarded command of an SRML arena is executed, the system it describes can be seen as transitioning from one state to the next. Moreover, as the set of enabled commands at a state only depends on the set of propositional set to true in that state, states with the same associated valuation can be viewed as identical. Thus, every SRML arena induces a finite Kripke structure. For instance, a comparison with Fig. 6 reveals that the Kripke structure in Fig. 3 models the behaviour of the SRML arena of Example 1 in this way.

At this point, it is useful to remind ourselves that one of the reasons for using SRML specifications is that they allow a *succinct* representation of models that capture the possible (infinite) computations of a distributed, concurrent, and multi-agent (computer) system. For technical reasons, however, in some cases it is useful to explicitly refer to the Kripke structure – i.e., transition system – that is induced by an arena $A = (N, \Phi, m_1, \ldots, m_n)$; such a structure we denote by $K_A$, which may in general be *exponential* in the size of $A$. Formally, we have:

**Lemma 2** *(Kripke-based semantics). For every arena $A$, there is a Kripke structure $K_A$ of size at most exponential in $|A|$ with the same set of runs and the same set of computation trees, i.e., with:*

$$\mathbf{runs}(A) = \mathbf{runs}(K_A) \quad \textit{and} \quad \mathbf{trees}(A) = \mathbf{trees}(K_A).$$

*Likewise, for every Kripke structure $K = (S, S^0, R, \pi)$, there exists an SRML arena $A_K$ containing a single SRML module $m_K = (\Phi \cup S, I_K, U_K)$ of linear size with respect to $|K|$ whose runs and computation trees* when restricted to $\Phi$ *are exactly those in $K$, i.e.,*

$$\mathbf{runs}(K) = \{\boldsymbol{\rho}|_\Phi : \boldsymbol{\rho} \in \mathbf{runs}(A_K)\} \quad \textit{and} \quad \mathbf{trees}(K) = \{\boldsymbol{\kappa}|_\Phi : \boldsymbol{\kappa} \in \mathbf{trees}(A_K)\}.$$

**Proof.** An algorithm to construct Kripke structures $K_A$ for SRML arenas $A$ is given in Fig. 12 in the Appendix.

To prove the second half of the lemma, we now provide a general explicit construction of an SRML module based on a given Kripke structure. Thus, let $K = (S, S^0, R, \pi)$ with $\pi : S \to V(\Phi)$. Define the SRML module $m_K$ for the Kripke structure $K$ as in Fig. 7. Observe that this construction introduces new variables, in addition to the variables $\Phi$ of the Kripke structure. Specifically, we introduce a new variable for each state of the Kripke structure.[3]

In the module in Fig. 7, $s_0' := \top$; means that the initial state is $s_0$. In the update rules the transition relation $R$ is modelled as follows: for each $(s, s') \in R$ we have a rule indicating that at $s$ we can choose $s'$ as a next state where propositions $p, \ldots$ hold. All other states are set to $\bot$ to indicate that the system is currently not in those states.

To see that the size of $m_K$ is linear in the size of $K$, observe that we have exactly one controlled variable for each state of the Kripke structure as well as exactly one **update** guarded command for each pair in $R$. On the other hand, the fact that both $K$ and $m_K$ have the same sets of runs and computation trees (when the trees induced by $A_K$ are restricted to the variables $\Phi$) immediately follows from the construction in Fig. 7. □

[3] Observant readers will note that in fact, we only need to introduce $O(\log_2 |S| + 1)$ new variables, but the construction of $m_K$ in this case would be much more convoluted.

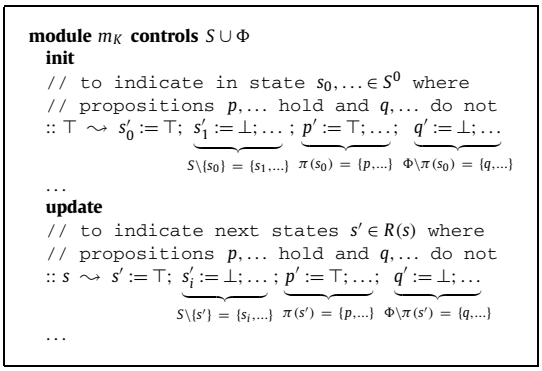



**Fig. 7.** Construction for generating SRML modules from Kripke structures.

We now present another lemma that will also be useful subsequently. The lemma shows, firstly, that for every arena $A$, it is possible to define an LTL formula $\text{TH}_{LTL}(A)$ that acts as the LTL *theory* of $A$, in the sense that the runs satisfying $\text{TH}_{LTL}(A)$ are exactly the runs of $A$. Formally, we have the following result:

**Lemma 3** *(Logic-based semantics). For every arena $A$ of size $|A|$, there is an LTL formula $\text{TH}_{LTL}(A)$ of size polynomial in $|A|$ such that for all $\boldsymbol{\rho} : \mathbb{N} \to 2^{\Phi}$,*

$$\boldsymbol{\rho} \in \mathbf{runs}(A) \quad \textit{if and only if} \quad \boldsymbol{\rho} \models \text{TH}_{LTL}(A).$$

*Moreover, $\text{TH}_{LTL}(A)$ can be computed in time polynomial in the size of $A$.*

**Proof.** Let $A = (N, \Phi, m_1, \ldots, m_n)$ be the given arena. We define the formula $\text{TH}_{LTL}(A)$ as the conjunction of two formulae, which roughly speaking define the effect of the initialisation and update commands in $A$, respectively:

$$\text{TH}_{LTL}(A) = \text{INIT}(A) \wedge \text{UPDATE}(A).$$

We begin by defining a temporal predicate $\text{UNCH}(\Psi)$, which asserts that the variables $\Psi \subseteq \Phi$ take the same value in the next state that they do in the present state (i.e., they remain unchanged):

$$\text{UNCH}(\Psi) = \bigwedge_{x \in \Psi} (x \leftrightarrow \mathbf{X}x).$$

Then, we define the effect of a single initialisation guarded command.

$$\text{INIT}_i(g = \top \rightsquigarrow x'_1 := b_1; \cdots; x'_k := b_k) = \Big(\bigwedge_{l=1}^{k} x_l \leftrightarrow b_l\Big) \wedge \Big(\bigwedge_{x \in \Phi_i \setminus ctr(g)} x \leftrightarrow \bot\Big).$$

The expression $\text{INIT}_i$ then captures the semantics of initialisation commands (we write $\oplus$ as an abbreviation of the "exactly one" operator – which is equivalent to the "exclusive-or" operator in the binary case):

$$\text{INIT}_i = \bigoplus_{g \in I_i} \text{INIT}_i(g) = \bigvee_{g \in I_i} \Big(\text{INIT}_i(g) \wedge \bigwedge_{g' \in I_i \setminus \{g\}} \neg \text{INIT}_i(g')\Big)$$

Then:

$$\text{INIT}(A) = \bigwedge_{i \in N} \text{INIT}_i.$$

Next we define the semantics of update rules. Again, we do this in two parts. We first define the effect of a single update guarded command:

$$\text{UPDATE}_i(g = \varphi \rightsquigarrow x'_1 := \psi_1; \cdots; x'_k := \psi_k) = \varphi \wedge \left(\bigwedge_{l=1}^{k} \psi_l \leftrightarrow \mathbf{X}x_l\right) \wedge \text{UNCH}(\Phi_i \setminus ctr(g)).$$

We then define the overall effect of $i$'s update commands:

$$\text{UPDATE}_i = \left(\bigwedge_{g \in U_i} \neg guard(g) \wedge \text{UNCH}(\Phi_i)\right) \vee \bigoplus_{g \in U_i} \text{UPDATE}_i(g).$$

Finally:

$$\textsc{Update}(A) = \mathbf{G} \bigwedge_{i \in N} \textsc{Update}_i$$

In order to show that $\rho \in \mathbf{runs}(A)$ if, and only if, $\rho|_\Phi \models \textsc{Th}_{LTL}(A)$, one proceeds by induction. Moreover, that $\textsc{Th}_{LTL}(A)$ is polynomial in the size of $A$ immediately follows from the construction of the LTL formula, and that it can be computed in time polynomial in the size of $A$ follows similarly. □

Hereafter, we will write $\textsc{Th}(A)$ for $\textsc{Th}_{LTL}(A)$. Moreover, we will say that a temporal logic formula $\varphi$ *characterises or represents* the behaviour of an arena $A$ if $\varphi$ is logically equivalent to $\textsc{Th}(A)$. It should also be easy to see that the constructions of $\textsc{Th}(A)$, for arenas, can be restricted to characterise the behaviour of (subsets of) single modules. In particular, we write $\textsc{Th}(m_i)$ for the formula that characterises the behaviour of module $m_i$, and write $\textsc{Th}(m_{-i})$ for the formula that characterises the behaviour of the set of modules $\{m_j : j \in N \text{ and } i \neq j\}$.

## 5. Reactive modules games

We can now introduce the game model we work with in the remainder of this article. The games we consider are called *reactive modules games* (RMGs) and have two components. The first component is an arena: this defines the players in the game, the variables they control, and the choices available to these players in every game state. The arena in an RMG plays a role analogous to that of a *game form* in conventional game theory [53, p. 201]: while it defines players and their choices, it does not specify the preferences of players. Preferences in RMGs are specified by the second component of an RMG: every player $i$ is assumed to be associated with a *goal* $\gamma_i$, which in RMGs will be a temporal logic formula. The idea is that players desire to see their goal satisfied. Moreover, as one would expect, they are indifferent between two outcomes that satisfy their goal, and indifferent between outcomes that do not achieve it.

Formally, an RMG $G$ is given by a structure:

$$G = (A, \gamma_1, \ldots, \gamma_n)$$

where:

- $A = (N, \Phi, m_1, \ldots, m_n)$ is an SRML arena with components as defined earlier;
- for all $i \in N$, the formula $\gamma_i$ is the *goal* of player $i$, represented as a temporal logic formula; in what follows, we consider both LTL and CTL as possible goal languages.

Games in which all players have goals expressed as LTL formulae are called *LTL RMGs*, while games in which players have goals expressed as CTL formulae are called *CTL RMGs*. Both types of game are played by each player $i$ selecting a *strategy* $\sigma_i$ that will define how to make choices over time. In the case of LTL RMGs strategies will be *deterministic*, whereas for CTL RMGs strategies are *non-deterministic*. (Formal definitions will be given shortly.) Then, once every player $i$ has selected a strategy $\sigma_i$, a *strategy profile* $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ results and the game has an *outcome*, which given the nature of the strategies will be:

- A run (an infinite word), denoted by $\rho(\vec{\sigma})$, for LTL RMGs;
- A Kripke structure, denoted by $K_{\vec{\sigma}}$, for CTL RMGs.

In either case, the outcome will determine whether or not each player's goal is or is not satisfied. Recall that LTL formulae are interpreted on runs and CTL formulae on Kripke structures. In order to simplify notation, whenever the game is clear from the context, we will write $\vec{\sigma} \models \varphi$ for both $\rho(\vec{\sigma}) \models \varphi$, if $\varphi \in \textsc{Ltl}$, and $K_{\vec{\sigma}} \models \varphi$, if $\varphi \in \textsc{Ctl}$. Although neither strategies nor outcomes $\rho(\vec{\sigma})$ and $K_{\vec{\sigma}}$ have been formally given yet (defined in the next sections), we are now in a position to define a preference relation $\succsim_i$ over outcomes for each player $i$ with goal $\gamma_i$. For strategy profiles $\vec{\sigma}$ and $\vec{\sigma}'$, we say that

$$\vec{\sigma} \succsim_i \vec{\sigma}' \quad \text{if and only if} \quad \vec{\sigma}' \models \gamma_i \text{ implies } \vec{\sigma} \models \gamma_i.$$

We say that player $i$ *strictly prefers* outcome $\sigma$ over $\sigma'$ if $\sigma \succsim_i \sigma'$ but not $\sigma' \succsim_i \sigma$, and is *indifferent* between $\sigma$ and $\sigma'$ whenever both $\sigma \succsim_i \sigma'$ and $\sigma' \succsim_i \sigma$. As defined above, players strictly prefer outcomes that satisfy their goals over outcomes that do not, and will be indifferent otherwise. It can easily be established that the preference relations $\succsim_i$ satisfy the standard requirements of reflexivity, transitivity, and completeness, so that they can play their conventional role in subsequent game-theoretic definitions (see, e.g., [53, p. 7]). Here, we state the lemma but omit the straightforward proof.

**Lemma 4.** *Each relation $\succsim_i$, as defined above, is reflexive, transitive, and complete.*

We now define the standard solution concept of Nash equilibrium for RMGs. For this we need one small additional piece of notation. Given a strategy profile $\vec{\sigma} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i, \sigma_{i+1}, \ldots, \sigma_n)$ and a strategy $\sigma_i'$ for player $i$, then by $(\vec{\sigma}_{-i}, \sigma_i')$ we will denote the strategy profile that is the same as $\vec{\sigma}$ except that the strategy for player $i$ is $\sigma_i'$, that is:

$$(\vec{\sigma}_{-i}, \sigma_i') = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n).$$

Then $\vec{\sigma}$ is said to be a *Nash equilibrium* of $G$ if for all players $i$ and all strategies $\sigma_i'$ for player $i$, we have

$$\vec{\sigma} \succsim_i (\vec{\sigma}_{-i}, \sigma_i').$$

Hereafter, we let $NE(G)$ be the set of (pure strategy) Nash equilibria for an RMG $G$. As is usually the case in game theory, it is possible that $NE(G) = \emptyset$ (there are no equilibria), or that $NE(G)$ contains just one, or multiple equilibria; there may even be infinitely many equilibria. If there are no equilibria, then the system is inherently *unstable* – this is one of the key system properties that we will be interested in checking, in the form of the Non-Emptiness problem, described below. If there are multiple equilibria, then this presents the players in the game with a *coordination problem*: they need to find some way of fixing on one of the equilibria. One way for players to do this is to look for *focal points*, that is, equilibria that "stand out" from others in the sense that they have certain distinctive features; see, e.g., [61]. In the RMGs setting, for example, an equilibrium that satisfied the goals of all players might be regarded as particularly distinctive. From the point of view of a system designer, the existence of multiple equilibria may not be an issue if all of these equilibria satisfy certain desirable properties. This is another key computational problem that we study, in the form of the A-Nash decision problem, described below.

Before studying LTL and CTL RMGs in detail, we should make explicit some important assumptions that underpin our work. The game-theoretic interpretation that we place on RMGs essentially corresponds to the standard game-theoretic model of non-cooperative strategic form games, and in common with this work, we make two assumptions. First, we assume that the game is common knowledge to all players, i.e., all the players know the arena $A$ and goals $\gamma_i$, and know that other players know this, etc. Second, we assume that all players act rationally, taking into account the fact that all other players act rationally. We emphasise that these are common (although not universal) assumptions in game theory. Dropping them would be an interesting topic for future work, but beyond the present paper.

### 5.1. LTL reactive modules games

Players in an LTL RMG possess LTL goals and choose deterministic strategies. The interaction between (i.e., parallel composition of) such strategies will determine a unique run of the arena, that is, an infinite sequence of states/valuations, which can be used as the LTL model against which to interpret players' goals.

Let us fix some notation. We will write $\Phi_{-i}$ for $\Phi \setminus \Phi_i$ and let $V_i$ (respectively, $V_{-i}$) denote the set of valuations to variables in $\Phi_i$ (respectively, $\Phi_{-i}$).

For LTL RMGs, we model strategies as finite state machines with output – technically, deterministic Moore machines. The input language of such a machine representing a strategy for player $i$ corresponds to the choices of other players, i.e., assignments for variables that other players control, $V_i$, while outputs are variable assignments for the player implementing the strategy, i.e., $V_i$.

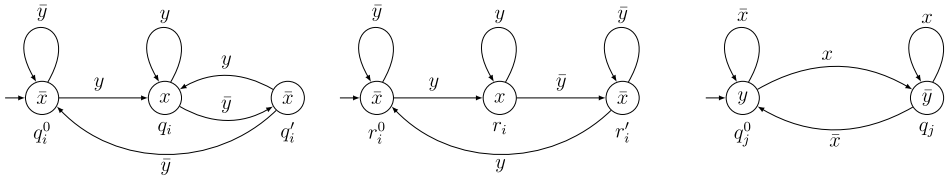
Representing strategies as finite state machines has a number of advantages. First, as the name suggests, it is a *finite* representation scheme. A more mathematically abstract representation for strategies would be model them as functions $f_i : V_{-i}^* \to V_i$, i.e., functions that map a sequence of choices for other players to a choice for player $i$. The problem with this representation from a computational point of view is that the domain of such functions is infinite, which raises substantial difficulties if we want to study decision problems that take such strategies as input. Second, while it may appear that finite state machine representations are weaker than the more mathematically abstract representation, this is in fact not the case if we are only concerned with players whose goals are expressed as temporal logic formulae. In such cases, all we need are finite state machine strategies: the existence of a "general" strategy for a player that will accomplish a player's goal implies the existence of a finite state machine strategy that will do the same job. Finally, the use of finite state machine strategies is, in fact, standard in the literature on iterated games [6]. These and related questions are discussed in more detail in [29].

Now, given $A = (N, \Phi, m_1, \ldots, m_n)$, an SRML arena, we define a *deterministic strategy* for module $m_i = (\Phi_i, I_i, U_i)$ as a structure $\sigma_i = (Q_i, q_i^0, \delta_i, \tau_i)$, where

- $Q_i$ is a finite and non-empty set of *states*,
- $q_i^0 \in Q_i$ is the *initial* state,
- $\delta_i : Q_i \times V_{-i} \to Q_i$ is a *transition function*, and
- $\tau_i : Q_i \to V_i$ is an *output function*.

Note, in particular, that not all strategies for a given module will comply with that module's specification. For instance, if the only guarded command of a module $m_i$ has the form $\top \rightsquigarrow x' := \bot$, then a strategy for $m_i$ should not prescribe $m_i$ to set $x$ to true under any contingency. Thus, given an arena $A = (N, \Phi, m_1, \ldots, m_n)$, a strategy $\sigma_i = (Q_i, q_i^0, \delta_i, \tau_i)$ for module $m_i$ is *consistent with* $m_i$ if the following two conditions are satisfied:

(*i*) for the initial state $q_i^0$, we have $\tau_i(q_i^0) = exec_i(g, v_\bot)$ for some $g \in I_1 \times \cdots \times I_n$, and
(*ii*) for all $q, q' \in Q_i$ and $v = v_i \cup v_{-i} \in V$ such that $\delta(q, v_{-i}) = q'$ and $v_i = \tau_i(q)$, we have $\tau_i(q') = exec_i(g_i, v)$ for some $g_i \in enabled_i(v)$.




(a) Strategy $\sigma_i$ (b) Strategy $\sigma_i'$ (c) Strategy $\sigma_j$

**Fig. 8.** Three deterministic machine strategies. The states are represented by the vertices. The labels inside the vertices denote the truth values the player sets its propositional variables to when in the corresponding state. An arrow between two vertices indicates a transition the machine makes on reading the choices for the propositional variables controlled by the other players as given by the label.

Note that, defined in this way, strategies implicitly have *perfect information*: the player can completely and correctly perceive the choices of other players in the game. Clearly this is ultimately too strong an assumption in many settings, but it will suffice as a first approximation. Also, we are implicitly assuming strategies have finite memory (although we impose no bounds on memory size).

If a strategy $\sigma_i$ is consistent with module $m_i$, we simply say that $\sigma_i$ is a *strategy for* $m_i$. Given an SRML arena $A = (N, \Phi, m_1, \ldots, m_n)$, we say that a strategy profile $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ is *consistent with A* if each $\sigma_i$ is consistent with $m_i$. In the remainder of the article, we restrict our attention to consistent strategies, and so from here on, the term "strategy" should be understood to mean "consistent strategy".

Finally, because each strategy $\sigma_i$ is *deterministic*, there is a *unique* run induced by $\vec{\sigma}$, which we denote by $\boldsymbol{\rho}(\vec{\sigma})$.

**Example 5.** Consider once more the SRML arena of Example 1 and the (deterministic) strategies depicted in Fig. 8. First consider the strategy profile $(\sigma_i, \sigma_j)$, which gives rise to the computation run that starts with:

$$\{y\}, \{x, y\}, \{x\}, \emptyset, \{y\}, \ldots$$

Here, the machine strategy $\sigma_i$ subsequently visits the states $q_i^0, q_i, q_i, q_i', q_i^0, \ldots$ and $\sigma_j$ the states $q_j^0, q_j^0, q_j, q_j, q_j^0, \ldots$. Both strategies comply with their respective modules. For instance, the transition $\delta(q_i, \emptyset) = q_i'$ in $\sigma_i$ is warranted on basis of the guarded command $g_i^5$, i.e., $x \rightsquigarrow x' := y$. Notice that $\{x\} \not\models y$ and, as $exec_i(g_i^5, \{x\}) = \emptyset = \tau(q_i')$, condition $(ii)$ above is satisfied. By contrast strategy $\sigma_i'$ does not comply with $m_i$. To see this, consider the transition $\delta(r_i', \{y\}) = r_i^0$ and observe that $g_i^4$, i.e., $y \rightsquigarrow x' := \neg x$, is the only update guarded command that is enabled at valuation $\{y\}$. Yet, $\tau_i(r_i^0) = \emptyset$, whereas, as $\{y\} \models \neg x$ and thus $exec_i(g_i^4, \{y\}) = \{x\}$, condition $(ii)$ above would require that $\tau_i(r_i^0) = \{x\}$.

### 5.2. CTL reactive modules games

Players in a CTL RMG possess CTL goals and have non-deterministic strategies at their disposal. Intuitively, whereas *deterministic* strategies can be seen as *controllers*, which resolve the non-determinism of the system by choosing a single computation path, *non-deterministic* strategies should be understood as *supervisors*, which disable undesirable behaviours of the system but leave the remaining choices to be selected/executed in a non-deterministic manner.

Thus, whereas a profile $\vec{\sigma}$ of deterministic strategies can be associated with a unique and infinite sequence of states/valuations of the system, a profile of non-deterministic strategies can be associated with a unique and infinite tree of system states/valuations – where only the non-deterministic choices made by $\vec{\sigma}$ are left in the tree.

The definition of non-deterministic strategies is a simple generalisation of that of deterministic ones, where only the *transition function* is redefined, as follows: $\delta_i \colon Q_i \times V_{-i} \to 2^{Q_i} \setminus \{\emptyset\}$. Finally, the definitions of outcome (that is, of $K_{\vec{\sigma}}$), consistency, and Nash equilibrium with respect to profiles $\vec{\sigma}$ of consistent non-deterministic strategies are straightforwardly extended.

### 5.3. Example: rational mutual exclusion

*Mutual exclusion algorithms* (*MEA*s) play a fundamental role in distributed computing. As the name suggests, they are designed to ensure mutual exclusion, i.e., that in concurrent settings two processes or agents cannot simultaneously enter a "*critical region*" (*CR*). Classic examples of MEAs are Peterson's algorithm [55], Lamport's bakery algorithm [44], and Ricart and Agrawala's improvement of the latter [59]. We now show how such an algorithm can be defined using SRML, how it ensures mutual exclusion, and how rational action can moreover guarantee *non-starvation*. Non-starvation is the desirable property that no process or agent is perpetually denied access to a shared resource; in the context of MEAs, the resource in question is access to the CR.

In a *ring-based MEA*, the agents are organised in a cycle along which a "token" is passed – similar to a distributed implementation of a round-robin scheduler [40]. Possession of the token signifies exclusive permission to enter the CR, i.e., an agent can enter the CR only if she is in possession of the token. The modules in Fig. 9 specify a SRML arena

**module** $rbme_0$ **controls** $x_0, y_0$
**init**
$:: \top \rightsquigarrow x'_0 := \bot, y'_0 := \bot$
**update**
$:: y_2 \vee x_0 \rightsquigarrow x'_0 := \top$
$:: y_2 \vee x_0 \rightsquigarrow x'_0 := \bot, y'_0 := \top$
$:: y_0 \rightsquigarrow y'_0 := \bot$

**module** $rbme_1$ **controls** $x_1, y_1$
**init**
$:: \top \rightsquigarrow x'_1 := \bot, y'_1 := \bot$
**update**
$:: y_0 \vee x_1 \rightsquigarrow x'_1 := \top$
$:: y_0 \vee x_1 \rightsquigarrow x'_1 := \bot, y'_1 := \top$
$:: y_1 \rightsquigarrow y'_1 := \bot$

**module** $rbme_2$ **controls** $x_2, y_2$
**init**
$:: \top \rightsquigarrow x'_2 := \bot, y'_2 := \top$
**update**
$:: y_1 \vee x_2 \rightsquigarrow x'_2 := \top$
$:: y_1 \vee x_2 \rightsquigarrow x'_2 := \bot, y'_2 := \top$
$:: y_2 \rightsquigarrow y'_2 := \bot$

**Fig. 9.** Modules for the ring-based mutual exclusion algorithm.

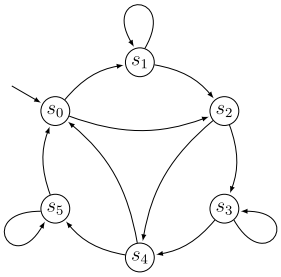



**Fig. 10.** Kripke structure for the arena $A^{rbme}$.

**Table 1**
Description of the behaviour of the agents with respect to Fig. 10. A ↻-symbol indicates that the respective step can be repeated any number of times.

| State | $x_0$ | $y_0$ | $x_1$ | $y_1$ | $x_2$ | $y_2$ | | Annotation |
|---|---|---|---|---|---|---|---|---|
| $s_0$ | 0 | 0 | 0 | 0 | 0 | 1 | | 2 passes token on to agent 0 |
| $s_1$ | 1 | 0 | 0 | 0 | 0 | 0 | ↻ | 0 has token and is in CR |
| $s_2$ | 0 | 1 | 0 | 0 | 0 | 0 | | 0 passes token on to agent 1 |
| $s_3$ | 0 | 0 | 1 | 0 | 0 | 0 | ↻ | 1 has token and is in CR |
| $s_4$ | 0 | 0 | 0 | 1 | 0 | 0 | | 1 passes token on to agent 2 |
| $s_5$ | 0 | 0 | 0 | 0 | 1 | 0 | ↻ | 2 has token and is in CR |
| $s_0$ | 0 | 0 | 0 | 0 | 0 | 1 | | 2 passes token on to agent 0 |

$$A^{rbme} = (\{0,1,2\}, \{x_0, x_1, x_2, y_0, y_1, y_2\}, rbme_0, rbme_1, rbme_2),$$

which implements a three-agent distributed ring-based MEA (cf., [17], pp. 474–475). The algorithm easily extends to settings with any finite number of agents.

Thus, in $A^{rbme}$ each agent $i$ controls two variables, $x_i$ and $y_i$. Variable $x_i$ being true means that agent $i$ has the token and enters the CR, whereas $y_i$ indicates that agent $i$ has the token and passes it to the next agent $i+1$. (We assume arithmetic modulo 3 throughout the example). In the initial state, agent 2 has the token and passes it on to agent 0; no agent is in the CR. On possession of the token, each agent can decide whether to enter the CR or to pass on the token. Once an agent $i$ is in the CR, i.e., if $x_i$ is true, it can remain there as long as it suits him. On exiting the CR, i.e., if $\neg x_i \wedge y_i$ holds, agent $i$ immediately passes the token on to agent $i+1$. The behaviour of the system is described by the modules' specifications. The Kripke structure for arena $A^{rbme}$ is depicted in Fig. 10 and Table 1 further illustrates the dynamics of the system.

Observe that, in every state – and thus at every time and on all runs of $A^{rbme}$ – at most one agent has the token and can enter the CR. Hence, the formula

$$\mathbf{G} \bigwedge_{i\in\{0,1,2\}} \big((x_i \vee y_i) \rightarrow (\neg x_{i+1} \wedge \neg y_{i+1} \wedge \neg x_{i+2} \wedge \neg y_{i+2})\big)$$

is satisfied, which signifies that the protocol guarantees mutual exclusion.

The system, however, still allows for many different runs, some of which are undesirable, e.g., if at some point some agent enters the CR and does not leave it ever afterwards. Such behaviour would violate the desired starvation-free property. This is even the case if we assume that each agent $i$ has as a goal to have the token (and thus the ability to enter the CR) infinitely often, i.e., $\gamma_i = \mathbf{GF}(x_i \vee y_i)$, and we restrict our attention to runs that are sustained by a Nash equilibrium. If all agents adopt a strategy in which they enter the CR and remain there for ever when given the opportunity, a Nash equilibrium results in which two agents can never enter the CR.

If, on the other hand, each agent $i$ has as a goal the CTL formula $\gamma_i = \mathbf{GF}\neg x_i$, i.e., to *be out of the CR* infinitely often, then each of them can guarantee its own goal to be satisfied by simply adopting any strategy according to which it exits the CR after a finite period of time in possession of the token. Thus, no agent will perpetually be excluded from having

the opportunity to enter the CR and each agent *rationally* follows a starvation-free protocol. Under these preferences, a powerful property of our game-like specification of the ring-based mutual exclusion algorithm is that on *all* runs that are sustained by a Nash equilibrium, the distributed system is guaranteed to be starvation-free! Thus, *safety* (mutual exclusion) and *starvation-freeness* for the distributed system can all be uniformly dealt with within RMGs.

### 5.4. Example: deferred acceptance

The deferred acceptance algorithm, as proposed in 1962 by David Gale and Lloyd Shapley [24], has been tremendously influential both in theory and in practice. The power of this two-sided allocation scheme lies in its very simplicity and its ramifications have numerous and various practical applications to, e.g., kidney exchange, school choice, and the assignment of doctors to hospitals (also see [60,27,45]).

Agents are divided in two disjoint finite groups, m-agents and f-agents, and each agent has preferences over the agents in the other group. The goal is to find a *stable matching*, that is, a pairing of m-agents and f-agents for which there are no matched pairs $(m_i, f_i)$ and $(m_j, f_j)$ such that $m_i$ and $f_j$ prefer one another to $f_i$ and $m_j$, respectively. The *deferred acceptance algorithm (DA)* is a scheme in which each m-agent proposes to one of his most preferred f-agents. If the offer is accepted the engagement stands until a more preferred m-agent comes along and proposes to f-agent. If the offer is rejected, the m-agent strikes the f-agent from his list and proposes to one of his next most preferred f-agents. The algorithm is guaranteed to terminate and to yield a stable matching.

We can model and reason about the DA setting within the RMG framework. We assume there to be *m-agents* $m_1, \ldots, m_k$ and an equal number of *f-agents* $f_1, \ldots, f_k$. Every m-agent $m_i$ controls a set of variables $\Phi_{m_i} = \{x_{i1}, \ldots, x_{ik}, z_{i1}, \ldots, z_{ik}\}$, where $x_{ij}$ indicates that m-agent $m_i$ proposes to f-agent $f_j$, while $z_{ij}$ records that $m_i$ has proposed to $f_j$ in the past. Similarly, every f-agent $f_j$ controls a set of variables $\Phi_{f_j} = \{y_{1j}, \ldots, y_{kj}\}$, where $y_{ij}$ signifies that f-agent $f_j$ provisionally accepts $m_i$'s proposal.

In our setting, each agent has dichotomous preferences, dividing the agents of the other group into those that are approved and those that are not approved. Let for every m-agent $m_i$ the set of indices of approved f-agents be denoted by $F_i$ and for every f-agent $f_j$ the set of indices of approved m-agents by $M_j$. That is, $F_i = \{j : \ m_i \text{ approves } f_j\}$ and $M_j = \{i : f_j \text{ approves } m_i\}$. The preferences of m-agent $m_i$ and f-agent $f_j$ are then given by

$$\gamma_{m_i} = \mathbf{AFAG}\Big(\bigvee_{j \in F_i} y_{ij}\Big) \qquad \gamma_{f_j} = \mathbf{AFAG}\Big(\bigvee_{i \in M_j} y_{ij}\Big),$$

respectively, that is, every m-agent wishes to be eventually for ever matched with one of his approved f-agents, and every f-agent with one of her approved m-agents. In our setting with dichotomous preferences, stability means that there is no m-agent $m_i$ and no f-agent $f_j$ such that $f_j$ is one of $m_i$'s approved f-agents and $m_i$ is one of $f_j$'s approved m-agents, and neither $m_i$ is matched to one of his approved f-agents and $f_j$ is not matched with one of her approved m-agents. If $j \in F_{m_i}$ and $i \in M_j$, we say that $(i, j)$ is a *blocking pair* and let $X^*$ denote the set of blocking pairs. Then,

$$\varphi_{stability} = \bigwedge_{(i,j) \in X^*} \mathbf{AFAG}\Big(\bigvee_{l \in F_i} y_{il} \vee \bigvee_{l \in M_j} y_{lj}\Big)$$

signifies that on every computation path eventually stable matchings will obtain for ever after, a desirable property.

The execution of the DA algorithm takes place over several rounds. We assume that, in every odd round, m-agents that are not engaged can (non-deterministically) propose to any of the f-agents he has not proposed to before. By contrast, in every even round each of the f-agents can (non-deterministically) engage any m-agent that proposed to her in the previous round or, alternatively, decide to stay with her current fiancé, if any, and remain single otherwise. Observe that once an f-agent is engaged, she will be engaged for ever after, be it not necessarily to the same m-agent. These provisions are laid down by the modules depicted in Fig. 11. It is worth observing that these specifications are independent of the agents' preferences and that they guarantee that none of the agents will be engaged with more than one agent at any one time.

Obviously, $\varphi_{stability}$ does not hold in every computation tree of the resulting RMG $G_{DA}$: an f-agent may very well accept an unfavourable offer and stick to it, even if one of her approved m-agents is rejected by all of his approved f-agents. More surprisingly, $\varphi_{stability}$ does hold not even hold in every run that is sustained by a Nash equilibrium. To see this, consider the case with two m-agents, $m_1$ and $m_2$, and two f-agents, $f_1$ and $f_2$, such that both m-agents approve of $f_1$, but not of $f_2$, and both f-agents approve of $m_1$, but not of $m_2$. Thus, $m_1$ and $f_i$ are the only blocking pair. Let, furthermore, $m_1$ and $m_2$ adopt strategies in which they propose in the first round to $f_2$ and $f_1$, respectively, and $f_1$ one in which she invariably accepts offers from $m_2$ only. Then, if $f_2$ accepts $f_1$'s proposal in the first round, a non-stable matching will prevail ever after, that is, $\varphi_{stability}$ will fail to hold forever. Even though $m_1$'s and $f_1$'s strategies may seem irrational and leads to neither the former's nor the latter's goal being satisfied, they cannot offset each others' folly by choosing another strategy. As, moreover, $m_2$ and $f_2$ are fully satisfied with the outcome, it follows that the strategy profile described above is a Nash equilibrium.

If, however, the situation had been different and all m-agents, rather than wishing to end up with an approved f-agent, weaken their goals to and aim to have at least *proposed* to all approved f-agents before proposing to a non-approved f-agent, all (non-deterministic) computations sustained by a Nash equilibrium will satisfy $\varphi_{stability}$. To appreciate this, assume that

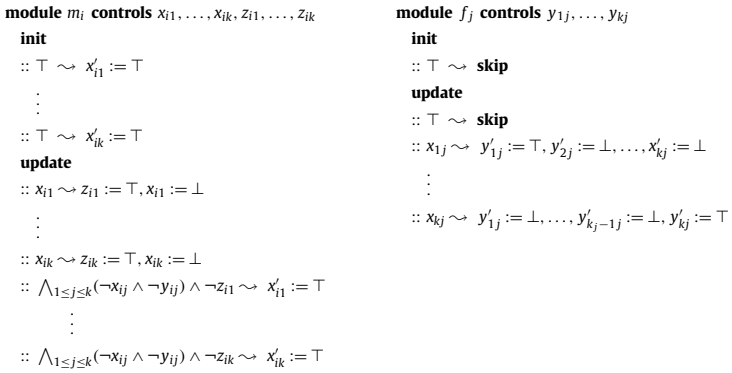



**Fig. 11.** Modules for the m-agents $m_i$ (left) and f-agents $f_j$ (right) for the deferred acceptance setting.

some blocking pair $m_i$ and $f_j$ ends up to be forever unmatched with any of their approved agents. Then, either $m_i$ has successfully proposed to an non-approved f-agent before proposing to $f_j$ or $f_j$ has declined a proposal by $m_i$. In the former case, $m_i$ can achieve his goal by diverting to strategy in which he proposes to his approved f-agents before proposing to the other agents. In the latter case, $f_j$ would have obtained a better outcome if she had accepted $m_i$'s proposal and stuck with it.

## 6. Equilibrium analysis

In this section, we study the following game-theoretic decision problems for RMGs:

- Realizability: whether a player can guarantee the satisfaction of a temporal formula within a given arena;
- Non-emptiness: whether a given game has at least one Nash equilibrium;
- NE-membership: whether a strategy profile forms a Nash equilibrium; and
- E-Nash and A-Nash: whether a temporal logic formula can be satisfied on some or on all Nash equilibria of a given game.

In order to solve the problems above mentioned, we first note that, similar to the case of Kripke structures, any strategy $\sigma_i$ for a player $i$ can be specified by an srml module. We will describe the construction of modules for non-deterministic strategies; clearly, the construction for deterministic strategies appears as a, simpler, special case.

Let $m(\sigma_i) = (\Phi'_i, I'_i, U'_i)$ be the module specifying some $\sigma_i = (Q_i, q_i^0, \delta_i, \tau_i)$ for a module $m_i = (\Phi_i, I_i, U_i)$. With each state $q_i \in Q_i$, we associate a fresh propositional variable, which, for presentational convenience, we will denote by $q_i$ as well. The module $m(\sigma_i)$ then controls the variables $\Phi'_i = \Phi_i \cup Q_i$. The **init** part $I'_i$ of $m_i(\sigma_i)$ contains a single guarded command:

$$\top \rightsquigarrow q_i^0 := \top; \underbrace{x_1 := \top; \cdots; x_k := \top}_{\text{where } \tau_i(q_i^0) = \{x_1, \ldots, x_k\}} .$$

We then use the **update** commands $U'_i$ of the module $m_i(\sigma_i)$ to encode both the output function $\tau_i$ and the transition function $\delta_i$. The **update** part contains for all valuations $v_i \in v$ and $v_{-i} \in V_{-i}$ and all $q, r \in Q_i$ such that $r \in \delta_i(q, v_{-i})$ and $\tau_i(q) = v_i$ an update guarded command – and recall that we have defined $\chi_v = \bigwedge_{x \in v} x \wedge \bigwedge_{x \notin v} \neg x$ – given by

$$\chi_v \wedge q \rightsquigarrow q := \bot; r := \top; \underbrace{x_1 := \top; \cdots; x_k := \top}_{\tau_i(r) = \{x_1, \ldots, x_k\}}; \underbrace{x_{k+1} := \bot; \cdots; x_l := \bot}_{\Phi_i \setminus \tau_i(r) = \{x_{k+1}, \ldots, x_l\}},$$

if $q \neq r$, or, otherwise, an update guarded command given by

$$\chi_v \wedge q \rightsquigarrow r := \top; \underbrace{x_1 := \top; \cdots; x_k := \top}_{\tau_i(r) = \{x_1, \ldots, x_k\}}; \underbrace{x_{k+1} := \bot; \cdots; x_l := \bot}_{\Phi_i \setminus \tau_i(r) = \{x_{k+1}, \ldots, x_l\}} .$$

Note that $m(\sigma_i)$ has one initialisation rule and one rule for each transition in $\delta_i$. Then, $m(\sigma_i)$ is of size linear in $|\sigma_i|$. That the behaviour of $m(\sigma_i)$ is exactly as the behaviour of $\sigma_i$ is obvious from the construction. Moreover, using Lemmas 2 and 3, and the fact that the size of $m(\sigma_i)$ is linear in $|\sigma_i|$, analogous results with respect to strategies and strategy profiles are obtained.

In particular, given an arena $A = (N, \Phi, m_1, \ldots, m_n)$ and a strategy $\sigma_i$, we can define the Kripke structure and runs induced by $\sigma_i$ on $A$ by replacing $m_i$ with $m(\sigma_i)$, that is, induced by the semantics of the srml arena $A_{\sigma_i} = (N, \Phi \cup Q_i, m_1, \ldots, m(\sigma_i), \ldots, m_n)$. The outcome with respect to some $\vec{\sigma}$, then, is given by the Kripke structure or runs associated with

the SRML arena $A_{\vec{\sigma}} = (N, \Phi \cup \bigcup_{i \in N} Q_i, m(\sigma_1), \ldots, m(\sigma_n))$. Since we are interested only in the computation runs and trees with respect to $\Phi$ associated with both $K_{\vec{\sigma}}$ and $A_{\vec{\sigma}}$, let us write $\mathbf{runs}(K_{\vec{\sigma}})|_\Phi$ for the restriction with respect to $\Phi$, that is for the set $\{\boldsymbol{\rho}|_\Phi : \boldsymbol{\rho} \in \mathbf{runs}(K_{\vec{\sigma}})\}$, and similarly for the set of computation trees. Then, once $\vec{\sigma}$ is specified in SRML, the following result immediately follows from Lemma 2.

**Lemma 6** *(SRML semantics of strategies). Let $\vec{\sigma}$ be some strategy profile with respect to an arena $A$. Then, there are SRML modules $m(\sigma_1), \ldots, m(\sigma_n)$, where each SRML module $m(\sigma_i)$ is of linear size in $|\sigma_i|$ such that*

$$\mathbf{runs}(K_{\vec{\sigma}})|_\Phi = \mathbf{runs}(A_{\vec{\sigma}})|_\Phi \quad \textit{and} \quad \mathbf{trees}(K_{\vec{\sigma}})|_\Phi = \mathbf{trees}(A_{\vec{\sigma}})|_\Phi.$$

Notice that in the case of strategy profiles containing non-deterministic strategies, the behaviour of such strategies is very easily handled by the semantics of the modules: non-determinism is simply captured by the choices that the module representing a particular strategy has at its disposal.

Finally, since strategy profiles can be specified (in linear size) as arenas in SRML, while preserving their induced sets of runs and computation trees, we can define a temporal logic-based theory for strategy profiles. Note, however, that outcomes of this new arena, $A_{\vec{\sigma}}$, are now with respect to valuations in the set $\Phi \cup \bigcup_{i \in N} Q_i$ rather than with respect to valuations in $\Phi$ only. One can, nevertheless, restrict outcomes to their projection over valuations in $\Phi$. A simple corollary of Lemma 6, using Lemma 3, is the following result.

**Corollary 7** *(Logic theory of strategies). Let $A = (N, \Phi, m_1, \ldots, m_n)$ be an arena. For every strategy profile $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$, of size $|\vec{\sigma}|$, for $A$ there is an LTL formula $\mathrm{TH}_{LTL}(A_{\vec{\sigma}})$, of size polynomial in $|\vec{\sigma}|$, such that*

$$\boldsymbol{\rho}|_\Phi \in \mathbf{runs}(K_{\vec{\sigma}})|_\Phi \quad \textit{if and only if} \quad \boldsymbol{\rho}|_\Phi \models \mathrm{TH}_{LTL}(A_{\vec{\sigma}}).$$

With these results in place, we can now start to investigate the decision problems discussed at the beginning of this section, for both the linear and branching time case.

### 6.1. On the complexity of LTL RMGs

The first problem we study is REALIZABILITY for LTL RMGs, which asks whether a player has a strategy to achieve its temporal logic goal with the context of a given arena, no matter how the other agents behave.

REALIZABILITY
*Given:* RMG $G$, player $i$, and LTL formula $\varphi$.
*Question:* Does there exist a strategy $\sigma_i$ such that for all $\vec{\sigma}_{-i}$ it holds that $\boldsymbol{\rho}(\vec{\sigma}_{-i}, \sigma_i) \models \varphi$?

**Proposition 8.** REALIZABILITY *for LTL RMGs is 2EXPTIME-complete. It is 2EXPTIME-hard for 2-player games, and PSPACE-complete for 1-player games.*

**Proof.** For membership we reduce REALIZABILITY to the synthesis problem with respect to reactive environments, a problem known to be 2EXPTIME-complete for LTL specifications [42]. First, let $\mathrm{TH}(m_i)$ be an LTL formula characterising the module for player $i$ and let $\mathrm{TH}(m_{-i})$ be an LTL formula characterising the modules of all other players in $G$ – i.e., an LTL formula for the SRML system comprising the modules in $\{m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_n\}$. (We do not give the construction for $\mathrm{TH}(m_i)$, which should be immediate from results given above.) Then, it should be easy to see that the LTL formula $\varphi$ is realizable in the context of LTL SRML games if, and only if, the following LTL formula can be synthesised with respect to any reactive environment:

$$\mathrm{TH}(m_i) \wedge (\mathrm{TH}(m_{-i}) \rightarrow \varphi)$$

provided that the environment (the coalition of players in $N \setminus \{i\}$) controls the variables in $\Phi \setminus \Phi_i$ and the system (player $i$) controls the variables in $\Phi_i$.

Note that since in order to solve the REALIZABILITY problem we checked whether the LTL formula $\mathrm{TH}(m_i) \wedge (\mathrm{TH}(m_{-i}) \rightarrow \varphi)$ can be synthesised, and the size of such a formula is polynomial in the sizes of both the LTL formula $\varphi$ and the game $G$, then it follows that REALIZABILITY is 2EXPTIME in the sizes of both the game $G$ and the input formula $\varphi$.

For hardness, we first reduce LTL games (also called control games for LTL [5]) to an instance of REALIZABILITY with 3 players/modules (for the sake of simplicity in the presentation), and then such an instance to a game with only 2 modules. Now, let us introduce LTL games. An LTL game is a two-player zero-sum game given by $(G_{LTL}, \varphi, v_0)$ where $\varphi$ is an LTL formula over a set $2^\Sigma$, and

$$G_{LTL} = (V_G, V_0, V_1, \gamma : V \rightarrow 2^V, \mu : V \rightarrow 2^\Sigma)$$

is a graph with vertices in $V_G$ which are partitioned in player 0 vertices $V_0$ and player 1 vertices $V_1$. The transitions of the graph are given by $\gamma$; if $v' \in \gamma(v)$, for some $v, v' \in V_G$, then there is a transition from $v$ to $v'$, which we may assume is labelled by $v'$. Each vertex $v$ has a set of properties $P$ associated to it, which are given by $\mu$; thus $P \in 2^{\Sigma}$ holds in vertex $v$ if $P = \mu(v)$. The graph $G_{LTL}$ is assumed to be total: for every $v \in V_G$, we have $\gamma(v) \neq \emptyset$.

The game is played for infinitely many rounds by each player choosing a successor vertex whenever it is their turn: player 0 plays in vertices in $V_0$ and player 1 plays in vertices in $V_1$. The game starts in the initial vertex $v_0 \in V_G$. Playing the game defines an infinite sequence of vertices $w = v_0, v_1, v_2, \ldots$, such that for each $v_k$, with $k \in \mathbb{N}$, we have $v_{k+1} \in \gamma(v_k)$. An LTL game is won by player 0 if $w \models \varphi$; otherwise player 1 wins. LTL games are determined, that is, there is always a winning strategy either for player 0 or for player 1. Checking whether player 0 has a winning strategy in an LTL game is a 2EXPTIME-complete problem [5].

We construct an LTL RMG with three players $m_0$, $m_1$, and *out* and show that player 0 has a winning strategy in the LTL game $(G_{LTL}, \varphi, v_0)$ if, and only if, there exists a strategy $\sigma_0$ for $m_0$ such that for all strategy profiles $\vec{\sigma}$, $K(\vec{\sigma}_{-0}, \sigma_0) \models \varphi$. The construction given below is similar to the one used in Lemma 2 to translate Kripke transition systems to SRML modules. Let us, first, define the following sets of controlled variables:

- $S_0^0 = \{x_v : v \in V_0\}$
- $S_0^1 = \{x_v : v \in V_1\}$
- $S_1^0 = \{y_v : v \in V_0\}$
- $S_1^1 = \{y_v : v \in V_1\}$

Using the above sets of controlled variables, we can define the following modules.

**module** $m_0$ **controls** $S_0^0 \cup S_0^1$
**init**
// to indicate that the initial state is $v_0$
$:: \top \rightsquigarrow x'_{v_0} := \top$
**update**
// for each $v \in V_0$ and $w \in \gamma(v) \setminus \{v\}$:
$:: x_v \vee y_v \rightsquigarrow x'_v := \bot; x'_w := \top$
// for each $v \in V_0$ with $v \in \gamma(v)$, i.e., for "loops":
$:: x_v \vee y_v \rightsquigarrow x'_v := \top$
// for each $v \in V_1$, where player 1 moves:
$:: x_v \rightsquigarrow x'_v := \bot$

**module** $m_1$ **controls** $S_1^0 \cup S_1^1$
**init**
// to indicate that the initial state is $v_0$
$:: \top \rightsquigarrow y'_{v_0} := \top$
**update**
// for each $v \in V_1$ and $w \in \gamma(v) \setminus \{v\}$:
$:: x_v \vee y_v \rightsquigarrow y'_v := \bot; y'_w := \top$
// for each $v \in V_1$ with $v \in \gamma(v)$, i.e., for "loops":
$:: x_v \vee y_v \rightsquigarrow y'_v := \top$
// for each $v \in V_0$, where player 0 moves:
$:: y_v \rightsquigarrow y'_v := \bot$

**module** *out* **controls** $\Sigma$
**init**
// initialize all propositions in $\Sigma$ to false, i.e.,
$:: \top \rightsquigarrow$ **skip**
**update**
// for each $v \in V_0 \cup V_1$ and
// where $\mu(v) = \{p_1, \ldots, p_k\}$ and $\Sigma \setminus \mu(v) = \{q_1, \ldots, q_m\}$:
$:: x_v \vee y_v \rightsquigarrow p'_1 := \top; \ldots; p'_k := \top; q'_1 := \bot; \ldots; q'_m := \bot$

Now, see that the runs with respect to *out* are completely defined by the possible runs of the LTL game. More precisely, we have

$$runs(out) = \{\emptyset; \boldsymbol{\rho} \ : \ \boldsymbol{\rho} \in runs(G_{LTL})\}\,.$$

As a consequence, for every formula $\psi$ and run $\boldsymbol{\rho}$ in the LTL game $G_{LTL}$, there is a strategy profile $\vec{\sigma}$ such that the following holds:

$\boldsymbol{\rho} \models \psi$ if and only if $K_{\vec{\sigma}} \models \mathbf{X}\psi$ .

Based on the above facts, we can now show that player 0 has a winning strategy $\sigma_c$ in the LTL game $(G_{LTL}, \varphi, v_0)$ if, and only if, there exist a winning strategy $\sigma_0$ for $m_0$ in the associated SRML game, as defined above.

($\Longrightarrow$) First, suppose that player 0 has a winning strategy $\sigma_c$ in the given LTL game $(G_{LTL}, \varphi, v_0)$, which can be assumed to be of finite memory. Then, we can use a construction similar to the one given for Lemma 6 to define an SRML module, say $m(\sigma_c)$, that characterises $\sigma_c$. Because of the definition of the SRML game induced by an LTL game, it immediately follows that module $m_0$ has a winning strategy, specified by $m(\sigma_c)$, for $\mathbf{X}\varphi$.

($\Longleftarrow$) Now, suppose that the SRML module $m_0$ has a winning strategy $\sigma_0$ in the game $(A, \gamma_0 = \mathbf{X}\varphi, \gamma_1 = \neg\mathbf{X}\varphi, \gamma_{out} = \top)$, with $A$ defined by SRML modules as those given above. Then, because $\sigma_0$ is a finite-memory strategy, it straightforwardly defines a strategy in the LTL game for $\varphi$.

This part of the proof shows that the problem is 2EXPTIME-hard for 3-player games. However, note that *out* is a deterministic module with the trivial goal $\top$. As a consequence, it can be eliminated from the game. Let $G_2$ be the game built based on $G$ such that the controlled variables and guarded commands of *out* are given to player $i$. It then follows that player $i$ has a strategy $\sigma_i$ such that for all $\vec{\sigma}_{-i}$ it holds that $\boldsymbol{\rho}(\vec{\sigma}_{-i}, \sigma_i) \models \varphi$ in $G$ if and only if $i$ has a strategy $\sigma_i^*$ such that for all $\vec{\sigma}^*_{-i}$ it holds that $\boldsymbol{\rho}(\vec{\sigma}^*_{-i}, \sigma_i^*) \models \varphi$ in $G_2$. The only if direction is trivial: simply let $\sigma_i^*$ be the strategy that plays as $\sigma_i$ and $\sigma_{out}$, any strategy for *out* in $G$. The if direction is almost just as easy. Consider the contrapositive argument, that is, that if $i$ does not have a strategy to realise $\varphi$ in $G_2$, then it does not have a strategy to realise $\varphi$ in $G$. Now, we only have to simply observe that the set of strategies for $i$ in $G_2$ is a super set of the set of strategies for $i$ in $G$. Therefore if $i$ does not have a strategy to realise $\varphi$ in $G_2$, then $o$ does not have such a strategy in $G$ either, which completes the proof of the 2EXPTIME-hardness result in 2-player games.

Finally consider the case of 1-player games. For membership of PSPACE, consider an arena $A$ and a formula $\varphi$. We simply need to check whether the LTL formula $\textsc{Th}(A) \wedge \varphi$ is satisfiable, which is clearly in PSPACE. For hardness, we can immediately reduce the LTL satisfiability problem for formulae over one variable; the details are straightforward from the description given above. □

It is natural to ask whether there are classes of games for which the decision problems we study are easier. While the study of such classes is not the main focus of this paper, there are natural subsets of LTL for which some of the decision problems we discuss are easier. Consider the class of as *Objective LTL* (OLTL) formulae. Informally, an OLTL formula is one in which no temporal connective occurs within the scope of a classical connective. Thus $\mathbf{FG}p$ and $p\,\mathbf{U}\,\mathbf{G}q$ are OLTL formulae, but $p \wedge \mathbf{X}q$ is not an OLTL formula; the latter is not an OLTL formula because the "**X**" operator occurs within the scope of the "$\wedge$" operator. Formally, the syntax of OLTL formulae is given as follows:

$$\begin{array}{lcl} \varphi & ::= & \mathbf{X}\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \psi \\ \psi & ::= & x \mid \neg\psi \mid \psi \vee \psi \end{array}$$

Now, we can show:

**Proposition 9.** REALIZABILITY *for OLTL RMGs is EXPTIME-complete.*

**Proof.** We first give an overview. Given an arena $A$, player $i$, and target formula $\varphi$, we start by constructing $K_A$, the Kripke structure induced by $A$. As we already noted, $K_A$ can be computed in time exponential in the size of $A$. We then use dynamic programming to label each state $s$ of $K_A$ with the subformulae of $\varphi$ that $i$ can guarantee to be able to bring about in $s$. The actual algorithm for this is closely related to the standard explicit state CTL model checking algorithm (see, e.g., [20]). The algorithm cannot be used for this purpose for arbitrary LTL formulae, because of the possibility of temporal operators occurring within the scope of a classical connective. However, the approach works for OLTL formulae. The algorithm operates in time polynomial in the size of $K_A$. Finally, we check whether all initial states of $K_A$ are labelled with the target formula $\varphi$; if the answer is "yes", then the answer to the REALIZABILITY problem is "yes", otherwise it is "no". The detailed algorithm is presented in Fig. 13 in the Appendix. For EXPTIME-hardness, we can straightforwardly reduce the problem of determining whether a player has a winning strategy in the game PEEK-$G_4$ (see the proof of Proposition 15, below, for details of the problem). □

Next, we ask whether a strategy profile is a Nash equilibrium:

NE-MEMBERSHIP
*Given:* RMG $G$ and strategy profile $\vec{\sigma}$.
*Question:* Is it the case that $\vec{\sigma} \in NE(G)$?

Note that the strategies $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ given in the input to this problem are represented as finite state machines. We assume that they are represented by explicitly enumerating the various components of the finite state machine.

**Proposition 10.** NE-MEMBERSHIP *for LTL RMGs is PSPACE-complete. In addition, it is PSPACE-hard even for one-player games and LTL goals over a singleton variable set* $\Phi$.

**Proof.** For membership in PSPACE we define a procedure that uses two oracles, one for LTL model-checking and one for satisfiability, which are both known to be in PSPACE [63]. The procedure is based on the observation that if the profile $\vec{\sigma}$ is a Nash equilibrium, then either $\boldsymbol{\rho}(\vec{\sigma}) \models \gamma_i$ or $\boldsymbol{\rho}(\vec{\sigma}_{-i}, \sigma_i') \not\models \gamma_i$, for every player $i$ and strategy $\sigma_i'$. For hardness, we reduce LTL satisfiability for formulae with one propositional variable [19] to a one-player LTL RMG. Let us first prove membership in PSPACE.

Recall that a profile $\vec{\sigma}$ of deterministic strategies in an LTL RMG is a pure-strategy Nash equilibrium if and only if for every player $i$ in $N$,

$$\text{either } \boldsymbol{\rho}(\vec{\sigma}) \models \gamma_i \text{ or } \boldsymbol{\rho}(\vec{\sigma}_{-i}, \sigma_i') \not\models \gamma_i \text{ for every strategy } \sigma_i' \in \Sigma_i.$$

Thus, we can, for every player $i$, check whether

$$\text{both } \boldsymbol{\rho}(\vec{\sigma}) \not\models \gamma_i \text{ and } \boldsymbol{\rho}(\vec{\sigma}_{-i}, \sigma_i') \models \gamma_i \text{ for some strategy } \sigma_i' \in \Sigma_i,$$

and conclude $\vec{\sigma} \notin NE(G)$ if such a statement holds; otherwise $\vec{\sigma} \in NE(G)$.

Checking $\boldsymbol{\rho}(\vec{\sigma}) \not\models \gamma_i$ can be done in PSPACE. First, use Lemma 6 and Corollary 7 to construct an LTL formula $\textsc{Th}(A_{\vec{\sigma}})$ that characterises $\vec{\sigma}$. It is not difficult to show that, for all runs $\boldsymbol{\rho}$ that satisfy $\textsc{Th}(A_{\vec{\sigma}})$, it is the case that $\boldsymbol{\rho}|_\Phi = \boldsymbol{\rho}(\vec{\sigma})$. To simplify notation, hereafter, we will write $\textsc{Th}(\vec{\sigma})$ for $\textsc{Th}(A_{\vec{\sigma}})$. Such a formula, namely $\textsc{Th}(\vec{\sigma})$, is polynomial in $|\vec{\sigma}|$.

Following the automata-theoretic approach to model checking, we now construct two alternating Büchi word (ABW) automata $\mathcal{A}_{\textsc{Th}(\vec{\sigma})}$ and $\mathcal{A}_{\gamma_i}$ which accept exactly all the words that satisfy the formulae $\textsc{Th}(\vec{\sigma})$ and $\gamma_i$, respectively. Such alternating automata are polynomial in $|\textsc{Th}(\vec{\sigma})|$ and $|\gamma_i|$. We now simply ask whether $L(\mathcal{A}_{\textsc{Th}(\vec{\sigma})})|_\Phi \subseteq L(\mathcal{A}_{\gamma_i})$, that is, whether the language accepted by $\mathcal{A}_{\textsc{Th}(\vec{\sigma})}$ and restricted to $\Phi$ is included in the language accepted by $\mathcal{A}_{\gamma_i}$. Such a question can be answered in PSPACE. If $L(\mathcal{A}_{\textsc{Th}(\vec{\sigma})})|_\Phi \subseteq L(\mathcal{A}_{\gamma_i})$ then $\boldsymbol{\rho}(\vec{\sigma}) \models \gamma_i$; otherwise $\boldsymbol{\rho}(\vec{\sigma}) \not\models \gamma_i$. Since PSPACE is a deterministic complexity class, checking $\boldsymbol{\rho}(\vec{\sigma}) \not\models \gamma_i$ is also in PSPACE, as required.

On the other hand, checking that $\boldsymbol{\rho}(\vec{\sigma}_{-i}, \sigma_i') \models \gamma_i$ for some $\sigma_i' \in \Sigma_i$ can be done using constructions very similar to those used in Lemma 6 and Corollary 7, but in this case we check LTL satisfiability instead. First, let us write $\textsc{Th}(m(\sigma_j))$ for the LTL formula that characterises $\sigma_j$, with $j \in N$. Specifically, we check whether the LTL formula $\gamma_i \wedge \bigwedge_{j \in N \setminus \{i\}} \textsc{Th}(m(\sigma_j))$ is satisfiable, which also can be done in PSPACE.

Note that since in order to solve the NE-MEMBERSHIP problem we checked whether the LTL formula $\gamma_i \wedge \bigwedge_{j \in N \setminus \{i\}} \textsc{Th}(m(\sigma_j))$ is satisfiable, and the size of such a formula is polynomial in the sizes of both the LTL goals $\gamma_i$ and the strategies $\sigma_j$, then it follows that NE-MEMBERSHIP is PSPACE in the sizes of both the game $G$ and the input strategy profile $\vec{\sigma}$.

Now, for hardness, we reduce the LTL satisfiability problem (which is PSPACE-complete, even for formulae with only one propositional variable [19]) in the same way that is done for iBG [29]. More specifically, we take the class of LTL formulae over one propositional variable and ask whether a given LTL formula $\varphi$ in this class is satisfiable. Because this formula is over only one propositional variable, say $p$, any model can be produced by an SRML module.

Then, reasoning as in the proof for iBG [29], we can show that the SRML module $m$, defined below, has a winning strategy to satisfy $\varphi \wedge q$ if, and only if, the formula $\varphi$ is satisfiable. This strategy can be used to build a strategy profile $\sigma_m$ that is not a Nash equilibrium if, and only if, $\varphi$ is satisfiable.

**module** $m$ **controls** $p, q$
**init**
$:: \top \rightsquigarrow p' := \top$
$:: \top \rightsquigarrow p' := \bot$
$:: \top \rightsquigarrow q' := \top$
$:: \top \rightsquigarrow q' := \bot$
**update**
$:: \top \rightsquigarrow p' := \top$
$:: \top \rightsquigarrow p' := \bot$

Now, consider a strategy $\sigma_m$ such that $q \notin \tau(q^0)$. Then, it follows that $\sigma_m \not\models \varphi \wedge q$. If $\varphi$ is satisfiable then $m$ has a beneficial deviation: switch to a strategy $\sigma_m'$ that builds a model for $\varphi$ and such that $q \in \tau(q^0)$. However, if $\varphi$ is not satisfiable, then no strategy for $m$ will satisfy $\varphi \wedge q$, and hence $\sigma_m$ would be a Nash equilibrium in such a case. Note that the reason this reduction works is because the arena where the game is played has constant size (because $\varphi$ is a formula with only one propositional variable). □

The next problem asks if a game has any Nash equilibrium.

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

**Proposition 11.** NON-EMPTINESS *for LTL RMGs is 2EXPTIME-complete, and it is 2EXPTIME-hard for 2-player games.*

**Proof.** For membership in 2EXPTIME, we instantiate the E-NASH problem (described below) with $\varphi = \top$. For hardness, one can reduce LTL synthesis with two variables, say $x$ and $y$ to a two-player game with five Boolean variables. The reduction takes as input an LTL synthesis game with two players, 1 and 2, and an LTL formula $\varphi$ such that player 1 has a winning strategy in the synthesis game if and only if $\varphi$ can be synthesised; otherwise, player 2 has a winning strategy to show that $\neg\varphi$ can be synthesised – because it is a zero-sum game. Based on this input game, the reduction uses the two modules below and following LTL goals:

- $\gamma_1 = (\neg\varphi \rightarrow (p \leftrightarrow q)) \wedge (\varphi \rightarrow r)$
- $\gamma_2 = (\neg\varphi \rightarrow \neg(p \leftrightarrow q)) \wedge (\varphi \rightarrow (\varphi \leftrightarrow \neg r))$

where $\varphi$ is an LTL formula over Boolean variables $x$ and $y$.

**module** $m_1$ **controls** $\{x, p, r\}$
**init**
$:: \top \rightsquigarrow x := \bot;\ p := \bot;\ r := \bot$
$:: \top \rightsquigarrow x := \bot;\ p := \bot;\ r := \top$
$:: \top \rightsquigarrow x := \bot;\ p := \top;\ r := \bot$
$:: \top \rightsquigarrow x := \bot;\ p := \top;\ r := \top$
$:: \top \rightsquigarrow x := \top;\ p := \bot;\ r := \bot$
$:: \ldots$
**update**
$:: \top \rightsquigarrow x := \bot;\ p := \bot;\ r := \bot$
$:: \top \rightsquigarrow x := \bot;\ p := \bot;\ r := \top$
$:: \top \rightsquigarrow x := \bot;\ p := \top;\ r := \bot$
$:: \top \rightsquigarrow x := \bot;\ p := \top;\ r := \top$
$:: \top \rightsquigarrow x := \bot;\ p := \bot;\ r := \bot$
$:: \ldots$

**module** $m_2$ **controls** $\{y, q\}$
**init**
$:: \top \rightsquigarrow y := \bot;\ q := \bot$
$:: \top \rightsquigarrow y := \bot;\ q := \top$
$:: \ldots$
**update**
$:: \top \rightsquigarrow y := \bot;\ q := \bot$
$:: \top \rightsquigarrow y := \bot;\ q := \top$
$:: \ldots$

Given this two-player game, $G$, it is not hard to check that player 1 has a winning strategy to synthesise $\varphi$ if and only if $G$ has a Nash equilibrium, from which 2EXPTIME-hardness follows.

- ($\Rightarrow$) Assume that player 1 has a winning strategy to synthesise $\varphi$. Then, module $m_1$ can use such a winning strategy to ensure that $\varphi$ holds, while setting $r$ to $\top$. Then, $\gamma_1$ is satisfied and $m_1$ will not deviate. On the other hand, in this case, $m_2$ will not have its goal $\gamma_2$ satisfied, but cannot beneficially deviate since $(\varphi \rightarrow (\varphi \leftrightarrow \neg r))$ will still be false in every unilateral deviation of $m_2$.
- ($\Leftarrow$) We prove the contrapositive statement: assume that player 1 does not have a winning strategy to synthesise $\varphi$ and show that in such a case $G$ does not have a Nash equilibrium. Since the game for LTL synthesis is determined and player 1 does not have a winning strategy to synthesise $\varphi$ then we know that player 2 has a winning strategy for $\neg\varphi$. We will analyse all possible cases and show that in each instance at least one of the players has a beneficial deviation. Let $\vec{\sigma} = (\sigma_1, \sigma_2)$ be an arbitrary strategy profile and assume first that $\vec{\sigma} \models \neg\varphi$. Then, if $\vec{\sigma} \models (p \leftrightarrow q)$ player $m_2$ will have a beneficial deviation. If, on the other hand, $\vec{\sigma} \models \neg(p \leftrightarrow q)$ then player $m_1$ will have a beneficial deviation. Now, suppose that $\vec{\sigma} \models \varphi$. Then, if $\vec{\sigma} \models \neg r$ player $m_1$ will have a beneficial deviation. If, on the other hand, $\vec{\sigma} \models r$ then player $m_2$ will have a beneficial deviation: in case $\vec{\sigma} \models \neg(p \leftrightarrow q)$, player $m_2$ simply has to deviate to a strategy $\sigma_2'$ that is winning for $\neg\varphi$ and keep the same value for $q$; in case $\vec{\sigma} \models (p \leftrightarrow q)$, player $m_2$ can deviate to a strategy $\sigma_2'$ that is winning for $\neg\varphi$ and change the value of $q$ with respect to the one given by $\sigma_2$. This analysis covers all possible cases with respect to $\varphi$, $r$, and $p, q$.

Then, the problem is 2EXPTIME-hard with two players and five Boolean variables. □

Arguably, the two key decision problems relating to the game-theoretic analysis of game-like concurrent and multi-agent systems are *(i)* checking whether an LTL formula holds on *some* run given by a Nash equilibrium of the system; and *(ii)* checking whether a given LTL formula holds on *all* runs given by the Nash equilibria of the system. These two problems are formalised in the E-Nash and A-Nash problems, respectively.

E-Nash
*Given:* RMG $G$, LTL formula $\varphi$.
*Question:* Does $\boldsymbol{\rho}(\vec{\sigma}) \models \varphi$ hold for some $\vec{\sigma} \in NE(G)$?

A-Nash
*Given:* RMG $G$, LTL formula $\varphi$.
*Question:* Does $\boldsymbol{\rho}(\vec{\sigma}) \models \varphi$ hold for all $\vec{\sigma} \in NE(G)$?

These two decision problems were first studied in [29] for iterated Boolean games (iBG). In such a setting, there was no language to specify the systems that the iBGs were to model. Even more, the setting was quite simple: it was assumed that in every system state all variables could be updated. In this article, we study systems that do not impose this restriction, and we show that we obtain similar complexity results in this case. Note that using RMGs allows us to express systems that cannot be directly expressed in iBGs; however, as we noted above, RMGs cannot *succinctly* express iBGs.

More specifically, even though RMGs can be used to impose constraints in the choices that players can make (i.e., the values they can give the variables they control), it is still possible to find a reduction from the E-Nash and A-Nash for LTL RMGs to the same problems in the context of iBGs. Before describing the reduction, let us define an iBG $G'$, which we will call a 'supervised game', in which all the Nash equilibria of the iBG $G'$ must be consistent with the Nash equilibria of some initially given LTL RMG $G$.

Formally, an iterated Boolean game is a structure

$$(N, \Phi, (\Phi_i)_{i \in N}, (\gamma_i)_{i \in N})$$

where every element of the structure is as for LTL RMGs. Strategies are defined as for LTL RMGs, that is, as Moore finite-state machines. However, as we can see, the arena where an iBG is played is implicitly given: it is the unique clique whose vertices are the valuations $V = 2^{\Phi}$. Having such an arena to play the game informally means that, in an iBG, it is assumed that in every state all agents can update the values of all the variables they control; note that, in contrast, in an LTL RMG only the variables that are in the scope of *enabled* guarded commands can be updated.

More specifically, note that, in general, the LTL RMG representation of an iBG may result in a game of *exponential* size since one would have to specify in the LTL RMG all the (exponentially many) choices with respect to $\Phi$ that players in an iBG have at their disposal. This is, indeed, the reason why a simple polynomial-time reduction from iBG to LTL RMGs is not possible – e.g., in order to obtain hardness results for LTL RMGs.

Nevertheless, as mentioned above, for the particular case of the E-Nash and A-Nash problems, a reduction from LTL RMGs to iBGs is possible. We first present a key technical result, which relies on the following game construction. Given an LTL RMG $G = (A = (N, \Phi, (m_i)_{i \in N}), (\gamma_i)_{i \in N})$, we let the iBG $G' = (N', \Phi', (\Phi_i)_{i \in N'}, (\gamma'_i)_{i \in N'})$ be its *supervised game*, given by

- $N' = N \cup \{n+1, n+2\}$ and $\Phi' = \Phi \cup \{x_{n+1}, x_{n+2}\}$,
- $\gamma'_i = \bigwedge_{j \in N} \textsc{Th}(m_j) \wedge \gamma_i$,
- $\gamma'_{n+1} = \bigwedge_{j \in N} \textsc{Th}(m_j) \vee (x_{n+1} \leftrightarrow x_{n+2})$,
- $\gamma'_{n+2} = \bigwedge_{j \in N} \textsc{Th}(m_j) \vee \neg(x_{n+1} \leftrightarrow x_{n+2})$,

where $i \in N$. Recall that each $\textsc{Th}(m_i)$ is an LTL formula that characterises module $m_i$. Based on this definition of supervised games, we obtain the following result.

**Lemma 12.** *Let $G = (A = (N, \Phi, (m_i)_{i \in N}), (\gamma_i)_{i \in N})$ be an LTL RMG and let $G' = (N', \Phi', (\Phi_i)_{i \in N'}, (\gamma'_i)_{i \in N'})$ be its supervised game, then:*

$$\vec{\sigma} \in NE(G) \quad \text{if and only if} \quad (\vec{\sigma}, \sigma_{n+1}, \sigma_{n+2}) \in NE(G')$$

*for all $\vec{\sigma}$ and strategies $\sigma_{n+1}$, $\sigma_{n+2}$ for players $n+1$ and $n+2$.*

**Proof.** We first prove the left-to-right direction. Since $\vec{\sigma}$ is a profile of strategies that is consistent with $A$, then the LTL formula

$$\bigwedge_{j \in N} \textsc{Th}(m_j)$$

is satisfied and, hence, also the LTL goals $\gamma_{n+1}$ and $\gamma_{n+2}$. Thus, players $n+1$ and $n+2$ will not deviate. Moreover, because players $n+1$ and $n+2$ do not control any variable already present in the game $G$ and the goals $(\gamma'_i)_{i \in N}$ do not depend on the Boolean variables in $\{x_{n+1}, x_{n+2}\}$, it follows that

$$\boldsymbol{\rho}(\vec{\sigma}) \models \gamma_i \quad \text{iff} \quad \boldsymbol{\rho}(\vec{\sigma}, \sigma_{n+1}, \sigma_{n+2})|_\Phi \models \gamma_i' \quad \text{iff} \quad \boldsymbol{\rho}(\vec{\sigma}, \sigma_{n+1}, \sigma_{n+2}) \models \gamma_i'$$

for all goals in $(\gamma_i')_{i \in N}$ and strategies $\sigma_{n+1}, \sigma_{n+2}$ for players $n+1$ and $n+2$. Then, players $i$ in $N$ who have their goal achieved in $G$ by $\boldsymbol{\rho}(\vec{\sigma})$ will also have it satisfied in $G'$ by $\boldsymbol{\rho}(\vec{\sigma}, \sigma_{n+1}, \sigma_{n+2})$ – thus, they will not want to deviate. Similarly, players $i$ in $N$ who did not have their goal achieved in $G$ by $\boldsymbol{\rho}(\vec{\sigma})$ will not have it satisfied in $G'$ by $\boldsymbol{\rho}(\vec{\sigma}, \sigma_{n+1}, \sigma_{n+2})$ either, and hence will not have a beneficial deviation in $G'$, provided that they only use strategies that are consistent with $A$. Then, if a player $i$ has a beneficial deviation in $G'$ and not in $G$ it must be by switching to a strategy $\sigma_i'$ that is not consistent with $A$. But, in such a case its goal $\gamma_i$ will not be satisfied because the conjunct $\bigwedge_{j \in N} \textsc{Th}(m_j)$ will be false. Then, in $G'$ no player can have a beneficial deviation by switching to a strategy that is not consistent with $A$. As a consequence, $(\vec{\sigma}, \sigma_{n+1}, \sigma_{n+2}) \in NE(G')$, as desired.

We now prove the right-to-left direction. Since $(\vec{\sigma}, \sigma_{n+1}, \sigma_{n+2}) \in NE(G')$, we know both that players $n+1$ and $n+2$ have their goals achieved and that the strategies in $\vec{\sigma}$ are consistent with the SRML modules $(m_i)_{i \in N}$ – because the LTL formula $\bigwedge_{j \in N} \textsc{Th}(m_j)$ is necessarily satisfied in all Nash equilibria of $G'$ (because the two final players are playing a version of Matching Pennies, for which there is no pure Nash equilibrium).

Hence, $\vec{\sigma}$ is a legal strategy in $G$ (i.e., consistent with the arena given by $(m_i)_{i \in N}$). For exactly the same reasons given in the left-to-right direction, the strategy profile $\vec{\sigma}$ is, therefore, a Nash equilibrium in $G$, that is, because

$$\boldsymbol{\rho}(\vec{\sigma}, \sigma_{n+1}, \sigma_{n+2}) \models \gamma_i' \quad \text{iff} \quad \boldsymbol{\rho}(\vec{\sigma}, \sigma_{n+1}, \sigma_{n+2})|_\Phi \models \gamma_i' \quad \text{iff} \quad \boldsymbol{\rho}(\vec{\sigma}) \models \gamma_i$$

as needed. Note, in particular, that if a player $i$ did not have its goal $\gamma_i'$ satisfied in $G'$ by $\boldsymbol{\rho}(\vec{\sigma}, \sigma_{n+1}, \sigma_{n+2})$, then such a strategy profile does not satisfy $\gamma_i$ either – because the LTL formula $\bigwedge_{j \in N} \textsc{Th}(m_j)$ is necessarily satisfied. Then, $\boldsymbol{\rho}(\vec{\sigma}) \not\models \gamma_i$. Moreover, player $i$ cannot have a beneficial deviation in $G$ by switching to an alternative strategy because every strategy in $G$ was already available in $G'$. □

The reason why we call the iBG $G'$ "the supervised game of $G$," under the definition given above, is that in $G'$ we have introduced two new players, namely $n+1$ and $n+2$, who act as "supervisors" of the behaviour of the players $i$ in $N$ in the following sense: they (players $n+1$ and $n+2$) can easily ensure that any strategy profile in the iBG $G'$ which forms a Nash equilibrium is possible if, and only if, the other players comply with the constrained behaviour imposed by the specification of the modules in $G$, that is, the behaviour formally specified by the LTL formula $\bigwedge_{j \in N} \textsc{Th}(m_j)$. Then, using Lemma 12, we can show the next result.

**Proposition 13.** *The* E-NASH *and* A-NASH *problems for LTL RMGs are both 2EXPTIME-complete.*

**Proof.** Consider E-NASH first. For hardness, we can trivially reduce the NON-EMPTINESS problem for LTL RMGs by asking whether $\top$ is satisfied on some Nash equilibrium outcome. For membership, use Lemma 12 to construct a supervised game and straightforwardly reduce E-NASH for LTL RMGs to E-NASH for iBG, which is known to be 2EXPTIME-complete [29]. Since 2EXPTIME is a deterministic complexity class, the desired complexity result for the A-NASH problem immediately follows.

Finally, since in [29] we use rational synthesis [23] to solve the E-NASH problem for iBG, which is 2EXPTIME in the sizes of players' goals and input LTL formula, then we obtain that E-NASH for LTL RMGs is 2EXPTIME with respect to both the size of the RMG $G$ and input LTL formula $\varphi$. □

This result shows, in particular, that in the linear-time setting analysing *equilibrium properties* (via the E-NASH and A-NASH problems) is as hard as the controller synthesis problem for LTL [57] – or, expressed in game-theoretic terms, as hard as solving two-player zero-sum perfect-information games with LTL goals.

### 6.2. On the complexity of CTL RMGs

In this section, we are interested in the computational complexity of the decision problems just investigated, but now in the branching-time framework we developed in previous sections. The first complexity result we are to present is for REALIZABILITY, which we obtain via a reduction to CTL supervisory-control games in [42] (aka CTL games on graphs).

**Proposition 14.** REALIZABILITY *for CTL RMGs is 2EXPTIME-complete.*

**Proof.** For membership in 2EXPTIME, similar to the case for LTL, we use a reduction to the control-synthesis problem for CTL specifications with reactive environments [42]. Firstly, from the set of modules in the given game $G$, construct the induced Kripke structure $K$, which is at most exponential in the size of the modules. Now, associate player $i$ with the "system" in the CTL control-synthesis problem, the set of players $N \setminus \{i\}$ with its (reactive) "environment" and let $\varphi$ be the CTL goal of the system – hence, $\neg\varphi$ would be the CTL goal of the environment since it is a zero-sum game. It is not hard to show that player $i$ has a strategy to achieve $\varphi$ if, and only if, the system has a strategy to win the CTL control-synthesis

game.[4] Because the complexity is doubly exponential in the size of the formula $\varphi$, but only exponential in the size of the given graph ($K$ in this case), then the procedure is now doubly exponential in the size of the initial game $G$. Therefore, containment in 2EXPTIME still follows, but with an exponentially higher system complexity, matching the result in the linear-time case using a logic-based approach instead, that is, a decision procedure that is doubly exponential in the sizes of both the system/game and the formula.

For hardness, again analogous to the LTL case, we use the CTL supervisory-control games in [5], where, in particular, non-deterministic strategies instead of only deterministic ones are considered. □

This result sharply contrasts with the computational complexity of the usual realizability problem for CTL specifications, which is known to be EXPTIME-complete [21]. There is, however, a clear difference between the standard realizability problem and the one we are considering: whereas in the former a strategy (for player $i$, the system) must be constructed with respect to a known special strategy of the environment (for players in $N \setminus \{i\}$), in the latter we have to construct a strategy that ensures the goal of player $i$ with respect to *any* strategy of player $i$'s opponents.

Proposition 14 shows, in addition, that whereas in the linear-time framework REALIZABILITY for a multi-player game is as hard as synthesis in a two-player system, in the branching-time context the problem is considerably harder, that is, while synthesis for CTL specifications is EXPTIME-complete, REALIZABILITY for CTL RMGs is 2EXPTIME-complete.

On the other hand, the following complexity result shows that, as in the linear-time case, the NE-MEMBERSHIP problem is as hard as the satisfiability problem for the temporal logic under consideration.

**Proposition 15.** NE-MEMBERSHIP *for CTL RMGs is EXPTIME-complete.*

**Proof.** To show membership in EXPTIME, we use a variation of the algorithm given to show Proposition 6 in [29]. In particular, we use the following two facts:

- CTL model checking over Kripke structures can be done in polynomial time (and the Kripke structure $K_A$ induced by an arena $A$ may be exponential in the size of $A$);
- the intersection non-emptiness problem for Alternating Büchi Tree (ABT) automata is EXPTIME-complete.

Then, firstly, in order to check whether $K_{\vec{\sigma}}|_\Phi \not\models \varphi$, where $\varphi$ is a CTL formula, we construct the Kripke structure $K_{\vec{\sigma}}$ induced by $A_{\vec{\sigma}}$. Since the Kripke structure $K_{\vec{\sigma}}$ may be exponential in the size of $\vec{\sigma}$, this step (which is used to check if a player gets or not its goal achieved by $\vec{\sigma}$) can be done in EXPTIME. The other step, namely, whether a player $i$ that did not get its goal $\gamma_i$ achieved can have a beneficial deviation by switching to an alternative strategy, can be reduced to the intersection non-emptiness problem for ABT automata – here is where we deviate from the algorithm given to show Proposition 6 in [29]. Instead of checking satisfiability of a CTL formula, we will construct an ABT $\mathcal{A}_{\gamma_i}$ that represents $\gamma_i$. Such an automaton is polynomial in the size of $\gamma_i$; see [41,70]. Also, because our strategies are (non-deterministic) transducers, they are also, in particular, Non-deterministic Büchi Tree (NBT) automata with a trivial acceptance condition (all states are accepting states). Write $\mathcal{A}_{\sigma_i}$ for each strategy regarded as an NBT of this kind. Then, checking that player $i$ has a strategy to deviate and achieve its goal can be verified by checking non-emptiness of the following automaton:

$$\mathcal{A}_{\sigma_1} \times \ldots \times \mathcal{A}_{\sigma_{i-1}} \times \mathcal{A}_{\gamma_i} \times \mathcal{A}_{\sigma_{i+1}} \times \ldots \times \mathcal{A}_{\sigma_n},$$

since we need to check that the language of trees accepted by this automaton is non-empty. Because every automaton $\mathcal{A}_x$ is linear or polynomial in the size of each $x$, this step is exponential in the size of $x$, that is, exponential in the sizes of both the strategies $\sigma_i$ and the goal $\gamma_i$. Then, inclusion in EXPTIME follows.

Hardness, on the other hand, is shown by a reduction from the problem of determining whether a given player has a winning strategy in the game PEEK-$G_4$ [64, p. 158]; our construction of a CTL RMG for an instance of PEEK-$G_4$ is inspired by the construction presented in [67].

An instance of PEEK-$G_4$ is given by a structure: $(X_1, X_2, X_3, \varphi)$, where:

- $X_1$ and $X_2$ are disjoint, finite sets of Boolean variables, with the intended interpretation that the variables in $X_1$ are under the control of agent 1, and $X_2$ are under the control of agent 2;
- $X_3 \subseteq (X_1 \cup X_2)$ are the variables deemed to be true in the initial state of the game; and
- $\varphi$ is a propositional logic formula over the variables $X_1 \cup X_2$, representing the winning condition.

The game is played in a series of rounds, with agents $i \in \{1, 2\}$ alternating turns – and agent 1 moving first – to select a value ($\top$ or $\bot$) for one of their variables in $X_i$. The game starts from the initial assignment of truth values defined by $X_3$. Variables that were not changed retain the same truth value in the subsequent round. An agent wins in a given round if it makes a move such that the resulting truth assignment defined by that round makes $\varphi$ true.

[4] The only difference is that in $G$ the two players play concurrently, whereas in the CTL control-synthesis game, they play sequentially. A similar situation is found in Proposition 3 of [32].

The decision problem associated with PEEK-$G_4$ is determining if agent 2 has a winning strategy in a given instance $(X_1, X_2, X_3, \varphi)$. We would like to remark that since PEEK-$G_4$ only requires "memoryless" strategies, they can be modelled with our finite-state machine model of strategies: whether an agent $i$ can win depends only on the current truth assignment, the distribution of variables, the winning formula, and whose turn it is currently. As a corollary, if agent $i$ can force a win, then it can force it in $O(2^{|X_1 \cup X_2|})$ moves.

From $(X_1, X_2, X_3, \varphi)$, the following CTL RMG can be constructed. For each Boolean variable $x \in (X_1 \cup X_2)$, we create a variable with the same name in our SRML model, and we create two additional Boolean variables: *turn* and $p$. The former, will have the intended interpretation that if $turn = \top$, then it is agent 1's turn to move, while if $turn = \bot$, then it is agent 2's turn to move. We then have an SRML module, which we call *move*, the purpose of which is to control *turn*, toggling its value in each successive round, starting from the initial case of it being agent 1's move. Note that because of the definition of the module *move*, it has only one possible legal strategy up to equality with respect to its observable behaviour, namely $\top, \bot, \top, \bot, \ldots$. The goal $\gamma_{move}$ of this player in the CTL RMG is defined to be $\gamma_{move} = \top$.

**module** *move* **controls** *turn*
**init**
$:: \top \rightsquigarrow turn' := \top$
**update**
$:: turn \rightsquigarrow turn' := \bot$
$:: (\neg turn) \rightsquigarrow turn' := \top$

Moreover, for each of the two PEEK-$G_4$ players $i \in \{1, 2\}$, we create an SRML module $ag_i$ that **controls** the variables $X_i$. We also let the module $ag_2$ control $p$. The modules $ag_1$ and $ag_2$ are as follows. For $ag_1$, it begins by deterministically initialising the values of all its variables to the values defined by $X_3$ (that is, if variable $x \in X_1$ appears in $X_3$ then this variable is initialised to $\top$, otherwise it is initialised to $\bot$). The module $ag_2$ is the same as defined for $ag_1$, save that $ag_2$ has two init commands instead of only one: while one of the init commands sets $p$ to $\top$, the other one sets $p$ to $\bot$.

Then, after initialisation, when it is a player's turn, such a player can non-deterministically choose at most one of the variables under its control and toggle the value of this variable; when it is not this player's turn, it has no choice but to do nothing, leaving the value of all its variables unchanged. The general structure of $ag_1$ is thus as follows, where $X_1 = \{x_1, \ldots, x_k\}$.

**module** $ag_1$ **controls** $x_1, \ldots, x_k$
**init**
`// initialise based on values from` $X_3$
$:: \top \rightsquigarrow x_1' := \ldots; x_k := \ldots$
**update**
$:: turn \rightsquigarrow x_1' := \bot$
$:: turn \rightsquigarrow x_1' := \top$
$\ldots$
$:: turn \rightsquigarrow x_k' := \bot$
$:: turn \rightsquigarrow x_k' := \top$
$:: \top \rightsquigarrow$ **skip**

And the general structure of $ag_2$ is thus as follows, where $X_2 = \{y_1, \ldots, y_l\}$.

**module** $ag_2$ **controls** $p, y_1, \ldots, y_l$
**init**
`// initialise based on values from` $X_3$
$:: \top \rightsquigarrow p' := \top; y_1' := \ldots; y_l := \ldots$
$:: \top \rightsquigarrow p' := \bot; y_1' := \ldots; y_l := \ldots$
**update**
$:: turn \rightsquigarrow y_1' := \bot$
$:: turn \rightsquigarrow y_1' := \top$
$\ldots$
$:: turn \rightsquigarrow y_l' := \bot$
$:: turn \rightsquigarrow y_l' := \top$
$:: \top \rightsquigarrow$ **skip**

Notice that an agent can always **skip**, electing to leave its variables unchanged; and, if it is not this agent's turn to move, this is the *only* choice it has. The SRML arena under consideration contains just these three modules. We define the goal of agent 1 to be $\gamma_1 = \top$, and the goal of agent 2 to be the CTL formula $\gamma_2 = p \wedge \mathbf{A}(\neg\varphi\, \mathbf{U}\, (turn \wedge \varphi))$.

In addition, we also let the strategy $\sigma_1$ of agent 1 be the trivial *non-deterministic* strategy that contains all possible choices available at each round of the game, i.e., a strategy that in every round non-deterministically chooses any of the

commands in the update part of the module $ag_1$. For $ag_2$ we let the strategy $\sigma_2$ be any strategy such that it chooses the initialisation command where $p = \bot$. Then, clearly, with this strategy profile, namely $(\sigma_1, \sigma_2, \sigma_{move})$, players $ag_1$ and $move$ have their goals satisfied, whereas player $ag_2$ fails to have its goal $\gamma_2$ satisfied (because $\sigma_2$ makes $p = \bot$). More formally, with respect to this strategy profile we know that $K_{(\sigma_1,\sigma_2,\sigma_{move})} \not\models \gamma_2$.

However, if agent 2 has a winning strategy in PEEK-$G_4$, then the player $ag_2$ has a beneficial deviation $\sigma_2'$ to a strategy making the same choices as such a winning strategy and which sets $p = \top$ in the initialisation stage, so that its goal $\gamma_2$ is achieved. Hence, whereas $(\sigma_1, \sigma_2, \sigma_{move})$ is not a Nash equilibrium, the strategy profile $(\sigma_1, \sigma_2', \sigma_{move})$ would be an equilibrium as in that case all players get their goals satisfied. Formally, the following holds:

if agent 2 has a winning strategy, then $(\sigma_1, \sigma_2, \sigma_{move}) \notin NE(G)$.

If, on the other hand, agent 2 does not have a winning strategy in PEEK-$G_4$, then, because the strategy $\sigma_1$ of $ag_1$, as defined above, contains all possible choices that agent 1 can make, it necessarily follows that there would be a run of the game that can be used as a witness to show that $\gamma_2$ is not satisfied, in particular, for all strategies $\sigma_2'$ that set $p = \top$ (as otherwise agent 2 would have a winning strategy – contradicting our initial hypothesis). Then, it follows that in this case player $ag_2$ cannot have a beneficial deviation and that the strategy profile $(\sigma_1, \sigma_2, \sigma_{move})$ is in fact a Nash equilibrium. Formally, in this case, the following statement holds:

if agent 2 has no winning strategy, then $(\sigma_1, \sigma_2, \sigma_{move}) \in NE(G)$.

Thus, agent 2 has a winning strategy for the instance of PEEK-$G_4$ if, and only if, $(\sigma_1, \sigma_2, \sigma_{move})$ is not a Nash equilibrium. Since EXPTIME is a deterministic complexity class, hardness of the NE-MEMBERSHIP problem for CTL RMGs with respect to the EXPTIME complexity class follows. Finally, observe that because we use intersection non-emptiness to show membership in EXPTIME, we obtain that NE-MEMBERSHIP is EXPTIME in the sizes of both the RMG $G$ and the strategy profile $\vec{\sigma}$. □

Finally, we show that, as in the linear-time case, checking both whether a system/game has at least one Nash equilibrium as well as whether a CTL formula holds on *some* or on *all* Nash equilibria of such a game-like multi-agent system are 2EXPTIME-hard problems. Formally:

**Proposition 16.** *The following three problems are 2EXPTIME-hard:*
NON-EMPTINESS, E-NASH, *and* A-NASH *for CTL RMGs.*

**Proof.** For these results, we reduce CTL supervisory-control games on graphs to the NON-EMPTINESS problem – a game construction in which we add two new players who ensure that the "system" player in the initially given CTL supervisory-control game has a winning strategy if, and only if, there is a Nash equilibrium in the constructed CTL RMG. Then, the other two decision problems we have considered, namely, E-NASH and A-NASH, inherit such a hardness complexity result. □

## 7. Beyond reactive modules

The results we prove in this article may at first sight appear to be closely tied to a single formalism for defining game arenas, i.e., the REACTIVE MODULES language. However, this language is in fact very general. To illustrate the scope of our framework, we now show how a standard AI formalism for planning agents can be directly modelled within our framework: the propositional STRIPS notation [22,12,25]. Descendants of the STRIPS notation remain widely used within the planning community, and various multi-agent planning models based on STRIPS have been investigated within the literature, such as the MA-STRIPS model of Brafman and Domshlak [10].

The basic idea of the STRIPS notation is to model the effects of actions in terms of "pre-condition, delete, add lists": the pre-condition defines what must be true in order for the action to be executed; the add list defines what propositions will be made true by the execution of the action; and the delete list defines what propositions will be made false by the execution of the action. As we did previously, we assume a vocabulary $\Phi$ of propositional variables, used to characterise the state of the system. Formally, a *STRIPS descriptor* for an action $\alpha$ is given by a structure

$$\alpha = (\mathcal{T}_\alpha, \mathcal{F}_\alpha, \mathcal{D}_\alpha, \mathcal{A}_\alpha),$$

where:

- $\mathcal{T}_\alpha$ is the set of propositions that must be true in order for $\alpha$ to be executed;
- $\mathcal{F}_\alpha$ is the set of propositions that must be false in order for $\alpha$ to be executed;
- $\mathcal{D}_\alpha$ is the set of propositions that characterise those facts made false by the performance of $\alpha$ (the *delete list*); and
- $\mathcal{A}_\alpha$ is the set of propositions that characterise those facts made *true* by the performance of $\alpha$ (the *add list*).

As with RMGs, we model the state of the system at any given time as a valuation $v \in V$. For every action $\alpha$ we define a propositional formula $pre_\alpha$, such that this formula is satisfied by a valuation whenever the pre-condition of $\alpha$ is met:

$$pre_\alpha = \bigwedge_{x \in \mathcal{T}_\alpha} x \wedge \bigwedge_{y \in \mathcal{F}_\alpha} \neg y.$$

The effect of executing an action $\alpha$ when the world is as described by the valuation $v$ is denoted $eff(\alpha, v)$, where this partial function is defined as follows:

$$eff(\alpha, v) = \begin{cases} (v \setminus \mathcal{D}_\alpha) \cup \mathcal{A}_\alpha & \text{if } v \models pre_\alpha \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We can then naturally define a *multi-agent STRIPS arena*:

$$A = (N, \Phi, v_0, Ac_1, \ldots, Ac_n)$$

where $N = \{1, \ldots, n\}$ is the set of players, $\Phi$ is the vocabulary of propositional variables used to represent the state of the system, $v_0 \in V$ is the initial state of the game, and for each player $i \in N$, $Ac_i = \{\alpha_i^1, \ldots, \alpha_i^k\}$ is set of action descriptors, representing the actions that $i$ can perform. The *domain* of an action $\alpha$, denoted dom $\alpha$, is the set of variables whose truth status can be changed by the performance of $\alpha$, i.e., dom $\alpha = \mathcal{D}_\alpha \cup \mathcal{A}_\alpha$. To keep things simple, we assume that the actions of different players are disjoint:

$$\forall i \neq j \in N, \forall \alpha_1 \in Ac_i, \forall \alpha_2 \in Ac_j, \text{dom}\, \alpha_1 \cap \text{dom}\, \alpha_2 = \emptyset.$$

Let $\Phi_i = \bigcup_{\alpha \in Ac_i} \text{dom}\, \alpha$ be the set of variables under the control of player $i$.

The execution of a STRIPS arena then proceeds over an infinite sequence of rounds: in the first round, variables are initialised to their values in $v_0$; on each subsequent round, each player $i$ selects an action $\alpha \in Ac_i$ whose pre-condition is satisfied, and the profile of actions selected by players are then executed as defined in the $eff(\cdots)$ function to generate the next state of the system. Each player then chooses another action whose pre-condition is satisfied, and so on.

A *STRIPS game* is then given by a structure

$$G = (A, \gamma_1, \ldots, \gamma_n)$$

where $A$ is a multi-agent STRIPS arena, and $\gamma_i$ is the temporal logic goal of player $i$; as with RMGs, we might consider LTL or CTL goals. Given this setup, it should be clear that we can then define strategies for agents in the same way that we did for RMGs, and then runs, Kripke structures, etc. similarly. (The details are straightforward but tedious, and we omit them here.) We remark that the definition we give here is very similar to that of the MA-STRIPS model of Brafman and Domshlak [10], except that in MA-STRIPS, goals are represented as sets of states, with the idea being that an agent wants to bring about one of the set. This can be represented directly within our model by giving agents goals of the form $\mathbf{F}\varphi$.

We now show how we can translate a STRIPS arena into an equivalent RML arena. First, let $\alpha = (\mathcal{T}_\alpha, \mathcal{F}_\alpha, \mathcal{D}_\alpha, \mathcal{A}_\alpha)$ be an action descriptor. Then we define a guarded command $g_\alpha$ for this action descriptor as follows:

$$pre_\alpha \rightsquigarrow \underbrace{r_1' := \top; \cdots; r_k' := \top;}_{\mathcal{A}_\alpha = \{r_1, \ldots, r_k\}} \underbrace{z_1' := \bot; \cdots; z_k' := \bot}_{\mathcal{D}_\alpha = \{z_1, \ldots, z_k\}}$$

We define a single **init** guarded command $g_i^0$ for player $i$, which sets the variables under the control of $i$ to correspond to the valuation $v_0$:

$$\top \rightsquigarrow \underbrace{x_i' := \top; \cdots;}_{x_i \in \Phi_i \cap v_0} \underbrace{x_j' := \bot; \cdots;}_{x_j \in (\Phi_i \setminus v_0)}$$

Finally, the module $m_i$ for each player $i$ is then simply $m_i = (\Phi_i, I_i, U_i)$ where $\Phi_i$ is as defined above, $I_i = \{g_i^0\}$ is the single initialisation guarded command for $i$ as defined above, and $U_i = \{g_\alpha \mid \alpha \in Ac_i\}$ is the set of update guarded commands for $i$.

Now, it should be clear from construction that the SRML arena constructed in this way will operate in the same way as the STRIPS arena that we began with. In particular, when each player $i$ selects an enabled **update** command $g_\alpha$ for execution, this corresponds to selecting an action $\alpha \in Ac_i$ whose pre-condition is satisfied. Executing a guarded command simulates the behaviour of executing the action $\alpha$ as defined by the descriptor $(\mathcal{T}_\alpha, \mathcal{F}_\alpha, \mathcal{D}_\alpha, \mathcal{A}_\alpha)$.

Thus, we can translate propositional multi-agent STRIPS systems into our formalism. Note that translation in the opposite direction is not directly possible, however, because SRML guarded commands are richer than propositional STRIPS operators: in the action part of a guarded command, we are allowed to have actions such as $x' := \varphi$, meaning that the variable $x$ should take the value of the formula $\varphi$. Such constructs are not provided in propositional STRIPS (they can be simulated, but only at the cost of an exponential blow up in the number of actions).

Finally, note that the fact that propositional STRIPS planning games can be encoded within our game framework implies that membership results for the corresponding decision problems studied in the article carry over directly (hardness results do not carry over automatically, although many of them can be seen to hold from the constructions presented earlier).

## 8. Related work

Our work has its antecedents in two closely related threads of computer science research: the first, arising from the computer-aided verification community, is the use of logics and techniques inspired by game theory for the automated verification of computer programs (see, e.g., [3]); the second, arising largely from the artificial intelligence community, is the use of logic in the formal specification and verification of multi-agent systems (see, e.g., [76,68] for surveys).

Historically, the computer-aided verification community focussed on the problem of checking whether a system $S$ satisfies a given property $\varphi$, where $\varphi$ is expressed as a formula of temporal logic (typically either LTL or CTL) [20,16]. Using temporal logics such as LTL or CTL makes it possible to express properties relating to the temporal ordering of events in a system, but does not enable the expression of *strategic* properties, such as "component $i$ can guarantee that the system never enters a fail state." This observation motivated Alur et al. to develop Alternating-time Temporal Logic (ATL) [3], which is explicitly intended to support such reasoning. However, there are important limitations on the game properties that can be expressed using ATL. First, ATL provides no direct mechanism to refer to *strategies* in the object language – which appears to be required if one intends to capture game-theoretic properties such as Nash equilibrium. Second, the language provides no mechanism for directly representing the *preferences* of system components – and again, this seems appropriate if one hopes to reason about game-theoretic equilibrium properties, as we do in this paper.

The idea of introducing strategies into the object language has been pursued by a number of researchers [14,50]. Probably the best-known and most successful such formalism is *Strategy Logic* (SL), which can be thought of as a temporal logic that includes a framework for naming strategies and reasoning about their properties [66,14,50].

Relatively little work within the computer aided verification community has addressed the problem of capturing the preferences of players, and the role that these preferences play in strategic, game theoretic reasoning. However, of the work on this topic that is present in the literature, it seems fairly common to model player preferences in the same way that we do in the present paper: as temporal logic formulae that the player desires to see satisfied [5,23] (or else, more generally, as $\omega$-regular objectives [13,8]). To pick one example of this work, Alur et al. [5] investigated the complexity of game-theoretic questions for games played on graphs, in which players attempt to satisfy a goal, expressed as a temporal logic formula. One aspect of such work is that it assumes that agents make moves in a game arena that is explicitly modelled as a graph. This is an unrealistic assumption in practice, since the state transition graph for any real system will be of size exponential in the number of variables in the system. Moreover, the significance of complexity results for such games might reasonably be questioned, since an unrealistic representation of the game arena is assumed. Similar comments apply to many other studies of game-theoretic properties of concurrent systems.

In the second strand of work, researchers in the AI community have developed a range of formalisms intended for the specification and verification of multi-agent systems. Originally, this work focussed on practical reasoning agents, (i.e., planning systems [25]), and in particular, on the belief-desire-intention model of agency [58]. In this model, agents are viewed as selecting intentions from a set of possible desires, which then represent commitments to achieve certain states of affairs; they then select and execute plans of action to achieve their intentions. However, difficulties in obtaining a satisfactory formalisation of rational mental states led researchers to seek alternative foundations for modelling rational action in multi-agent systems, and game theory became increasingly adopted as a foundation. See, e.g., [39] for an excellent survey of this stream of work. Of this body of work, it is worth identifying several threads of research that are related to the present paper. Bulling et al. [11] consider variations of ATL with the explicit intention of reasoning about temporal properties of strategic notions such as Nash equilibrium. The key differences with the present work are, first, that our aim is not to reason about game theoretic properties in the object language; and second, our use of a compact system modelling language (SRML) to define the underpinning game structure.

Our work in the present article draws upon both of these paradigms, and it is most closely related to the work presented in [29,30,33]. In [29], Gutierrez et al. introduced a model called *Iterated Boolean Games* (iBGs). In an iBG, each player exercises unique control over a set of Boolean variables, and play proceeds over an infinite series of rounds, where at each round each player makes a Boolean assignment to the variables they control. Each player is associated with an LTL formula that the player desires to see satisfied in the LTL model traced out by the infinite play of the game. A key concern of Gutierrez et al. was to classify the complexity of game-theoretic decision problems in iBGs (such as checking for the existence of pure strategy Nash equilibria).[5] Observe that, in contrast to many other studies of game properties in concurrent systems, the arena in which an iBG is played is defined *succinctly*: simply by listing the variables that each player controls. While we believe the iBG model and the Boolean games model that it generalises are simple and elegant models with which to formulate questions of strategic interaction in multi-agent systems, they clearly represent a very high-level abstraction of real systems. In particular, in iBGs, it is assumed that at every time point, every player has complete freedom to assign values to the variables it controls. This greatly limits the multi-agent scenarios that can be modelled with iBGs. In [30,33], Gutierrez et al. presented a more realistic model of concurrent programs, but as with other studies of game-like properties in concurrent systems, it assumes that game arenas are explicitly presented – and for the reasons described above, we believe this is an unrealistic assumption in practice.

[5] The iBG model is essentially a generalisation of the conventional Boolean games model introduced by Harrenstein et al. [37]. Conventional Boolean games are like iBGs, except that play takes just one round, and goals are expressed using classical propositional logic, rather than LTL.

**Table 2**
Overview of computational complexity results.

| | LTL RMGs | | CTL RMGs | |
|---|---|---|---|---|
| Realizability | 2EXPTIME-c | (Proposition 8) | 2EXPTIME-c | (Proposition 14) |
| NE-membership | PSPACE-c | (Proposition 10) | 2EXPTIME-c | (Proposition 15) |
| Non-emptiness | 2EXPTIME-c | (Proposition 11) | 2EXPTIME-hard | (Proposition 16) |
| E-Nash | 2EXPTIME-c | (Proposition 13) | 2EXPTIME-hard | (Proposition 16) |
| A-Nash | 2EXPTIME-c | (Proposition 13) | 2EXPTIME-hard | (Proposition 16) |

In short, while our broad aim is the same as [5,11,13,8,29,30,33] (i.e., the game-theoretic analysis of concurrent and multi-agent systems), our approach is to use a language for defining game arenas that is much closer to real-world programming and system modelling. We represent game arenas using the Reactive Modules language, which permits the *succinct* definition of concurrent and multi-agent systems and protocols. Questions relating to the complexity of game-theoretic questions on such succinctly-specified arenas are, we believe, much more meaningful in this context than on arenas in which the game graph is explicitly presented. Interestingly, we find that for many of the decision problems we consider, the complexity does not increase when studying game arenas specified using Reactive Modules.

## 9. Conclusion and future work

In this article, we have studied a novel game model that admits a natural characterisation in a simple, yet computationally powerful, syntactic fragment of the Reactive Modules language [2]. Also, the complexity of reasoning about equilibria in this model as well as of checking and synthesising temporal properties of multi-agent systems specified in such a language was investigated, both, in the linear-time and branching-time frameworks [20].

Several research groups are working on problems related to equilibrium checking: Fisman et al. [23] study the problem of synthesising systems so that certain desirable properties hold in equilibrium; in addition, extensive overviews of decision problems related to equilibrium checking are given in [13,8], for a range of concurrent game models.

In our own work, we have investigated many issues surrounding rational verification, particularly using *Boolean games* [37]. Boolean games are essentially "one-shot" versions of iterated Boolean games, as described above, where play consists of a single round, and agent goals $\gamma_i$ are specified as propositional formulae. A question that we have investigated at length is possible mechanisms for *managing* games. This might be necessary, for example, if the game contains socially undesirable equilibria [73], or where a game possesses no equilibrium, and we wish to introduce one (which we call *stabilisation*). One obvious mechanism for manipulating games is the use of taxation schemes, which provide incentives for players to avoid undesirable equilibria, or to prefer desirable equilibria [74]. (Related issues have recently been studied in the context of concurrent games [1].) Another possibility is to try to influence players by changing their beliefs through communication. For example, [26] considered a Boolean games setting where players make their choices based on beliefs about some variables in the environment, and a central authority is able to reveal certain information in order to modify these beliefs. Another issue we have investigated is the extent to which we can develop a language that supports reasoning about strategies directly in the object language. Strategy Logic (SL) is a variation of temporal logic, closely related to Alternating-time temporal logic [3], which includes names for strategies in the object language [14], and using SL, it is possible to reason about Nash equilibria. However, SL is in general undecidable, which raises the question of whether weaker languages might be used. For instance, in [30] a temporal logic containing a quantifier $[NE]\varphi$, meaning "$\varphi$ holds on all Nash equilibrium computations", is proposed. Also, in [31] it is shown that the existence of Nash equilibria can be represented, up to bisimilarity, in logic languages that are even weaker than full SL. Other researchers have investigated similar concerns [11]. Another interesting question is the extent to which the model of interaction used in a particular setting affects the possible equilibria that may result. In [35], we investigated Nash equilibria in games based on event structures [71], and were able to characterise conditions required for the existence of equilibria. Another research direction is the extent to which we can go beyond the representation of preferences as simple binary formulae; one possible approach, investigated in [48], is to represent player's goals as formulae of Łukasiewicz logic, which permits a much richer class of preferences to be directly represented.

Many issues are being currently studied, or remain for future work. Mixed (stochastic) strategies is an obvious major topic of interest, as it is the possibility of imperfect information [34], and of course solution concepts beyond Nash equilibria, such as subgame perfect equilibrium. Our Eagle tool is a prototype [65], limited in its scope, and not optimised: extensions supporting LTL, incomplete information, etc., are all highly desirable.

Our main *complexity* results are summarised in Table 2 and some of them are rather surprising. For instance, we showed that with respect to goals given by LTL formulae and deterministic strategies, it is equally hard to verify *equilibrium properties* (i.e., those studied in E-Nash and A-Nash) regardless of whether the arenas modelling the systems of interest are either explicitly specified in srml or implicitly given – as in iterated Boolean games (iBGs [29]), where the induced Kripke structures are exponential in the number of variables. Recall that, in general, the srml model of any iBG is also of exponential size.

Less surprising is the fact that the complexity of all the key questions we studied is 2EXPTIME-hard. This is high computational complexity, which presents a real challenge for practical implementations. Our results show that, as for (LTL)

```
function Arena2Kripke(A = (N, Φ, m_1, ..., m_i, ..., m_n))
1.  S_A := ∅, S_A^0 := ∅, R_A := ∅, π_A := ∅
2.  C_1 := I_1; ··· ; C_n := I_n;
3.  for each J ∈ C_1 × ··· × C_n do
4.      S_A := S_A ∪ {exec(J, v_⊥)}
5.      S_A^0 := S_A
6.      π_A := π_A ∪ {(exec(J, v_⊥), exec(J, v_⊥))}
7.  end-for
8.  X := ∅
9.  while X ≠ S_A do
10.   X := S_A
11.     for each v ∈ S_A do
12.       C_1 := enabled_1(π_A(v)); ... ; C_n := enabled_n(π_A(v))
13.       for each J ∈ C_1 × ··· × C_n do
14.           S_A := S_A ∪ {exec(J, π_A(v))}
15.           π_A := π_A ∪ {(exec(J, π_A(v)), exec(J, π_A(v)))}
16.       end-for
17.     end-for
18. end-while
19. for each (v, v') ∈ S_A × S_A do
20    C_1 := enabled_1(π_A(v)), ..., C_n := enabled_n(π_A(v))
21.   for each J ∈ C_1 × ··· × C_n do
22.     if exec(J, π_A(v)) = π_A(v') do
23.       R_A := R_A ∪ {(v, v')}
24.     end-if
25.   end-for
26. end-for
```

**Fig. 12.** Algorithm for generating Kripke structures. Here, $K_A = (S_A, S_A^0, R_A, \pi_A)$ is the output of the function, produced from the SRML arena $A = (N, \Phi, m_1, \ldots, m_n)$ passed as input.

synthesis problems, this is the best that one can do. As a consequence, if practical implementations are desired, one would have to look for suitable simplifications to the main setting: for instance, games with memoryless strategies, or games with goals in simpler LTL syntactic fragments. In the former case, one cannot do better than EXPTIME for general LTL goals; in the latter case, the complexity may be reduced even to PSPACE for safety goals. Either way, any practical implementation will probably have to deal with an associated *synthesis* problem from a logical specification, which most likely will be of high complexity. In fact, even though we deal with succinct system specifications, which are closer to the ones used in realistic scenarios, as clearly shown in [5], high complexities would be obtained even if explicit system representations are used, simply due to the fact that players goals are expressed using the full language of LTL. An alternative is studied in [28], where Nash equilibria is studied with respect to fragments of LTL within the iBG framework.

Moreover, from an *expressivity* standpoint, RMGs are more general than iBGs. Indeed, the class of induced Kripke structures induced by RMGs is strictly *larger* than the class of semantic structures associated with iBGs. A simple example is the distributed algorithm in Section 5.3, which cannot be specified as an iBG, but has an SRML representation. Then, with RMGs, we have *more expressive power with the same complexity*.

A somewhat different question is the extent to which the techniques and solution concepts developed here can be used in practice, for example when designing electronic commerce systems. The issue here touches on a substantial debate within game theory itself, concerning the applicability of game-theoretic concepts. If we view game theory as being a *descriptive* theory, which aims to predict what people will actually do in strategic settings, then game-theoretic solution concepts seem to be of limited value in many cases (see, e.g., [7] for extensive discussion). There are several points to make here. First, one can argue that while *people* fail to be rational in the sense that game theory defines, when we design *computer programs* to make decisions, we can design them to act rationally. Second, we suggest that Nash equilibrium analysis is a *starting point* for understanding the rational dynamics of a system, rather than an end point. The Nash equilibrium computations of an RMG are a subset of the overall possible computations of the system at hand, and assuming rational action, then the *actual* trajectory of the system will lie with the Nash equilibrium set. But the analysis of the present paper stops at this point, once we have identified the rational computations. Finally, if we aim to use our techniques to understand the rational dynamics of *human* system, then work is needed on developing computational models of human strategic decision making. This would take us into the territory of *behavioural game theory*, and is somewhat beyond the scope of the present paper.

In future work, we plan to use our specification language and games framework to implement a *model checker of equilibrium properties* in game-like concurrent and multi-agent systems. It would be also very interesting to investigate how the complexity results we have obtained so far are affected by the adoption of other temporal logic languages and solution concepts. Studying the relationship to Markov games and probabilistic model checking would also be interesting, perhaps using logics such as PCTL [43]. It would also be interesting to study situations in which the actions of players have unreliable effects (i.e., non-deterministic actions). Finally, much work remains to be done in settings where one considers imperfect information [34] or more general payoff sets and preference relations. All these issues should certainly be also investigated further.

```
function realizable(i, φ, s)
1.  ℓ := ∅
2.  let L be the set osf(φ) ordered according to length (shortest first)
3.  for c := 1 to |L| do
4.     repeat
5.        for each s′ ∈ S_A do
6.           if L[c] is a propositional formula then
7.              return π_A(s′) ⊨ L[c]
8.           elsif L[c] = X ψ then
9.              return can(i, ψ, s′)
10.          elsif L[c] = ψ U χ then
11.             if χ ∈ ℓ(s′) or
12.                (ψ ∈ ℓ(s′) and can(i, ψ U χ, s′)) then
13.                   ℓ(s′) := ℓ(s′) ∪ {ψ U χ}
14.             end-if
15.          end-if
16.       end-for
17.    until no change to ℓ
18. end-for
19. if φ ∈ ℓ(s) then return ⊤ else return ⊥

function can(i, φ, s)
1.  C_1 := enabled_1(π_A(s)); ··· ; C_n := enabled_n(π_A(s))
2.  flag := ⊥
3.  for each g ∈ C_i
4.     T := ∅
5.     for each J ∈ C_1 × ··· × {g} × ··· C_n
6.        T := T ∪ {exec(J, π_A(s))}
7.     end-for
8.     if T = {s′ : s′ ∈ T and φ ∈ ℓ(s′)} then
9.        flag := ⊤
10.    end-if
11. end-for
12. return flag
```

**Fig. 13.** Algorithm for checking whether a player $i$ can realise a formula $\varphi$ from a state $s$.


## Acknowledgements

We would like to thank Giuseppe Perelli and Alexis Toumi for very useful comments on an earlier version of this paper. Richard Appleby helped us debug some of the algorithms. This work was financially supported by the ERC Grant 291528 ("RACE") at Oxford. The paper includes motivational/background material from the AAAI-2016 paper [75], although we emphasise that all the results presented in the present paper are new.


## Appendix A. Algorithms from the main text

In this appendix, we present algorithms that we held over from the main text in the interests of readability.

First, Fig. 12 presents an algorithm that takes as input an RML arena $A$, and from this arena computes the Kripke structure induced by $A$. From inspection it is straightforward to see that the algorithm is correct, and is guaranteed to terminate in time exponential in $|A|$. Lines (3)–(7) compute the initial states $S^0_A$. Lines (8)–(18) then extend these initial states to compute all reachable states of the system. Finally, lines (19)–(26) compute the relation $R_A$.

Fig. 13 presents an algorithm for checking whether an OLTL formula $\varphi$ is realizable for a player $i$ from a state $s$ in a Kripke structure $K_A = (S_A, S^0_A, R_A, \pi_A)$. The algorithm works by backwards induction, building up a labelling function $\ell$, which associates with each state in $S_A$ the formulae that $i$ can realize from that state. The algorithm is closely related to Zermelo's algorithm, and critically depends upon the fact that the input is an OLTL formula.

The algorithm makes use of a subsidiary function $can(i, \varphi, s)$, which checks to see whether player $i$ has a guarded command available for execution in state $s$, such that in all states that could result from the execution of this guarded command, the formula $\varphi$ is true.

It is straightforward from construction that the algorithm is correct, and guaranteed to terminate in time exponential in the size of $A$.

## References


[1] S. Almagor, G. Avni, O. Kupferman, Repairing multi-player games, in: L. Aceto, D. de Frutos Escri (Eds.), Proceedings of the Twenty-Sixth Annual Conference on Concurrency Theory, CONCUR'15, Madrid, Spain, in: Leibniz International Proceedings in Informatics (LIPIcs), 2015, pp. 325–339.
[2] R. Alur, T.A. Henzinger, Reactive modules, Form. Methods Syst. Des. 15 (11) (1999) 7–48.
[3] R. Alur, T.A. Henzinger, O. Kupferman, Alternating-time temporal logic, J. ACM 49 (5) (2002) 672–713.
[4] R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, S. Taşiran, MOCHA: modularity in model checking, in: Proceedings of the 10th International Conference on Computer Aided Verification, CAV'98, in: LNCS, vol. 1427, Springer, 1998, pp. 521–525.
[5] R. Alur, S. La Torre, P. Madhusudan, Playing games with boxes and diamonds, in: Proceedings of the Fourteenth Annual Conference on Concurrency Theory, CONCUR'03, in: LNCS, vol. 2761, Springer, 2003, pp. 127–141.
[6] K. Binmore, Fun and Games: A Text on Game Theory, D.C. Heath and Company, Lexington, MA, 1992.
[7] K. Binmore, Does Game Theory Work?, The MIT Press, Cambridge, MA, 2007.
[8] P. Bouyer, R. Brenguier, N. Markey, M. Ummels, Pure Nash equilibria in concurrent games, Log. Methods Comput. Sci. 2 (2015) 9.
[9] R.S. Boyer, J.S. Moore (Eds.), The Correctness Problem in Computer Science, The Academic Press, London, England, 1981.
[10] R. Brafman, C. Domshlak, On the complexity of planning for agent teams and its implications for single agent planning, Artif. Intell. 198 (2013) 52–71.
[11] N. Bulling, W. Jamroga, J. Dix, Reasoning about temporal properties of rational play, Ann. Math. Artif. Intell. 53 (1–4) (2008) 51–114.
[12] T. Bylander, The computational complexity of propositional STRIPS planning, Artif. Intell. 69 (1–2) (1994) 165–204.
[13] K. Chatterjee, T.A. Henzinger, A survey of stochastic $\omega$-regular games, J. Comput. Syst. Sci. 78 (2012) 394–413.
[14] K. Chatterjee, T.A. Henzinger, N. Piterman, Strategy logic, Inf. Comput. 208 (6) (2010) 677–693.
[15] E.M. Clarke, E.A. Emerson, Design and synthesis of synchronization skeletons using branching time temporal logic, in: Proceedings of the Workshop on Logics of Programs, in: LNCS, vol. 131, Springer-Verlag, Berlin, Germany, 1981, pp. 52–71.
[16] E.M. Clarke, O. Grumberg, D.A. Peled, Model Checking, The MIT Press, Cambridge, MA, 2000.
[17] G. Coulouris, J. Dollimore, T. Kindberg, Distributed Systems: Concepts and Design, Addison-Wesley, 2005.
[18] S. Demri, V. Goranko, M. Lange, Temporal Logics in Computer Science: Finite-State Systems, Cambridge Tracts in Theoretical Computer Science, vol. 58, Cambridge University Press, 2016.
[19] S. Demri, Ph. Schnoebelen, The complexity of propositional linear temporal logics in simple cases, Inf. Comput. 174 (1) (2002) 84–103.
[20] E.A. Emerson, Temporal and modal logic, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, Elsevier, 1990, pp. 996–1072.
[21] E.A. Emerson, E.M. Clarke, Using branching time temporal logic to synthesize synchronization skeletons, Sci. Comput. Program. 2 (3) (1982) 241–266.
[22] R.E. Fikes, N. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, Artif. Intell. 2 (1971) 189–208.
[23] D. Fisman, O. Kupferman, Y. Lustig, Rational synthesis, in: 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'10, in: LNCS, vol. 6015, Springer, 2010, pp. 190–204.
[24] D. Gale, L.S. Shapley, College admissions and the stability of marriage, Am. Math. Mon. 69 (1) (1962) 9–15.
[25] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufmann Publishers, San Mateo, CA, 2004.
[26] J. Grant, S. Kraus, M. Wooldridge, I. Zuckerman, Manipulating games by sharing information, Stud. Log. 102 (2014) 267–295.
[27] D. Gusfield, R.W. Irving, The Stable Marriage Problem: Structure and Algorithms, MIT Press, Cambridge, MA, USA, 1989.
[28] J. Gutierrez, P. Harrenstein, G. Perelli, M. Wooldridge, Expressiveness and Nash equilibrium in iterated Boolean games, in: C.M. Jonker, S. Marsella, J. Thangarajah, K. Tuyls (Eds.), Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems, AAMAS'16, ACM, 2016, pp. 707–715.
[29] J. Gutierrez, P. Harrenstein, M. Wooldridge, Iterated Boolean games, in: F. Rossi (Ed.), Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI'13, IJCAI/AAAI Press, 2013, pp. 932–938.
[30] J. Gutierrez, P. Harrenstein, M. Wooldridge, Reasoning about equilibria in game-like concurrent systems, in: C. Baral, G. De Giacomo, Th. Eiter (Eds.), Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning, KR'14, Vienna, Austria, AAAI Press, 2014.
[31] J. Gutierrez, P. Harrenstein, M. Wooldridge, Expressiveness and complexity results for strategic reasoning, in: Proceedings of the Twenty-Sixth Annual Conference on Concurrency Theory, CONCUR'15, Madrid, Spain, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 42, 2015, pp. 268–282.
[32] J. Gutierrez, P. Harrenstein, M. Wooldridge, Iterated Boolean games, Inf. Comput. 242 (2015) 53–79.
[33] J. Gutierrez, P. Harrenstein, M. Wooldridge, Reasoning about equilibria in game-like concurrent systems, Ann. Pure Appl. Logic 169 (2) (2017) 373–403.
[34] J. Gutierrez, G. Perelli, M. Wooldridge, Imperfect information in reactive modules games, in: C. Baral, J.P. Delgrande, F. Wolter (Eds.), Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning, KR'16, AAAI Press, 2016, pp. 390–400.
[35] J. Gutierrez, M. Wooldridge, Equilibria of concurrent games on event structures, in: Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic, CSL'14, and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'14, Vienna, Austria, 2014, Article No. 46.
[36] P. Harrenstein, Logic in Conflict, PhD thesis, Utrecht University, 2004.
[37] P. Harrenstein, W. van der Hoek, J.-J.Ch. Meyer, C. Witteveen, Boolean games, in: J. van Benthem (Ed.), Proceedings of the Eighth Conference on Theoretical Aspects of Rationality and Knowledge, TARK'01, Siena, Italy, 2001, pp. 287–298.
[38] M. Hennessy, R.A. Connolly Milner, Algebraic laws for nondeterminism and concurrency, J. ACM 32 (1) (1985) 137–161.
[39] W. Jamroga, W. Penczek, Specification and verification of multi-agent systems, in: N. Bezhanishvili, V. Goranko (Eds.), Lectures on Logic and Computation, in: LNCS, vol. 7388, Springer, 2012, pp. 210–263.
[40] L. Kleinrock, Analysis of a time-shared processor, Nav. Res. Logist. Q. 11 (1) (1964) 59–73.
[41] O. Kupferman, M.Y. Vardi, P. Wolper, An automata-theoretic approach to branching time model checking, J. ACM 47 (2) (2000) 312–360.
[42] Orna Kupferman, P. Madhusudan, P.S. Thiagarajan, Moshe Y. Vardi, Open systems in reactive environments: control and synthesis, in: C. Palamidessi (Ed.), Proceedings of the Eleventh International Conference on Concurrency Theory, CONCUR'00, in: LNCS, vol. 1877, Springer, 2000, pp. 92–107.
[43] M. Kwiatkowska, G. Norman, D. Parker, PRISM: probabilistic model checking for performance and reliability analysis, ACM SIGMETRICS Perform. Eval. Rev. 36 (4) (2009) 40–45.
[44] L. Lamport, A new solution of Dijkstra's concurrent programming problem, Commun. ACM 17 (8) (1974) 453–455.
[45] D. Manlove, Algorithmics of Matching Under Preferences, World Scientific Publishing Company, 2013.
[46] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems, Springer, 1992.
[47] Z. Manna, A. Pnueli, Temporal Verification of Reactive Systems — Safety, Springer, 1995.
[48] E. Marchioni, M. Wooldridge, Łukasiewicz games: a logic-based approach to quantitative strategic interactions, ACM Trans. Comput. Log. 16 (4) (2015), Article No. 33.
[49] M. Maschler, E. Solan, S. Zamir, Game Theory, Cambridge University Press, Cambridge, England, 2013.
[50] F. Mogavero, A. Murano, G. Perelli, M.Y. Vardi, Reasoning about strategies: on the model-checking problem, ACM Trans. Comput. Log. 15 (4) (2014).

[51] F. Mogavero, A. Murano, M.Y. Vardi, Reasoning about strategies, in: K. Lodaya, M. Mahajan (Eds.), Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'10, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 8, 2010, pp. 133–144.
[52] N. Nisan, T. Rougarden, E. Tardos, V.V. Vazirani (Eds.), Algorithmic Game Theory, Cambridge University Press, Cambridge, England, 2007.
[53] M.J. Osborne, A. Rubinstein, A Course in Game Theory, The MIT Press, Cambridge, MA, 1994.
[54] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, Reading, MA, 1994.
[55] G.L. Peterson, Myths about the mutual exclusion problem, Inf. Process. Lett. 12 (3) (1981) 115–116.
[56] A. Pnueli, The temporal logic of programs, in: Proceedings of the Eighteenth IEEE Symposium on the Foundations of Computer Science, FOCS'77, The Society, 1977, pp. 46–57.
[57] A. Pnueli, R. Rosner, On the synthesis of an asynchronous reactive module, in: G. Ausiello, M. Dezani-Ciancaglini, S. Ronchi Della Rocca (Eds.), Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programs, ICALP'89, 1989, pp. 652–671.
[58] A.S. Rao, M. Georgeff, Decision procedures for BDI logics, J. Log. Comput. 8 (3) (1998) 293–344.
[59] G. Ricart, A.K. Agrawala, An optimal algorithm for mutual exclusion in computer networks, Commun. ACM 24 (1) (1981) 9–17.
[60] A.E. Roth, M.A.O. Sotomayor, Two-Sided Matching: A Study in Game Theoretic Modelling and Analysis, Cambridge University Press, 1990.
[61] T.C. Schelling, The Strategy of Conflict, Harvard University Press, 1981.
[62] Y. Shoham, K. Leyton-Brown, Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge University Press, Cambridge, England, 2008.
[63] A.P. Sistla, E.M. Clarke, The complexity of propositional linear temporal logics, J. ACM 32 (3) (1985) 733–749.
[64] L.J. Stockmeyer, A.K. Chandra, Provably difficult combinatorial games, SIAM J. Comput. 8 (2) (1979) 151–174.
[65] A. Toumi, J. Gutierrez, M. Wooldridge, A tool for the automated verification of Nash equilibria in concurrent games, in: M. Leucker, C. Rueda, F.D. Valencia (Eds.), Proceedings of the Twelfth International Colloquium on Theoretical Aspects of Computing, ICTAC'15, in: LNCS, vol. 9399, Springer, 2015, pp. 583–594.
[66] W. van der Hoek, W. Jamroga, M. Wooldridge, A logic for strategic reasoning, in: M. Pechoucek, D. Steiner, S. Thompson (Eds.), Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS'05, 2005, pp. 157–164.
[67] W. van der Hoek, A. Lomuscio, M. Wooldridge, On the complexity of practical ATL model checking, in: H. Nakashima, M.P. Wellman, G. Weiss, P. Stone (Eds.), Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS'06, ACM, 2006, pp. 201–208.
[68] W. van der Hoek, M. Wooldridge, Towards a logic of rational agency, Log. J. IGPL 11 (2) (2003) 135–159.
[69] M.Y. Vardi, P. Wolper, An automata-theoretic approach to automatic program verification, in: First Symposium in Logic in Computer Science, LICS'86, IEEE Computer Society, 1986, pp. 322–331.
[70] Moshe Y. Vardi, Thomas Wilke, Automata: from logics to algorithms, in: Logic and Automata: History and Perspectives, in: Texts in Logic and Games, vol. 2, Amsterdam University Press, 2008, pp. 629–736.
[71] Glynn Winskel, Event structures, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.), Advances in Petri Nets 1986. Proceedings of an Advanced Course, Bad Honnef, 8.–19. September 1986, Part 2: Petri Nets: Applications and Relationships to Other Models of Concurrency, in: LNCS, vol. 255, 1987, pp. 325–392.
[72] M. Wooldridge, An Introduction to Multiagent Systems, second edition, John Wiley & Sons, 2009.
[73] M. Wooldridge, Bad equilibria (and what to do about them), in: L. De Raedt, C. Bessiere, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, P. Lucas (Eds.), Proceedings of the Twentieth European Conference on Artificial Intelligence, ECAI'12, Montpellier, France, 2012, pp. 6–11.
[74] M. Wooldridge, U. Endriss, S. Kraus, J. Lang, Incentive engineering for boolean games, Artif. Intell. 195 (2013) 418–439.
[75] M. Wooldridge, J. Gutierrez, P. Harrenstein, E. Marchioni, G. Perelli, A. Toumi, Rational verification: from model checking to equilibrium checking, in: D. Schuurmans, M. Wellman (Eds.), Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, Phoenix, AZ, 2016, pp. 4184–4190.
[76] M. Wooldridge, N.R. Jennings, Intelligent agents: theory and practice, Knowl. Eng. Rev. 10 (2) (1995) 115–152.