Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Synthesizing Strategies for Epistemic Goals by Epistemic Model Checking: An Application to Pursuit Evasion Games*

Xiaowei Huang, Ron van der Meyden
School of Computer Science and Engineering,
University of New South Wales, Australia

Abstract

The paper identifies a special case in which the complex problem of synthesis from specifications in temporal-epistemic logic can be reduced to the simpler problem of model checking such specifications. An application is given of strategy synthesis in pursuit-evasion games, where one or more pursuers with incomplete information aim to discover the existence of an evader. Experimental results are provided to evaluate the feasibility of the approach.

Introduction

Epistemic model checking is a formal method in which one automatically verifies whether a system in which agents have incomplete information satisfies a specification in the logic of knowledge and time. In this paper, we show that in some circumstances, epistemic model checking can provide not just a technology with which to verify agent behaviours, but also a technology for *synthesizing* such behaviours. In synthesis, one starts with a specification, as well as the context in which it is to be satisfied, and one is interested in finding a strategy, i.e., a pattern of behavior of the agents, that makes the specification true.

The synthesis problem for temporal-epistemic specifications is quite hard: double-exponential time for a single agent, and generally undecidable for multiple agents (van der Meyden and Vardi 1998). The known synthesis algorithms generally require complex automaton theoretic constructions, and as yet, there exist no implementations of these algorithms. The special case of synthesis we consider in this paper, however, turns out to be significantly simpler. We show that techniques for counter-example construction already implemented in the epistemic model checker MCK suffice for the synthesis with respect to certain types of specification in a special class of environments.

We illustrate the result by applying it in the setting of *Pursuit-Evasion games*, which are a type of multi-player game in which one or more pursuers have the objective of identifying the presence of one or more evaders, and of capturing them. Solutions to pursuit-evasion problems have multiple applications, including air and naval combat, ship navigation, (Isaacs 1965), automatic car collision avoidance systems (Lachner, Breitner, and Pesch 2000), air traffic control, and unmanned aerial vehicle control (Vidal et al. 2002). It has been argued by (Huang, Maupin, and van der Meyden 2011) that in settings where agents have incomplete information concerning the game state, *temporal-epistemic* goals, which concern how the *knowledge* of agents evolves over time, become relevant to the analysis of pursuit-evasion scenarios. We show that our result can be used to solve a strategy synthesis problem arising from one such goal, concerned with *region clearing*, i.e., determining whether an evader is present. To evaluate the approach, we carry out some experiments using the technique in which the game is played on a variety of types of graph: a class of random graphs and a class of graphs in the form of a Manhattan grid with holes.

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

Epistemic Model Checking

Model checking (Clarke, Grumberg, and Peled 1999) is a formal verification method, based on automated computations that determine whether a specification holds in a model representing the system to be verified. Model checking is traditionally based on specifications expressed in temporal logic, but a number of model checkers, e.g., MCK (Gammie and van der Meyden 2004), MCMAS (Lomuscio, Qu, and Raimondi 2009), VerICS(Kacprzak et al. 2008), DEMO (Eijck 2004) and MCTK (Su, Sattar, and Luo 2007) now also support epistemic operators, which deal with properties of agents' knowledge (Fagin et al. 1995).

In this paper we apply the epistemic model checker MCK. MCK has a rich range of modal operators. The fragment we need in this paper is based on the following grammar:

$$\phi \;=\; p \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid X\phi \mid G\phi \mid K_i\phi$$

where p is an atomic proposition from the set Prop, and i is an element of the set of agents Agt. Intuitively, $X\phi$ means that ϕ holds at the next moment of time, $G\phi$ means that ϕ holds now and at all future times, and $K_i\phi$ means that agent i knows that ϕ is true. Another useful definable operator is $F\phi = \neg G\neg\phi$ (ϕ holds at some future time).

The logic can be given a semantics using *interpreted systems* (Fagin et al. 1995), which are pairs $\mathcal{I} = (\mathcal{R}, \pi)$, where $\mathcal{R}$ is a set of runs, describing how the system evolves over time,

and $\pi : \mathcal{R} \times \mathbf{N} \to \mathcal{P}(Prop)$ is an interpretation function, that indicates which atomic propositions are true at each *point* of the system, represented by a pair (r, n) where $r \in \mathcal{R}$ is a run and $n \in \mathbf{N}$ is a natural number representing a time. More concretely, each run $r \in \mathcal{R}$ is represented as a function $r : \mathbf{N} \to S \times \Pi_{i \in Agt} L_i$, where S is a set, representing the possible states of the environment in which the agents operate, and where each L_i is a set, representing the possible *local states* of agent i. Given a run r, agent i and time m, we write $r_i(m)$ for the $i+1$-st component (in L_i) of $r(m)$, and $r_e(m)$ for the first component (in S).

The semantics of the language is given by 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, defined inductively as follows (the obvious cases for boolean operators are omitted):

- $\mathcal{I}, (r, m) \models p$ if $p \in \pi(r, m)$,
- $\mathcal{I}, (r, m) \models X\phi$ if $\mathcal{I}, (r, m + 1) \models \phi$,
- $\mathcal{I}, (r, m) \models G\phi$ if for all $m' \geq m$ we have $\mathcal{I}, (r, m') \models \phi$,
- $\mathcal{I}, (r, m) \models K_i\phi$ if for all points (r', m') of $\mathcal{I}$ such that $r_i(m) = r'_i(m')$ we have $\mathcal{I}, (r', m') \models \phi$.

We write $\mathcal{I} \models \phi$ if $\mathcal{I}, (r, 0) \models \phi$ for all runs r of $\mathcal{I}$, i.e., the formula holds at time zero in all runs.

In model checking, the system to be verified is generally given in the form of (a program representing) a finite state transition system. We will view this system as composed of two parts: an environment E and a joint *protocol* P for the agents. Environments describe how the state of the system changes in response to agent actions, and protocols describe how the agents choose their actions in response to observations of the environment.

Environments have the form $E = (S, S_0, \tau, \{O_i\}_{i \in Agt}, \pi_E)$, where S is a set of states, S_0 is the set of possible initial states, $\tau : S \times Act \to \mathcal{P}(S)$ represents how the state of the system changes (nondeterministically) in response to actions performed by the agents, component $O_i : S \to O$ is a function that represents the observation made by agent i in each state (here O is the set of all possible observations), and $\pi_E : S \to \mathcal{P}(Prop)$ interprets atomic propositions at the states. More concretely, the set $Act = \Pi_{i \in Agt} Act_i$ is a set of tuples, with each set Act_i representing the actions performed by agent i. We write $s \xrightarrow{a} t$ if $t \in \tau(s, a)$, for $a \in Act$. We write $E, s \models p$ if $p \in \pi_E(s)$, and extend this to Boolean combinations of atomic propositions in the usual way.

A protocol for agent i is a mapping $P_i : O^* \to \mathcal{P}(Act_i)$, mapping each sequence of observations that can be made by the agent to a set of actions that the agent may choose to perform after making that sequence of observations. A joint protocol is a collection $P = \{P_i\}_{i \in Agt}$ where each P_i is a protocol for agent i.

Given an environment E, and a joint protocol P, we may construct an interpreted system that represents all the possible runs of E when agents choose their actions according to the protocol P. In doing so, we also need to define how the local states of agents are determined, which in turn impacts what the agents know. MCK provides several different ways to define the local states. We work in this paper with the *synchronous perfect recall* assignment of local states, in which the local state is the sequence of all observations that has been made by the agent. We denote the system constructed from E and P under this assignment by $\mathcal{I}^{spr}(E, P) = (\mathcal{R}, \pi)$, defined by taking each $L_i = O^*$ and taking $\mathcal{R}$ to be the set of runs $r : \mathbf{N} \to S \times \Pi_{i \in Agt} L_i$ such that

1. $r_e(0) \in S_0$,
2. for all $n \in \mathbf{N}$, we have $r_i(n) = O_i(r_e(0)) \dots O_i(r_e(n))$, and
3. for all $n \in \mathbf{N}$, there exists a joint action a with $a_i \in P_i(r_i(n))$ for each agent i, and $r_e(n) \xrightarrow{a} r_e(n + 1)$.

The assignment $\pi : \mathcal{R} \times \mathbf{N} \to \mathcal{P}(Prop)$ is obtained from the assignment π_E of E by defining $\pi(r, n) = \pi_E(r_e(n))$. Runs as defined above do not record the action chosen at each step: to avoid complicating the definitions we assume that these are recorded in the environment state, so that there exists a function $f : S \to Act$ such that if $s \xrightarrow{a} t$ then $f(t) = a$.

The model checking problem solved by MCK is the following: given an environment E, a joint protocol P and a formula ϕ, determine if $\mathcal{I}^{spr}(E, P) \models \phi$. For this to be feasible, we need some further assumptions: first, the set of states of E needs to be finite, the protocol P also needs to be represented in a finite state way, and some restrictions on the formula are required. For finite state representation of the protocol, we may assume that the environment has been defined so that the protocol depends only on the final observation, i.e. for each agent i there exists a function $f_i : O \to \mathcal{P}(Act_i)$ such that $P_i(o_0 \dots o_n) = f_i(o_n)$.

We concentrate in this paper on two types of formula for which MCK supports model checking with respect to the synchronous perfect recall semantics: formulas $G\phi$ where ϕ is a formula in which the only modal operator is K_i for a fixed agent i, and formulas of the form $X^n\phi$, with the same restriction on ϕ. MCK model checks formulas of the form $G\phi$ by computing a "belief space" transition system in which states correspond to sets of states consistent with the agent's observations, and represents these sets as binary decision diagrams (Bryant 1986). (This algorithm is closely related to approaches that have been used in planning under partial observability using BDD representations (Bertoli et al. 2006).) For formulas of the form $X^n\phi$, all knowledge states possible at time n are represented in a single binary decision diagram, constructed using an algorithm described in (van der Meyden and Su 2004).

For both types of formula, MCK is able to return a *counter-example* when the formula does not hold. The counter-example is in the form of a run prefix $\rho = r(0) \dots r(k)$ such that for all runs r extending ρ, we have $\mathcal{I}^{spr}(M, P), (r, k) \models \neg\phi$. (In case of the formula $X^n\phi$ we have $k = n$.) We make critical use of this capability in this paper.

A problem related to model checking is the *synthesis* problem: given an environment E and a formula ϕ, find a protocol P such that $\mathcal{I}^{spr}(E, P) \models \phi$ (or determine that no such protocol exists). This is a significantly more complicated problem than model checking: in general, it is undecidable, and double exponential time complete for environments with a single agent (van der Meyden and Vardi 1998). One of our contributions in this paper is to identify a case where this more complicated problem can be reduced to the

simpler problem of model checking. (In both cases, there may be exponentially many possible states of knowledge as a function of the number of states of the underlying system, which in turn may be exponential in the number of variables used in its programmatic representation, so the algorithms are dealing with an inherently complex problem and are not guaranteed to work efficiently. However, they have been found to be useful in a range of interesting problems.)

A Result on Epistemic Synthesis

We now present a general result on epistemic synthesis, that shows how the counter-examples produced by an epistemic model checker can, in some circumstances, be used to solve a synthesis problem. Formulation of the result requires a number of technical definitions. In what follows, E is an environment with a single agent i.

Say that a proposition α is *bi-stable* in E if for states s, s' such that $s \xrightarrow{a} s'$ for some a, we have $E, s \models \alpha$ iff $E, s' \models \alpha$. Thus, if α is true, it remains true for the rest of time, and similarly, if α is false, then it remains so.

Say that agent i has a *unique initial observation under condition* α in E if for all initial states s, s' of E with $E, s \models \alpha$ and $E, s' \models \alpha$ we have $O_i(s) = O_i(s')$.

Say that *agent i's observations are deterministic under condition* α if for all states s_1, s_2, t_1, t_2 of E and actions a of agent i, if $E, s_1 \models \alpha$, and $E, s_2 \models \alpha$ and $O_i(s_1) = O_i(s_2)$ and $s_1 \xrightarrow{a} t_1$ and $s_2 \xrightarrow{a} t_2$ then $O_i(t_1) = O_i(t_2)$. Intuitively, this says that under condition α, the agent's next observation can be uniquely determined from its current observation and the action that it is performing.

Say that *agent i's observations are action-recording* if for all states s_1, s_2, t_1, t_2 and actions a, b of agent i, if $O_i(s_1) = O_i(s_2)$ and $s_1 \xrightarrow{a} t_1$ and $s_2 \xrightarrow{b} t_2$ and $O_i(t_1) = O_i(t_2)$ then $a = b$. Intuitively, this means that the most recent action performed by the agent can be deduced from its observations.

Define the protocol P_{rand} to be the protocol that allows the agent to choose any of its actions at any time. That is, for all sequences σ of observations, $P_{rand}(\sigma) = Act_i$, the set of all actions of the agent.

The formula $F((K_i\alpha) \vee K_i\neg\alpha)$ states that the agent eventually learns whether the proposition α is true or false. The following result states that to synthesize a protocol for this specification in environments satisfying the above conditions, it suffices to find a counter-example to model checking another specification with respect to the specific protocol P_{rand}. This reduces the more complex synthesis problem to a simpler model checking problem.

Theorem 1 *Let the environment E have the single agent i. Suppose that agent i has a* unique initial observation *in E. Let α be a bistable proposition in E, nontrivial in the sense that there exist initial states satisfying α, and suppose that agent i's observations are action-recording and deterministic under condition α. Then the following are equivalent:*

- *there exists a protocol P for agent i such that $\mathcal{I}^{\mathrm{spr}}(E, P) \models F((K_i\alpha) \vee K_i\neg\alpha)$*
- *not $\mathcal{I}^{\mathrm{spr}}(E, P_{rand}) \models G(\neg K_i\alpha)$.*

Proof: Assume first that not $\mathcal{I}^{\mathrm{spr}}(E, P_{rand}) \models G(\neg K_i\alpha)$. We show that there exists a protocol P for agent i such that $\mathcal{I}^{\mathrm{spr}}(E, P) \models F((K_i\alpha) \vee K_i\neg\alpha)$. By assumption, there exists a point (r^*, n) of $\mathcal{I}^{\mathrm{spr}}(E, P_{rand})$ such that $\mathcal{I}^{\mathrm{spr}}(E, P_{rand}), (r^*, n) \models K_i\alpha$. By validity of $(K_i\alpha) \Rightarrow \alpha$ and bistability, we have $E, r^*(k) \models \alpha$ for all k. In particular, $E, r^*(0) \models \alpha$. Define the protocol P to be the (deterministic) protocol with $P(o_0 \ldots o_k) = \{a_k\}$ where a_k is the action taken by r in run r^* at time k. We claim that $\mathcal{I}^{\mathrm{spr}}(E, P) \models F((K_i\alpha) \vee K_i\neg\alpha)$. For, let r be any run of $\mathcal{I}^{\mathrm{spr}}(E, P)$. We consider two cases: $E, r(0) \models \alpha$ and $E, r(0) \models \neg\alpha$.

Assume first that $E, r(0) \models \alpha$. We show that in this case $\mathcal{I}^{\mathrm{spr}}(E, P), (r, n) \models K_i\alpha$. We first claim that $r_i(k) = r_i^*(k)$ for all $k \leq n$. This can be seen by induction. The base case follows from the assumption that agent i has a unique initial observation under condition α, since both $E, r(0) \models \alpha$ and $E, r^*(0) \models \alpha$, so we have $r_i(0) = O_i(r(0)) = O_i(r^*(0)) = r_i^*(0)$. Inductively, if, for $k < n$, we have $r_i(k) = r_i^*(k)$, then since, in the perfect recall semantics, each of these local states is a sequence of observations, we have $O_i(r(k)) = O_i(r^*(k))$. Moreover, by bistability we have that $E, r(k) \models \alpha$ and $E, r^*(k) \models \alpha$. Since r is a run of P, the same action a is taken in the runs r and r^* at time k. Thus $r(k) \xrightarrow{a} r(k+1)$ and $r^*(k) \xrightarrow{a} r^*(k+1)$. Since agent i's observations are deterministic under condition α we obtain that $O_i(r(k+1)) = O_i(r^*(k+1))$. Hence $r_i(k+1) = r_i(k) \cdot O_i(r(k+1)) = r_i^*(k) \cdot O_i(r^*(k+1)) = r_i^*(k+1)$. This completes the proof of the claim.

Suppose now that r' is a run of $\mathcal{I}^{\mathrm{spr}}(E, P)$ with $(r, n) \sim_i (r', n)$. Then r' is also a run of $\mathcal{I}^{\mathrm{spr}}(E, P_{rand})$ and $r'_i(n) = r_i(n) = r_i^*(n)$. Since we have $\mathcal{I}^{\mathrm{spr}}(E, P_{rand}), (r^*, n) \models K_i\alpha$, we must have $E, r'_i(n) \models \alpha$. This shows that $\mathcal{I}^{\mathrm{spr}}(E, P), (r, n) \models K_i\alpha$.

Alternatively, suppose that $E, r(0) \models \neg\alpha$. By bistability, we have $E, r(n) \models \neg\alpha$. Since r is also a run of $\mathcal{I}^{\mathrm{spr}}(E, P_{rand})$ and $\mathcal{I}^{\mathrm{spr}}(E, P_{rand}), (r^*, n) \models K_i\alpha$, we cannot have $r_i(n) = r_i^*(n)$. But we showed above that for all runs r' of $\mathcal{I}^{\mathrm{spr}}(E, P)$ with $E, r'(0) \models \alpha$, we have $r'_i(n) = r_i^*(n)$. Thus, for all runs r' of $\mathcal{I}^{\mathrm{spr}}(E, P)$ with $r'_i(n) = r_i(n)$, we must have $E, r'(0) \models \neg\alpha$, and indeed $E, r'(n) \models \neg\alpha$ by bistability. This shows that $\mathcal{I}^{\mathrm{spr}}(E, P), (r, n) \models K_i\neg\alpha$.

Thus, in either case we have $\mathcal{I}^{\mathrm{spr}}(E, P), (r, n) \models (K_i\alpha) \vee K_i\neg\alpha$ for all runs r of $\mathcal{I}^{\mathrm{spr}}(E, P)$, so $\mathcal{I}^{\mathrm{spr}}(E, P) \models F((K_i\alpha) \vee K_i\neg\alpha)$.

Conversely, assume that $\mathcal{I}^{\mathrm{spr}}(E, P_{rand}) \models G(\alpha \Rightarrow \neg K_i\alpha)$. We show that there does not exists a protocol P for agent i such that $\mathcal{I}^{\mathrm{spr}}(E, P) \models F((K_i\alpha) \vee K_i\neg\alpha)$. By way of contradiction, suppose that there is such a protocol. Applying nontriviality of α, let r be any run of $\mathcal{I}^{\mathrm{spr}}(E, P)$ with $E, r(0) \models \alpha$. Then for some n we have $\mathcal{I}^{\mathrm{spr}}(E, P), (r, n) \models (K_i\alpha) \vee K_i\neg\alpha$. By bistability, we have $E, r(n) \models \alpha$, so we cannot have the latter disjunct, and we in fact have $\mathcal{I}^{\mathrm{spr}}(E, P), (r, n) \models K_i\alpha$. Note that r is also a run of $\mathcal{I}^{\mathrm{spr}}(E, P_{rand})$. Consider any other run r' of $\mathcal{I}^{\mathrm{spr}}(E, P_{rand})$ with $r'_i(n) = r_i(n)$. Then the sequence of observations of agent i to time n are the same in the two runs r and r', as are the local states $r_i(k)$ and $r'_i(k)$ for $k \leq n$. Since agent i's observations are action-recording, it follows that the same actions are performed in r and r'

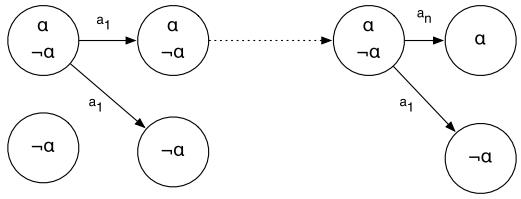

Figure 1: Structure of $\mathcal{I}^{spr}(E, P_{rand})$

to time k, so that r' satisfies all the conditions for being a run of protocol P, up to at least time n. By varying just the actions after time n, we may construct a run r'' of P that is identical to r' up to time n. In particular, we have $r''_i(n) = r_i(n)$, so from $\mathcal{I}^{\mathrm{spr}}(E, P), (r, n) \models K_i\alpha$ we obtain that $\mathcal{I}^{\mathrm{spr}}(E, P), (r'', n) \models \alpha$, i.e., $E, r''(n) \models \alpha$. Since $r''(n) = r'(n)$, we in fact have $E, r'(n) \models \alpha$. This shows that $\mathcal{I}^{\mathrm{spr}}(E, P_{rand}), (r, n) \models K_i\alpha$. But this is a contradiction with the assumption that $\mathcal{I}^{\mathrm{spr}}(E, P_{rand}) \models G(\neg K_i\alpha)$. ■

Figure 1 illustrates the structure of the knowledge states in the system $\mathcal{I}^{\mathrm{spr}}(E, P_{rand})$ that the theorem exploits. A counter-example to the specification

$$G(\neg K_i\alpha) \tag{1}$$

of the form that MCK returns, when this formula is not valid in the system $\mathcal{I}^{spr}(E, P_{rand})$, would have the form of a sequence of states $r_e(0) \dots r_e(n)$ from a run r where $\mathcal{I}^{spr}(E, P_{rand})(r, k) \models \neg K_i\alpha \wedge \neg K_i\neg\alpha$ for $k < n$, and $\mathcal{I}^{spr}(E, P_{rand})(r, n) \models K_i\alpha$. Performing the action at time k may result in several different observations, but at most one leaves the agent uncertain as to α, the rest give the agent the information that $\neg\alpha$. The protocol P synthesized simply follows the actions performed in the run r up to time n (and can then behave arbitrarily.) Thus, the protocol P is easily constructed from the counter-example returned by MCK. Indeed, the same would apply when the formula $X^n(\neg K_i\alpha)$ is not valid: again, the counter-example returned by MCK can be used to construct the protocol P. We therefore have an alternate way to obtain P: check

$$X^n(\neg K_i\alpha) \tag{2}$$

for increasing values of n until this formula is found to be invalid. This enables us to construct P in a way that guarantees it uses a minimal length sequence of actions: the model checking proves that all shorter protocols fail to solve the problem. So we get not just a solution to the problem, but also a solution that is *optimal* in the time take to make a decision about the truth of α. We note that, in the worst case, the solution may lie at depth of the order of $2^{|S|}$ steps, where $|S|$ is the number of states of the environment, since potentially we need to traverse that many different states of knowledge.

Pursuit-Evasion Games

To illustrate the application of the result of the previous section, we focus for the remainder of the paper on *pursuit-evasion* games, a type of multi-player game with incomplete information. The agents in these games consist of a number of *pursuers*, who seek to discover and intercept a number of *evaders*. We focus on the discovery part of the game.

We define a general framework for these games in this section. Specifically, the pursuit-evasion games we consider are discrete games in which the agents have limited visibility, but have complete knowledge of the map, which is represented as a graph. The players are able to move to an adjacent node in each step.

More precisely, each game is played on a map $\mathcal{M} = (V, E_m, E_v)$ where V is a set of vertices (or positions), $E_m \subseteq V \times V$ is a set of adjacency edges, along which players can move, and $E_v \subseteq V \times V$ is a set of edges indicating visibility: a pursuer at position u can see the evader at position v just in case $(u, v) \in E_v$. All our examples use undirected graphs, so that both relations E_m, E_v are symmetric. We also assume that E_m is reflexive, so that staying at the same location is a valid move.

A *game* is given by a tuple $\mathcal{G} = (\mathcal{M}, A_P, A_E, I, sched, v)$, where $\mathcal{M}$ is a map, A_P is the set of pursuers, A_E is the set of evaders, I is a set of initial game states on M, and *sched* is a scheduler, and $v \in \mathbf{N} \cup \{\infty\}$ gives a maximal velocity for the evaders. We confine ourselves to games with a single evader, so that $A_E = \{e\}$ – it can be shown that this is without loss of generality for the problem we consider.

A scheduler is represented by a tuple $sched = (\Sigma, \sigma_0, \epsilon, \mu)$ where Σ is a set of scheduler states, $\sigma_0 \in \Sigma$ is the initial state of the sheduler, $\epsilon : \Sigma \to \mathcal{P}(A_P \cup A_E)$ is a function representing the set of agents enabled to move by each scheduler state, and $\mu : \Sigma \to \Sigma$ captures how the scheduler state is updated at each step of the game. (Since this is independent of players' actions, we have just one fixed schedule that is common to all runs.) We consider two types of schedulers. The *synchronous* scheduler enables all players at all moves, and a *turn-based* scheduler alternates scheduling of all pursuers with scheduling of all evaders. We discuss the winning condition of the game below.

A game state for the game played on a map $\mathcal{M} = (V, E_m, E_v)$ by pursuers A_P and evaders A_E consists of a tuple $(posn, \sigma)$, where $posn : A_P \cup A_E \to V$ is a function giving the location of each pursuer and evader, and $\sigma \in \Sigma$ is a scheduler state. For initial states in I we require that $\sigma = \sigma_0$.

Given a game $\mathcal{G}$, we obtain an environment $E_{\mathcal{G}}$ with states consisting of the set of game states of $\mathcal{G}$, and initial states equal to I. To enable application of Theorem 1, the environment has a single agent p, representing the set of pursuers A_P aggregated into one, and we model the evader as part of the environment rather than as a separate agent. More precisely, we define the observation $O_p(s)$ of the pursuers p at the state $s = (posn, \sigma)$ to be the tuple $(posn^*, \sigma)$, where $posn^*(b) = posn(b)$ if player b is a pursuer, and for the case where b is equal to the evader e, we have $posn^*(e) = posn(e)$ if the evader is visible from some pursuer's position, i,.e., $(posn(a), posn(e)) \in E_v$ for some $a \in A_P$, and $posn^*(e) = \perp$ otherwise, indicating an unknown position.

The set of actions of the pursuers is defined to be the set of functions $m : A_P \to V$, with $m(b)$ representing the vertex to which pursuer b wishes to move. The desired move is effective only if the pursuer is scheduled and vertex $m(b)$ is adjacent to b's current position. More precisely, we have

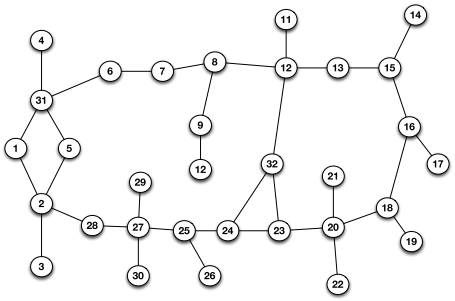

Figure 2: Gates

$(posn', \sigma') \in \tau((posn, \sigma), m)$ when the following hold. First, the scheduler state is correctly updated, i.e., $\sigma' = \mu(\sigma)$. Next, for all unscheduled agents $b \notin \epsilon(\sigma)$ and for all pursuers b requesting an illegal move $m(b)$, there is no change of position, i.e., we have $posn'(b) = posn(b)$. A move $m(b)$ is illegal just in case $(posn(b), m(b)) \notin E_m$. Next, pursuers requesting a legal move when they are enabled are allowed to make that move, i.e., if $b \in \epsilon(\sigma)$ and $(posn(b), m(b)) \in E_m$ then $posn'(b) = m(b)$. Finally, the evader, if scheduled, moves consistently with its maximal velocity v, but in such a way as to avoid passing through pursuer positions. More precisely, if $e \in \epsilon(\sigma)$ and the evader has maximum velocity v, then there is a sequence of positions $p_0, \ldots, p_k$ with $k \leq v$ such that $(p_j, p_{j+1}) \in E_m$ for $j = 0..k-1$, no p_j is equal to $posn(b)$ for any pursuer b (unless already $posn(e) = posn(b)$ for some pursuer b), and $posn'(e) = p_k$. In all other cases we have $posn'(e) = posn(e)$.

The only propositions of interest in our application will be the propositions "$pos_e \in X$" where $X \subseteq V$ is a set of game positions. The assignment π of the environment will make this true at a state $(posn, \sigma)$ just in case $posn(e) \in X$.

In particular, we will consider just the case that X is a connected component of the graph (V, E_m). It is easily seen that this makes the proposition $pos_e \in X$ bistable in the environment $E_{\mathcal{G}}$. Moreover, it can also be seen from the definition of the state transitions and the observation function that agent p's observations are action-recording, and that if all pursuers are located in the component X in all states, then agent p's observations are deterministic under condition $pos_e \notin X$. To obtain that agent p has a unique initial observation under condition $pos_e \notin X$, it suffices for each pursuer to start at a fixed position in all initial states.

This then means that Theorem 1 applies with $\alpha = pos_e \notin X$. Consequently, we may solve the problem of synthesizing a (joint) pursuer strategy satisfying the formula $F(K_p(pos_e \notin X) \vee K_p(pos_e \in X))$ by model checking the formula $G(\neg K_p(pos_e \notin X))$. Note that the formula $F(K_p(pos_e \notin X) \vee K_p(pos_e \in X))$ expresses that the pursuers are guaranteed to eventually learn whether or not the evader is in the connected component X. This is precisely the question of *region clearing*.

An Example

Figure 2 shows a map "Gates" from (Gerkey, Thrun, and Gordon 2006). This graph has 32 nodes and 35 edges. To model that the evader is outside of the scenario, we add an extra disconnected vertex 0.

We have constructed an MCK model in which the pursuers all start from node 1 and have local visibility, that is, each one can observe its current node and any adjacent nodes. The initial position of the evader may be any node, including the node 0 that is outside the scenario. We use the synchronous scheduler and assume that the evader has unit velocity, that is, in a round, it can only traverse a single edge. (The pursuers always have unit velocity.) The effect of the general description above is so that the evader strategy is to move randomly. The pursuers are allowed to move to adjacent nodes.

We utilize the synthesis theory above and the MCK model checker to discovery an optimal strategy, in terms of the number of pursuers and the search length, to clear the map. First, we use MCK's `spr_g` algorithm to work on the game of one pursuer and the formula (1) with $\alpha = pos_e \notin X$, where $X = \{1..32\}$. MCK returns True in 1.1 seconds, which means that there is no successful strategy for a single pursuer.[1]

Second, we use the `spr_xn` algorithm to work on the game of 2 pursuers and the formula (2). As we increase the search length parameter n in this formula, the MCK finds a counterexample at minimal length 10, in 4807 seconds. The successful strategy derived from the counter-example is $(1,1) \to (2,31) \to (28,6) \to (27,7) \to (25,8) \to (24,9) \to (23,8) \to (20,12) \to (18,13) \to (16,15)$, where a pair (x, y) denotes the positions of two pursuers at a specific time. If the pursuers see the evader while executing this strategy, they know that the evader is present, otherwise, at the end of this path, they know that there is no evader in the region X (and the evader must be at the dummy position 0). Note that the computation does more than find this solution: it also establishes that the minimal number of pursuers is two, and that there is no shorter strategy using two pursuers.

Performance in Two Classes of Problems

For some information on how our approach scales, we have conducted experiments on two classes of game models. The class of connected random graphs $\mathcal{G}_{rand}(n, m)$ is generated using two parameters: the number of nodes n and the number of edges m. In our experiments the random graphs are generated by the routine `gnm_random_graph` from the software networkx (`http://networkx.lanl.gov/`). To make sure that a graph is connected, we set up an iteration process to ignore all non-connected graphs and keep the first connected one. All game settings for $\mathcal{G}_{rand}(n, m)$ follow the Gates example, i.e., visibility is of adjacent nodes, and the scheduler is synchronous. The second class of graphs $\mathcal{G}_{grid}(n, m, l)$ are in the form of Manhattan Grids of size $n \times n$ with m randomly generated holes, each of which has a size of at most $l \times l$. Figure 6 gives an example of a grid of

[1] Our experiments were conducted on a Ubuntu Linux system (3.06GHz Intel Core i3 with 4G memory). Each process is allocated up to 500M memory.

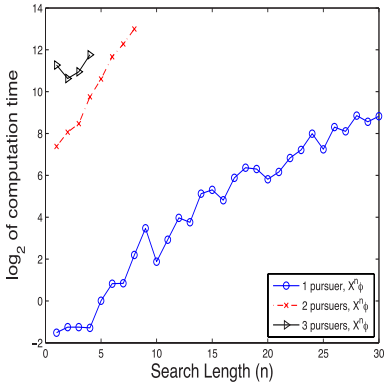

Figure 3: Computation time versus search depth in $\mathcal{G}_{rand}(32, 35)$, evader speed 1.

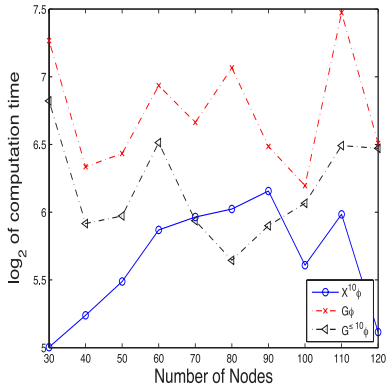

Figure 4: Computation time versus n in $\mathcal{G}_{rand}(n, m)$, $n + m = 300$. 1 pursuer.

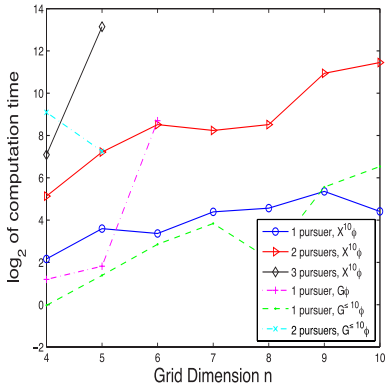

Figure 5: Computation time versus grid dimension n in $\mathcal{G}_{grid}(n, 2, 1)$.

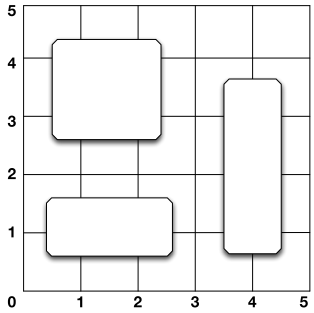

Figure 6: A Manhattan grid with holes

size 6x6 with three holes of size at most 3x3, i.e., a graph in $\mathcal{G}_{grid}(6, 3, 3)$. In the games on these graphs, we use the turn-based scheduler, and pursuers visibility is such that all nodes along straight lines from the pursuers position are visible, except when a hole blocks the view. In both types of games the evader has velocity 1.

The computation times we report are for deciding the existence of a valid strategy by model checking the corresponding formula. Construction of the counterexample after a specification has been determined to fail requires additional time, generally about double the model checking computation time. In general, the rate of growth in the computation time in our experiments is exponential, so we plot log base 2 of the computation time (in seconds), and a straight line in the diagram would indicate exponential growth. To keep the total cost of the experiments within reasonable bounds, we have generally scaled the experiments to the extent possible up to a maximal computation time of around 2^{16} seconds, i.e., roughly 10 hours.

As noted above, the number of possible states of knowledge of the pursuer agent p is potentially exponential: it may also scale exponentially in the depth n of the search path. One response to this complexity is to increase the number of pursuers, with the expectation of obtaining a shorter solution in this case. Figure 3 shows the performance of model checking the $X^n\phi$ formula as we increase n and the number of pursuers, working on random graphs of size 32 nodes and 35 edges. While we are able to handle significant search depths (as large as 30) for a single pursuer, there is a rapid blowup as we add pursuers, leading to a significant decrease in the search depth that can be handled efficiently. We would like to perform additional experiments to determine if this is because of inherent complexity of the problem or inefficiencies of the model checker.

We have been able to handle random graphs $\mathcal{G}_{rand}(n, m)$ with $n + m$ as large as 300 vertices plus edges within very reasonable time bounds. Figure 4 plots the performance of the two model checking algorithms given a fixed size graph (number of nodes plus number of edges = 300) as we vary the number of nodes. Since the two algorithms solve different problems, for a more balanced comparison we also plot a restriction $G^{\leq n}\phi$ of the $G\phi$ algorithm that applies the same approach with a maximal path length of n. The game has one pursuer. For the $X^n\phi$ algorithm, we assume a length 10 search strategy. We can see that both synthesis methods work efficiently in checking the existence of a pursuer strategy, taking less than 2 minutes to work on graphs of size 300.

Figure 5 shows the performance of the two algorithms on the games played on $\mathcal{G}_{grid}(n, 2, 1)$. In this case, the $G\phi$ algorithm scales poorly, but the $X^n\phi$ algorithm is able to handle moderate size grids for 1 or 2 pursuers.

Conclusion

A number of authors have considered similar pursuit-evasion problems, with an emphasis on heuristic approaches to synthesis of a strategy (Hollinger, Kehagias, and Singh 2010; Gerkey, Thrun, and Gordon 2006; Kehagias, Hollinger, and Singh 2009). In general these approaches are more efficient than ours, but we have not provided a detailed comparison because we solve a different problem: we are able to decide the existence of a strategy (or one of a given length), whereas the heuristic approaches are generally incomplete.

Synthesizing strategies by adapting model checking algorithms has been explored for temporal goals in fully-observable systems (Giunchiglia and Traverso 1999; Pistore and Traverso 2001) and partially-observable systems (Bertoli et al. 2006). MCK's algorithm for the perfect recall $G\phi$ case is closely related to that used in (Bertoli et al. 2006). Given the performance results comparing this with the $X^n\phi$ approach, an interesting topic for future research is to conduct a deeper comparison on other planning problems.

Another related line of work is based on alternating temporal logic ATL (Alur, Henzinger, and Kupferman 2002), which has been extended to epistemic variants, e.g., ATEL (van der Hoek and Wooldridge 2003). ATEL-based approaches have formed the basis for work on planning (van der Hoek and Wooldridge 2002; Jamroga 2004). However, work in this area has generally not been based on the perfect recall semantics for knowledge that we use in this paper.

We have shown one case where temporal-epistemic synthesis can be reduced to model checking. It would be interesting to find other such examples. Our result exploits the linear nature of the counter-examples returned by MCK, but one can envisage generalisations to tree-like counter-examples. MCK also has a bounded model checking algorithm for the synchronous perfect recall semantics, but it cannot be applied to the formula we use in this paper because the knowledge operator in the formula occurs in negative form. We believe that the problems in this paper constitute an interesting set of challenge problems for the epistemic model checking field.

References

Alur, R.; Henzinger, T. A.; and Kupferman, O. 2002. Alternating-time temporal logic. *Journal of the ACM* 49(5):672–713.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2006. Strong planning under partial observability. *Artifical Intelligence* 170(4-5):337–384.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers* C-35(8):677–691.

Clarke, E.; Grumberg, O.; and Peled, D. 1999. *Model Checking*. The MIT Press.

Eijck, D. 2004. Dynamic epistemic modelling. Technical Report E 0424, CWI. Software Engineering [SEN].

Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. 1995. *Reasoning About Knowledge*. The MIT Press.

Gammie, P., and van der Meyden, R. 2004. MCK: Model checking the logic of knowledge. In *Proc. Conf. on Computer Aided Verification, CONCUR'04*, 479–483.

Gerkey, B.; Thrun, S.; and Gordon, G. 2006. Visibility-based pursuit-evasion with limited field of view. *Int. Journal of Robotics Research* 25:299–316. (graph in videos at http://ai.stanford.edu/~gerkey/research/pe/).

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *Recent Advances in AI Planning, European Conf. on Planning, ECP'99*, 1–20.

Hollinger, G.; Kehagias, A.; and Singh, S. 2010. GSST: anytime guaranteed search. *Autonomous Robots* 29:99–118.

Huang, X.; Maupin, P.; and van der Meyden, R. 2011. Model checking knowledge in pursuit-evasion games. In *Int. Joint Conf. on Artificial Intelligence, IJCAI'11*.

Isaacs, R. 1965. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. Wiley and Sons (1965), Dover (1999).

Jamroga, W. 2004. Strategic planning through model checking of ATL formulae. In *Artificial Intelligence and Soft Computing - ICAISC 2004*, 879–884.

Kacprzak, M.; Nabialek, W.; Niewiadomski, A.; Penczek, W.; Pólrola, A.; Szreter, M.; Wozna, B.; and Zbrzezny, A. 2008. VerICS 2007 - a model checker for knowledge and real-time. *Fundamenta Informaticae* 85(1-4):313–328.

Kehagias, A.; Hollinger, G.; and Singh, S. 2009. A graph search algorithm for indoor pursuit/evasion. *Mathematical and Computer Modelling* 50:1305–1317.

Lachner, R.; Breitner, M. H.; and Pesch, H. J. 2000. Real-time collision avoidance: Differential game, numerical solution, and synthesis of strategies. In *Adv. in Dynamic Games and Applications*. Birkhäuser. 115–135.

Lomuscio, A.; Qu, H.; and Raimondi, F. 2009. MCMAS: A model checker for the verification of multi-agent systems. In *Proc. Conf. on Computer Aided Verification, CONCUR'09*, number 5643 in LNCS, 682–688.

Pistore, M., and Traverso, P. 2001. Planning as model checking for extended goals in non-deterministic domains. In *Proc. Int. Joint Conf. on Artificial Intelligence, ICJAI'01*, 479–486.

Su, K.; Sattar, A.; and Luo, X. 2007. Model checking temporal logics of knowledge via OBDDs. *The Computer Journal* 50(4):403–420.

van der Hoek, W., and Wooldridge, M. 2002. Tractable multiagent planning for epistemic goals. In *Int. Joint Conf. on Autonomous Agents and Multiagent Systems, AAMAS'02*, 1167–1174.

van der Hoek, W., and Wooldridge, M. 2003. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica* 75(1):125–157.

van der Meyden, R., and Su, K. 2004. Symbolic model checking the knowledge of the dining cryptographers. In *Proc. IEEE Workshop on Computer Security Foundations, CSFW'04*, 280–291.

van der Meyden, R., and Vardi, M. Y. 1998. Synthesis from knowledge-based specifications. In *Proc. Int. Conf. on Concurrency Theory, CONCUR'98*, number 1466 in LNCS, 34–49.

Vidal, R.; Shakernia, O.; Kim, J.; Shim, D. H.; and Sastry, S. 2002. Probabilistic pursuit-evasion games: Theory, implementation, and experimental evaluation. *IEEE Transactions on Robotics* 18(5):662–669.