Model Checking Games for a Fair Branching-Time Temporal Epistemic Logic ⋆

Xiaowei Huang and Ron van der Meyden

The University of New South Wales, Australia.
{xiaoweih,meyden}@cse.unsw.edu.au

Abstract. Model checking games are instances of Hintikka's game semantics for logic used for purposes of debugging systems verification models. Previous work in the area has developed these games for branching time logic. The paper develops an extension to a logic that adds epistemic operators, and interprets the branching time operators with respect to fairness constraints. The implementation of the extended games in the epistemic model checker MCK is described.

1 Introduction

Model checking is a technique used in computer science for the verification of systems designs. Traditionally, model checkers deal with specifications expressed in a variant of temporal logic — this class of model checkers is now widely applied to the verification of computer hardware and computer network communications protocols. In recent years, a number of model checkers have been developed that are based on modal logics that combine temporal and epistemic modalities [1–4]. These enable the analysis of systems from the perspective of information theoretic properties, and have been applied to problems such as the verification of security protocols [5] and the verification of knowledge-based programs [1, 6]. In the context of Artificial Intelligence, epistemic logic has been the focus of a line of work in the multi-agent systems literature, [7, 8], where it is used for reasoning about systems of communicating agents.

One of the reasons for the success of model checking technology is that, at least in the case of linear-time temporal logic specifications, it is possible for a model checker to return to the user a *counter-example*, in the form of an "error-trace" which illustrates a possible execution of the system on which the specification fails. This provides concrete information that helps the user to diagnose the source of the error.

For branching-time temporal logics, the situation is somewhat more complex: while counter-examples can be defined [9], in general, they have a structure that is neither easily presented to the user nor easily comprehended, since, rather than a single execution of the system, one needs to deal with multiple executions, in a complicated branching structure. Once one considers temporal and epistemic logics, it becomes even less clear how to make counter-examples comprehensible, since epistemic operators require even greater flexibility to move between different points of different executions of the system.

⋆ Work supported by Australian Research Council Linkage Grant LP0882961 and Defence Research and Development Canada (Valcartier) contract W7701-082453.

In the setting of branching-time temporal logic, the complexity of counter-examples has motivated the application of ideas based on Hintikka's game theoretical semantics for logic [10] as an interactive debugging tool. Game theoretical semantics characterizes the truth of a formula in a model in terms of the existence of a winning strategy in a game constructed from the formula and model. In Hintikka games, there are two players: a verifier, whose objective in the game is to justify that the formula holds in the model, and a refuter, whose objective is to show that the formula is false. The rules of the game are constructed so that steps of the game proceed from a formula to its subformulas, with moves corresponding to cases of the recursive semantics of the logic, with players taking turns depending on the structure of the subformula under consideration. If there exists a winning strategy for verifier, then the assertion holds, otherwise refuter has a winning strategy, and the assertion fails. Playing such a game forces the player to focus on particular subformulas of the specification, and particular states of the model. This provides a useful discipline for helping the user to understand the structure of both, while keeping each step of the process simple enough to be easily comprehended.

Originally developed for simpler logics, Hintikka Games have been adapted to the area of temporal logic model checking [11–15] where they are called model checking games. Our contribution in this paper is to further adapt model checking games to the richer setting of temporal epistemic model checking. We extend previous work on model checking games in two directions. First, we deal with epistemic operators as well as branching-time temporal logic operators. Second, we deal with systems that are subject to *fairness* constraints, which express properties of infinite executions, such as "every action that is always enabled is eventually executed". These are more commonly considered in linear-time temporal logic, but have also been considered in a branching time setting. Fair CTL [16] extends CTL (Computational Tree Logic [17]) models with fairness constraints and thus is strictly more expressive. In this paper, we deal with a language combining Fair CTL and epistemic logic with observational semantics.

The structure of the paper is as follows. Section 2 gives the syntax and semantics of a fair branching time epistemic logic CTLK$_n$. In Section 3, we present a number of variants of the model checking game for this logic. We state the main theoretical results concerning the connection between the semantics and strategies in the game in Section 4. In Section 5 we briefly describe our implementation of the game in the model checker MCK [1]. We make some concluding remarks in Section 6.

2 Syntax and Semantics

We work with a logic CTLK$_n$ that combines CTL and the logic of knowledge and common knowledge for n agents. It will be interpreted with respect to structures representing fairness constraints. Let $Prop$ be a set of atomic propositions and $Ags = \{1, \ldots, n\}$ be a set of n agents. The syntax of CTLK$_n$ is given by the following grammar:

$$\phi \ ::= \ p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid EX\phi \mid E[\phi_1 U \phi_2] \mid EG\phi \mid K_i\phi \mid C_G\phi \tag{1}$$

where $p \in Prop$ and $i \in Ags$ and $G \in \mathcal{P}(Ags) \setminus \{\emptyset\}$.

The formula $K_i\phi$ says that agent i knows ϕ, and $C_G\phi$ says that ϕ is common knowledge to the group of agents G. The operators EX, EU and EG are from the logic CTL,

and refer to the branching structure of time. $EX\phi$ says that in some possible future, ϕ will hold at the next moment of time, $E[\phi_1 U \phi_2]$ says that in some possible future, ϕ_1 holds until ϕ_2 does, and $EG\phi$ says that in some possible future, ϕ holds at all future times. The logic CTL contains other operators, but these can be treated as defined. For example, $EF\phi = E[TrueU\phi]$, $AX\phi = \neg EX\neg\phi$, $AF\phi = \neg EG\neg\phi$, etc.

We use a semantics for CTLK$_n$ that is based on a variant of the *interpreted systems* model for the logic of knowledge [18]. Let S be a set, which we call the set of global states. A *run* over S is a function $r : \mathbf{N} \to S$. An *interpreted system* for n agents is a tuple $\mathcal{I} = (\mathcal{R}, \sim_1, \ldots, \sim_n, \pi)$, where $\mathcal{R}$ is a set of runs over S, each $\sim_i$ is an equivalence relation on S, and $\pi : S \to \mathcal{P}(Prop)$ is an interpretation function.

A *point* of $\mathcal{I}$ is a pair (r, m) where $r \in \mathcal{R}$ and $m \in \mathbf{N}$. We say that a run r' is *equivalent to a run r up to time* $m \in \mathbf{N}$ if $r'(k) = r(k)$ for $0 \leq k \leq m$. We define the semantics of CTLK$_n$ by means of a relation $\mathcal{I}, (r, m) \models \phi$, where $\mathcal{I}$ is an intepreted system, (r, m) is a point of $\mathcal{I}$ and ϕ is a formula. This relation is defined inductively as follows:

- $\mathcal{I}, (r, m) \models p$ if $p \in \pi(r(m))$,
- $\mathcal{I}, (r, m) \models \neg\phi$ if not $\mathcal{I}, (r, m) \models \phi$
- $\mathcal{I}, (r, m) \models \phi_1 \vee \phi_2$ if $\mathcal{I}, (r, m) \models \phi_1$ or $\mathcal{I}, (r, m) \models \phi_2$
- $\mathcal{I}, (r, m) \models EX\phi$ if there exists a run $r' \in \mathcal{R}$ equivalent to r up to time m such that $\mathcal{I}, (r', m + 1) \models \phi$
- $\mathcal{I}, (r, m) \models E[\phi_1 U \phi_2]$ if there exists a run $r' \in \mathcal{R}$ equivalent to r up to time m and $m' \geq m$ such that $\mathcal{I}, (r', m') \models \phi_2$, and $\mathcal{I}, (r', k) \models \phi_1$ for $m \leq k < m'$.
- $\mathcal{I}, (r, m) \models EG\phi$ if there exists a run $r' \in \mathcal{R}$ equivalent to r up to time m such that $\mathcal{I}, (r, k) \models \phi$ for all $k \geq m$
- $\mathcal{I}, (r, m) \models K_i\phi$ if for all points (r', m') of $\mathcal{I}$ such that $r(m) \sim_i r'(m')$ we have $\mathcal{I}, (r', m') \models \phi$
- $\mathcal{I}, (r, m) \models C_G\phi$ if for all sequences of points $(r, m) = (r_0, m_0), (r_1, m_1), \ldots (r_k, m_k)$ of $\mathcal{I}$, such that for each $j = 0 \ldots k - 1$, there exists $i \in G$ such that $r_j(m_j) \sim_i r_{j+1}(m_{j+1})$, we have $\mathcal{I}, (r_k, m_k) \models \phi$.

For the knowledge operators, this semantics is essentially the same as the usual interpreted systems semantics. For the temporal operators, it corresponds to a semantics for branching time known as the *bundle semantics* [19, 20].

While they give a clean and coherent semantics to the logic, interpreted systems are not suitable as inputs for a model checking program, since they are infinite structures. We therefore also work with an alternate semantic representation based on transition systems with epistemic indistinguishability relations and fairness constraints. A *(finite) system* is a tuple $M = (S, I, \to, \sim_1, \ldots, \sim_n, \pi, \alpha)$ where S is a (finite) set of global states, $I \subseteq S$ is the set of initial states, $\to \subseteq S \times S$ is a serial temporal transition relation, each $\sim_i$ is an equivalence relation representing epistemic accessibility for agent $i \in Ags$, $\pi : S \to \mathcal{P}(Prop)$ is a propositional interpretation, and α is a set of subsets of S, representing a (generalised Büchi) *fairness condition*. The fairness condition is used to semantically represent constraints such as 'whenever A occurs, B occurs at some later time,' or 'A occurs infinitely often,' that refer to infinite temporal evolutions of the system.

Given a system M over global states S, we may construct an interpreted system $\mathcal{I}(M) = (\mathcal{R}, \sim_1, \ldots, \sim_n, \pi)$ over global states S, as follows. The components $\sim_i$ and π

are identical to those in M. The set of runs is defined as follows. We say that a *fullpath* from a state s is an infinite sequence of states $s_0 s_1 ...$ such that $s_0 = s$ and $s_i \rightarrow s_{i+1}$ for all $i \geq 0$. We use $Path(s)$ to denote the set of all fullpaths from state s, and $FinPath(s)$ for the set of finite prefixes of fullpaths in $Path(s)$. The fairness condition is used to place an additional constraint on paths. A fullpath $s_0 s_1 \ldots$ is said to be *fair* if for all $Q \in \alpha$, there exists a state $s \in Q$ such that $s = s_i$ for infinitely many i. We write $Path^F(s)$ for the set of all fair fullpaths from s. A *run* of the system is a fair fullpath $s_0 s_1 \ldots$ with $s_0 \in I$. We define $\mathcal{R}$ to be the set of runs of M. A formula is said to *hold* in M, written $M \models \phi$, if $\mathcal{I}(M), (r, 0) \models \phi$ for all $r \in \mathcal{R}$.

We say that a state s is *fair* if it is the initial state of some fair fullpath, otherwise the state is *unfair*. We write $F(M)$ for the set of fair states of M. A state s is *reachable* if there exists a sequence $s_0 \rightarrow s_1 \rightarrow \ldots s_k = s$ where $s_0 \in I$. A state is fair and reachable iff it occurs in some run. We write $FR(M)$ for the set of fair and reachable states of M.

3 Game Semantics for CTLK$_n$

We now reformulate the semantics of CTLK$_n$ on structures M in the form of a Hintikka game. In such a game, there are two *players*, namely system (*Sys*) and user (*Usr*). If p is a player, we write $opp(p)$ for the opponent of p; thus, $opp(Sys) = Usr$ and $opp(Usr) = Sys$. In addition to the two players, we have two *roles*, verifier (**V**) and refuter (**R**). At each game state each player will be in some role, and the opponent will be in the opposite role. Intuitively, a player is in the verifier's (refuter's) role when she believes that the specific formula holds (resp., fails) in current state.

One of the main novelties in our game is that we need to deal with fairness and reachability. In principle, one could avoid this by first restricting systems to the fair reachable states, which would not change the semantics of validity. However, in practice, the existence of unfair reachable states is typically an error that the user will want to be able to diagnose. For this reason, we include unfair and unreachable states in the game, and introduce new propositional constants $Fair, Reach$ and $Init$ to represent that a state is fair (resp., reachable, initial).

Each pair (M, ϕ) consisting of a system M and a formula ϕ determines a game. We assume a fixed system M in what follows, and focus on the role of the formula ϕ (and its subformulas) in determining the states of this game. We call the states of the game *configurations*. There are three types of configuration:

1. Initial configuration: there is a unique initial configuration of the game, denoted $Usr{:}\phi$. Intuitively, this corresponds to the user taking the role of the verifier **V**, and claiming that the formula ϕ is valid in M.
2. Intermediate configurations: these have the form $p : \{(s_1, \phi_1), ..., (s_m, \phi_m)\}$ where $p \in \{Sys, Usr\}$ is a player and $\{(s_1, \phi_1), ..., (s_m, \phi_m)\}$ is a set of pairs, where each s_k is a state in S and each ϕ_k is either a formula or one of the constants $Fair, Reach, Init$. Intuitively, such a configuration corrresponds to the player p taking the role of the verifier **V**, and claiming that the assertion represented by all of the pairs (s_k, ϕ_k) is true of the system M. If ϕ_k is a formula then pair (s_k, ϕ_k) asserts that $M, s_k \models \phi_k$. If $\phi_k = Fair$ $(Reach, Init)$ then pair (s_k, ϕ_k) means s_k is a fair (resp. reachable, initial) state.

3. Final configuration. Configurations "p wins", where $p \in \{Sys, Usr\}$, are used to denote the completion of play. Intuitively, this means that player p has won the game and $opp(p)$ has lost the game.

Note that each initial and intermediate configuration has the form $p : x$, implying that player p is in the role of verifier and player $opp(p)$ is in the role of the refuter. We write intermediate representations $p : \{(s_1, \phi_1)\}$ with a singleton set of pairs simply as $p : (s_1, \phi_1)$.

At each round of the game, it is the turn of one of the players to make a move, depending on the configuration. Players' roles may exchange during the game. In Table 1, we list the rules of the game. Each rule is in the form

$$\frac{\mathit{CurrentConfiguration}}{\mathit{NextConfiguration}} \quad \mathit{Role} \quad \mathit{(Condition)} \tag{2}$$

representing that "if the game is in the *CurrentConfiguration* and the *Condition* holds, then it is the turn of the player in role *Role* to move, and one of the choices available to this player is to move the game into configuration *NextConfiguration*." In the rules, *Condition* and *Role* may not be present. If *Condition* is not present, the move can be made unconditionally. If *Role* is not present, there is only one possibility for the next configuration of the game, and the move can be made automatically without any player making a choice. We assume that $M = (S, I, \rightarrow, \sim_1, \ldots, \sim_n, \pi, \alpha)$ and that the fairness condition α has been presented as a set of propositional logic formulas $\{\chi_1, \ldots, \chi_N\}$, where each χ_k represents the set of states $\{s \mid M, s \models \chi_k\}$. The rule for the common knowledge operator uses the set $Kchain(G, s)$, where G is a set of agents and s is a state, defined to be the set of finite sequences of states $s = s_1 \ldots s_m$ such that for all $k = 1 \ldots m-1$ we have $s_k \sim_j s_{k+1}$ for some $j \in G$.

Note the use of the propositions *Fair* and *Reach* in the cases for the epistemic operators. For example, to refute a claim that $K_i\phi$ holds at a state s, we need not just a state $t \sim_i s$ where $\neg\phi$ holds, but we also need to assure that this state is in fact fair and reachable. We remark that in the rule for $E[\phi_1 U \phi_2]$, of the tuples (s_k, Fair) it would suffice to include only the last (s_m, Fair). However, inclusion of the earlier such tuples allows the refuter at the next move to select the earliest stage in the path at which a transition to an unfair state is made: this is more informative for the user in debugging the system.

In the rules for $EG\phi$ and *Fair*, we make use of a fact concerning generalized Büchi automata, viz., that there exists a fair fullpath from s (satisfying ϕ at every state) iff there exists a cyclic finite path (satisfying ϕ at every state) such that each of the fairness conditions are satisfied on the loop. More precisely, there exists a finite path $s = s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_m$ such that $s_m = s_i$ for some $i < m$, and for each fairness constraint χ_k, there exists an index l_k in the loop (i.e.. $i \leq l_k < m$), such that $M, s \models \chi_k$. In the case of $EG\phi$ we also need that $M, s_k \models \phi$ for all $k = 0 \ldots m-1$. Note that, whereas rules typically include pairs of the form (s, Fair) to represent that only fair states are used in the semantics, we do not need to do this in the case of the rules for *EG* and *Fair* since these rules already imply fairness of the states introduced.

Table 1. Game Semantics for Fair CTLK$_n$

Initial:	$\frac{Usr : \phi}{Usr : (s, \phi)}$ **R** $(s \in I)$
AP:	$\frac{p : (s, q)}{p \text{ wins}}$ $(q \in Prop \wedge q \in \pi(s))$
	$\frac{p : (s, q)}{opp(p) \text{ } wins}$ $(q \in Prop \wedge q \notin \pi(s))$
$\phi_1 \vee \phi_2$:	$\frac{p : (s, \phi_1 \vee \phi_2)}{p : (s, \phi_k)}$ **V** $(k \in \{1, 2\})$
$\neg\phi$:	$\frac{p : (s, \neg\phi)}{opp(p) : (s, \phi)}$
$EX\phi$:	$\frac{p : (s, EX\phi)}{p : \{(t, \phi), (t, Fair)\}}$ **V** $(s \to t)$
$E[\phi_1 U \phi_2]$:	$\frac{p : (s, E[\phi_1 U \phi_2])}{p : \{(s_1, \phi_1), ..., (s_{m-1}, \phi_1), (s_m, \phi_2), (s_1, Fair), ..., (s_m, Fair)\}}$ **V**
	$(s_1 ... s_m \in FinPath(s))$
$EG\phi$:	$\frac{p : (s, EG\phi)}{p : \{(s_1, \phi), ..., (s_{m-1}, \phi), (s_{l_1}, \chi_1), ..., (s_{l_N}, \chi_N)\}}$ **V**
	$(s_1 ... s_m \in FinPath(s),\ s_m = s_i,\ i \leq l_1, ..., l_N < m)$
$K_i\phi$:	$\frac{p : (s, K_i\phi)}{opp(p) : \{(t, \neg\phi), (t, Fair), (t, Reach)\}}$ **R** $(t \in S,\ s \sim_i t)$
$C_G\phi$:	$\frac{p : (s, C_G\phi)}{opp(p) : \{(s_m, \neg\phi), (s_1, Fair), (s_1, Reach), ..., (s_m, Fair), (s_m, Reach)\}}$ **R**
	$(s_1 ... s_m \in Kchain(G, s))$
$(s_1, \phi_1), ..., (s_m, \phi_m)$:	$\frac{p : \{(s_1, \phi_1), ..., (s_m, \phi_m)\}}{p : \{(s_k, \phi_k)\}}$ **R** $(1 \leq k \leq m)$
Fair:	$\frac{p : (s, Fair)}{p : \{(s_{l_1}, \chi_1), ..., (s_{l_N}, \chi_N))\}}$ **V**
	$(s_1 ... s_m \in FinPath(s),\ s_m = s_k,\ k \leq l_1, ..., l_N < m)$
Reach:	$\frac{p : (s, Reach)}{p : (s_1, Init)}$ **V** $(s_1 ... s_m \in FinPath(s_1), s_m = s)$
Init:	$\frac{p : (s, Init)}{p \text{ wins}}$ $(s \in I)$
	$\frac{p : (s, Init)}{opp(p) \text{ wins}}$ $(s \notin I)$

4 Main result

We can now state the main theoretical result of the paper, which is the equivalence of the model checking problem given above with the existence of a winning strategy in the associated game.

A *strategy* of a player is a function mapping the set of configurations in which it is the players' turn, to the set of possible next configurations according to the rules in Table 1.

A *play* of the game for (M, ϕ) according to a pair of strategies $(\sigma_{Usr}, \sigma_{Sys})$ for the user and system, respectively, is a sequence of configurations $C_0 C_1 \ldots$ such that C_0 is the initial configuration $Usr : \phi$, and at each step k, if it is player p's turn on configuration C_k, then there is a successor C_{k+1} in the play, and $C_{k+1} = \sigma_p(C_k)$. Note that it is no player's turn on a final configuration. Thus, a play is either infinite, or ends in a final configuration. In fact, we can show that all plays are finite.

Proposition 1 *If M is a finite state system and ϕ is any $CTLK_n$ formula, then all plays of the game for (M, ϕ) are finite.*

A *winning strategy* for player p is a strategy σ_p, such that for all strategies $\sigma_{opp(p)}$ for the opponent, all plays of the game according to $(\sigma_p, \sigma_{opp(p)})$ are finite and end in the configuration "p wins".

Theorem 1 *For all finite state systems M and formulas ϕ of $CTLK_n$, we have $M \not\models \phi$ iff there exists a winning strategy for Sys in the game for (M, ϕ).*

This theorem forms the basis for our game-based debugging approach. Suppose the user has written a specification ϕ for a system M, and this specification fails to hold in the system. If the user takes the role *Usr* in the game for (M, ϕ), and plays against a winning strategy for *Sys*, then the user will lose the game, however they play. In the process of playing the game, and trying different strategies, the user's attention will be drawn to particular states and subformulas. This may help the user to diagnose why their intuitions (concerning either the systems or the specification ϕ) are not in accordance with the facts.

While the game as defined above guarantees termination of any play of the game, it does so at the cost of including rules that involve the construction of rather large game states: viz., the rules for $E[\phi_1 U \phi_2]$, $EG\phi$, $C_G\phi$ and *Reach*, which involve construction of possibly lengthy paths. This creates a cognitive burden for the human player. It is possible to alleviate this in some cases by using more incremental versions of the rules, provided we weaken the correspondence between satisfaction and strategies.

Define the *recursive variant* of the game by replacing the rules for $E[\phi_1 U \phi_2]$, $C_G\phi$ and *Reach* by the rules in Table 2. The recursive variant admits non-terminating plays. E.g., if $s \rightarrow s$ then $Usr : (s, Reach), Usr : (s, Reach), \ldots$ is an infinite play. Thus, Theorem 1 no longer holds for this variant. However, we can recover the result with a slightly different notion of strategy.

Say that a *non-losing* strategy is a strategy σ_p, such that for all strategies $\sigma_{opp(p)}$ for the opponent, all *finite* plays of the game according to $(\sigma_p, \sigma_{opp(p)})$ end in the configuration "p wins".

Table 2. Recursive Game Semantics for Fair CTLK$_n$

$E[\phi_1 U \phi_2]$:	$\dfrac{p : (s, E[\phi_1 U \phi_2])}{p : \{(s, \phi_2 \vee (\phi_1 \wedge EX(E[\phi_1 U \phi_2])))\}}$
$C_G\phi$:	$\dfrac{p : (s, C_G\phi)}{p : \{(s, \bigwedge_{i \in G} K_i(\phi \wedge C_G\phi))\}}$
Reach:	$\dfrac{p : (s, Reach)}{p \text{ wins}} \quad (s \in I)$
	$\dfrac{p : (s, Reach)}{p : (t, Reach)} \quad \mathbf{V} \quad (t \rightarrow s)$

Theorem 2 *For all finite state systems M and formulas ϕ of CTLK$_n$, we have $M \not\models \phi$ iff there exists a non-losing strategy for Sys in the recursive variant of the game for (M, ϕ).*

The recursive version of the game is equally useful for debugging purposes: it enables the user to diagnose the error based on seeing that they cannot win the game, rather than based on seeing that they always lose the game. The key feature that they may explore the most relevant states and subformulas while playing is retained.

Further variants of the game could be constructed: the recursive variant retains the path constructing rules for $EG\phi$, but we could also make this more incremental by recording in the configuration, the state on which we claim the path under construction loops, as well as the fairness constraints already claimed to have been satisfied on previous states in the loop. We could furthermore revise the recursive variant to make it terminating by adding to configurations sufficient information to detect when a play revisits a previously visited configuration (at the cost of admitting very large configurations). We leave the development of such variants to the reader.

5 Implementation

We have implemented the game to provide a debugging facility for the epistemic model checker MCK [1]. MCK provides the ability to model check specifications in both linear and branching time, using a variety of model checking algorithms, depending on the formula in question and a choice of semantics for knowledge: this can be the observational semantics (as in the present paper), a semantics in which local states consist of the current observation plus a clock value, and a synchrononous perfect recall semantics.

Older versions of MCK have been based on algorithms that use symbolic techniques (binary decision diagrams) [21] to do model checking. The implementation of the game adds to MCK a new *explicit-state* model checking facility: this is an algorithm that performs model checking by means of an explicit construction of the reachable states

of the system. The approach is essentially an extension of standard explicit-state algorithms for CTL [21] to include epistemic operators. Explicit-state model checking is only feasible for systems with a small state space, but its benefit for our present purposes is that the explicit-state model checking algorithm can be extended to construct during model checking a winning/non-losing strategy for the system. This strategy is then used by the system to play the game against the user in case the debugging game is invoked. An additional benefit of explicit construction of the (reachable) state space is that this allows the state space to be displayed using a graph visualization tool. Our implementation allows the Graphviz tools to be used for this purpose.

The implementation is based on a version of the game that employs the recursive rules of Table 2 only when it is the turn of the user; at system moves the rules of Table 1 are used. In case unfair reachable states exist in the system, the user is offered the choice of playing the game on the system as given (for diagnosing the reason for such states) or a variant of the system in which such states are removed (in case the existence of such states is what the user actually intended).

6 Conclusion and Future Work

We have presented a model checking game for a fair branching time epistemic logic and its implementation in the model checker MCK. Playing the game can help a user to diagnose errors in MCK models.

In future work, we intend to strengthen our tool in two directions: The first is to enable the system to play the game using symbolic model checking algorithms: this will allow the game to be played on models with much larger statespaces. The second is to make the logic supported by the game more expressive: we are presently developing an extension of the game to include μ-calculus operators (these are already supported in the symbolic model checking algorithms in MCK). This will enable notions such as eventual common knowledge [18] to be handled.

Acknowledgements: An initial implementation of the game and explicit state model checking facility for MCK was done by Jeremy Lee; the current implementation is a significant revision by the authors. Cheng Luo has conducted some maintenance on the system.

References

1. P. Gammie and R. van der Meyden. MCK: Model Checking the Logic of Knowledge, 16th International conference on Computer Aided Verification, CAV 2004, pp. 479- 483.
2. A. Lomuscio, and F. Raimondi. MCMAS: A Model Checker for Multi-agent Systems. In H. Hermanns and J. Palsberg, editors, TACAS, LNCS 3920, pages 450-454. Springer-Verlag, 2006.
3. H. P. van Ditmarsch, J. Ruan, and R. Verbrugge. Sum and Product in Dynamic Epistemic Logic. Journal of Logic and Computation 2008 18(4):563-588.
4. W. Nabialek, A. Niewiadomski, W. Penczek, A. Polrola, and M. Szreter. VerICS 2004: A model checker for real time and multi-agent systems. In Proceedings of the International Workshop on Concurrency, Specification and Programming (CS&P'04), Informatik-Berichte 170, pages 88-99, Humboldt University, 2004.

5. R. van der Meyden and K. Su. Symbolic Model Checking the Knowledge of the Dinning Cryptographers. In the Proceedings of 17th IEEE Computer Security Foundations Workshop (CSFW 2004). IEEE Computer Society, 2004.
6. K. Baukus and R. van der Meyden. A knowledge based analysis of cache coherence. International Conference on Formal Engineering Methods, Seattle, Nov 2004, LNCS No. 3308, pp. 99-114.
7. J.-J. Ch. Meyer and W. van der Hoek. Epistemic Logic for AI and Computer Science, Cambridge University Press, 1995.
8. Y. Shoham and K. Leyton-Brown. Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press, 2009.
9. E. Clark, O. Grumberg, S. Jha, Y. Lu, H. Veith. Counterexample-Guided Abstraction Refinement for Symbolic Model Checking, Journal of the ACM, 50 (5), pp. 752-794.
10. J. Hintikka. Logic, Language-Games and Information: Kantian Themes in the Philosophy of Logic. Clarendon Press, Oxford, 1973.
11. C. Stirling and D. Walker. Local Model Checking in the Modal mu-Calculus. Theoretical Computer Science 89, pages 161-177, 1991.
12. C. Stirling. Local Model Checking Games. In Proceedings of the 6th International Conference on Concurrency Theory, CONCUR '95. Lecture Notes in Computer Science 962, pages 1-11, 1995.
13. M. Lange and C. Stirling. Model Checking Games for Branching Time Logics. Journal of Logic Computation 12(4), pages 623-939, 2002.
14. D. Fischer, E. Gradel, and L. Kaiser. Model Checking Games for the Quantitive μ-calculus. In Dans Proceedings of the 25th Annual Symposium on the Theoretical Aspects of Computer Science (STACS 2008), pages 301-312.
15. H. Fecher, M. Huth, N. Piterman and D. Wagner. Hintikka Games for PCTL on Labeled Markov Chains. 2008.
16. E.M. Clark, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications. ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2, April 1986, Pages 244-263.
17. E.M. Clark and E.A. Emerson. Synthesis of Synchronization Skeletons for Branching Time Temporal Logic. In Logic of Programs: Workshop, Lecture notes in Computer Science 131, 1981. Springer.
18. R. Fagin, J.Y. Halpern, Y. Moses and M.Y. Vardi. Reasoning about Knowledge. The MIT Press, 1995.
19. J. Burgess. Logic and Time. Journal of Symbolic Logic, 44, pages 556-582, 1979.
20. R. van der Meyden and K. Wong. Complete Axiomatizations for Reasoning about Knowledge and Branching Time, Studia Logica Vol 75, Oct 2003, pp. 93-123.
21. E.M. Clark, O. Grumberg, and D.A. Peled. Model Checking. Cambridge MA: MIT Press, 1999.