Electronic Notes in Theoretical Computer Science 85 No. 2 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume85.html 24 pages

Knowledge as Strategic Ability

Sieuwert van Otterloo[1] Wiebe van der Hoek[1]
Michael Wooldridge[1]

Department of Computer Science
University of Liverpool
Liverpool L69 7ZF, UK

Abstract

The ultimate goal of our research is to develop techniques for model checking knowledge properties of multi-agent systems. ATEL, an extension of the Alternating-time Temporal Logic of Alur et al, is a logic for specifying epistemic and strategic properties of such systems. We present a technique for reducing the ATEL model checking problem to one of model checking in ATL, whereby epistemic relations are explicitly encoded in ATL models as as dynamic transitions. The techniques is illustrated by means of a knowledge game, which is used as a running example throughout the paper.

1 Introduction

Alternating-time Temporal Logic (ATL) is a powerful logic for reasoning about the abilities of coalitions in multi-agent systems [2,3]. ATL has been extended to Alternating-time Temporal Epistemic Logic (ATEL) in which the knowledge of agents, (in the sense of the Fagin-Halpern-Moses-Vardi model of knowledge [6]), is also considered [11]. ATEL can be used to express properties of agent communication, for instance whether communication is necessary in a certain situation, and whether communication has been successful.

Model checking is a widely used, and highly successful method for evaluating logic formulas on finite state systems [5]. A general method for ATEL model checking would clearly be of great interest for the verification of agent communication protocols and the planning of communication in multi-agent systems. While automatic model checkers for ATL exist [1], no such model checkers yet exist for ATEL. In the line of a previous approach [10], this paper

[1] Email: {sieuwert,wiebe,mjw}@csc.liv.ac.uk

further explores use of existing model checking tools for protocols involving knowledge.

A model checker takes a specification of a transition system T and a formula ϕ, and evaluates the validity (or truth) of ϕ on T; typically, we write $T \models \phi$ to mean that ϕ is true in the system T [5]. From a mathematical point of view, a model checker is a function MC which takes a system T and a formula ϕ as input and return a truth value $MC(T, \phi)$. Since ATEL appears to be more expressive than ATL, it may not be possible to find a general reduction R from an ATEL formula ϕ to an ATL-formula $R(\phi)$ such that $T \models \phi \Leftrightarrow MC(T, R(\phi))$. We here assume that T can serve as a model for ATEL *and* ATL — this seems a reasonable assumption, since the ATEL models we define contain an ATL model. Instead of only converting the formula one wants to check, changes to the transition system T might also be considered. That is, we can try to construct functions t and f such that $T \models \phi \;\Leftrightarrow MC(t(T), f(\phi))$. These functions are intended to work on models which represent systems in which each step is completely determined by a single agent and in which no infinite computations can occur. These systems are called *turn-based acyclic*. This is a limited but important class of models because it allows one to use ATEL for studying extensive games [9].

An argument for the existence of such reductions is that epistemic logic is a modal logic like ATL [4]. Both have a Kripke semantics, and thus can be evaluated on a transition system. In ATL, transitions correspond to computation steps, while ATEL contains transitions corresponding to computations and epistemic uncertainty. In order to evaluate epistemic formulas, one may extend the transition system with additional transitions modelling the epistemic relations. If these alterations to the transition system are manageable, (in that there is no exponential blow-up in the number of transitions), then this might be a good way to model check.

Of course, we do not claim that all knowledge-related model checking problems can be solved using existing model checkers. But, nevertheless, we think it is useful to investigate which problems can be solved using existing tools. In this paper we show one approach to model checking knowledge and evaluate the relative merits of this method.

In section 2 we introduce the example protocol and the logic ATEL used to reason about the protocol. Section 3 informally presents the main idea of this paper. Section 4 explains the theory underpinning the technique. Section 5 formally defines the method and contains a correctness proof. In section 6, the illustrate the technique by using it to prove all the properties introduced in section 2. Section 7 presents some conclusions.

2 A Running Example

Knowledge games [12] based on card deals are a rich source of examples of multi-agent Knowledge Problems. The example given here is derived from the Russian Cards Problem [13]. It involves three agents, sharing five cards. Assume agent A holds cards 0 and 1, agent B has cards 2 and 3, while agent C holds card 4. We denote this deal by 01|23|4. All agents know their own cards, which cards exist and how many cards each agent has. All of this is common knowledge, and nothing else is known. Thus, for agent A the deal 01|34|2 is a possibility, while 01|2|34 is not (it knows B has two cards), and neither is 04|23|1, because A knows its own cards. Agent B, however considers 04|23|1 a possibility.

Even without the additional richness allowed by introducing actions into this framework, interesting properties of this scenario can be formulated in epistemic logic. In the next table we list three properties of interest. In the logical formulation often used for these situations [12,13] the proposition X_i is used to express that agent X holds card i. To express the fact that agent X knows ϕ, we write $K_X\phi$.

$K_A a_0$	A knows it has card 0 (true)
$K_B a_0$	B knows A has card 0 (false)
$K_A \neg K_B a_0$	A knows that B does not know that A has card 0 (true)

All formulas are evaluated with the card deal 01|23|4. Epistemic formulas are evaluated before anything has been communicated. To make this a *dynamic* scenario, we introduce one action: A is allowed to make one of three statements, each corresponding to a logical formula. However, A can only make true statements. The whole structure of this situation, i.e., which card deals are possible, which statements A is allowed to make, that only A can say something, that each agent only knows its own cards, is assumed to be common knowledge. The possibilities are as follows.

$a_4 \vee a_0$	I have card 4 or card 0
$a_1 \vee a_2$	I have card 1 or card 2
$a_3 \vee a_4$	I have card 3 or card 4

For each possible world at least one statement is true. The limitation to just these three moves may seem artificial, but it makes the example smaller and thus easier to understand.

Suppose that in the actual world 01|23|4 agent A opts to say: I have card 1 or card 2 (the second of these formulas). This clearly changes the knowledge of the agents — both of those listening to the statement *and* the speaker of the statement (who knows that epistemic state of those listening to the statement has changed). The following formulas were false in the initial situation, so it is interesting to see whether they are true in the situation after that statement.

$K_B a_1$	B knows that A has card 1 (true)
$K_A K_B a_1$	A knows that B knows that A has card 1 (false)

The use of ATEL allows one to formulate abilities of (coalitions of) players. In addition to the epistemic operators K_i, ATEL contains two additional operators for each set of agents Γ: $\langle\langle\Gamma\rangle\rangle \diamond \phi$ and $\langle\langle\Gamma\rangle\rangle \Box \phi$. These operators refer to what the agents in Γ can achieve based on their available choices. In ATL and ATEL, $t \models \langle\langle\Gamma\rangle\rangle \diamond \phi$ is true if and only if there is a set of strategies, one for each agent in Γ, such that if all agents in Γ follow their strategy, eventually a state s such that $s \models \phi$ will be reached. For all Γ and ϕ, $t \models \langle\langle\Gamma\rangle\rangle \Box \phi$ is true if and only if there is a set of strategies, one for each agent in Γ, such that if all agents in Γ follow their strategy, $s \models \phi$ is true for all future states s.

In this example, we would like to express that agent A can achieve certain goals by selecting the right moves. The following formulae, which combine both epistemic and strategic modalities, are evaluated in 01|23|4.

$\langle\langle A\rangle\rangle \diamond K_B a_1$	A can cause B to know that it has card 1 (true)
$K_A \langle\langle A\rangle\rangle \diamond K_B a_1$	A knows it can cause B to know that A has card 1 (false)

The above formulas have been chosen because they illustrate some interesting combinations of different operators.

The semantics we use for ATEL is one in which no knowledge restrictions are placed upon strategies. This is the simplest semantics and the one proposed originally [11]. In this semantics, players can base their decisions upon things they are supposed not to know. This may seem counter-intuitive in certain situations. If needed it can be solved by assuming a different semantics [8,7].

3 Modelling Epistemic Relations

The initial situation of the card game example can be captured in a Kripke structure [6,4]. Nodes in the structure correspond to different possible deals, and arcs capture the epistemic alternatives of agent (i.e., an arc $d_1 \rightarrow d_2$ for agent i means that i cannot distinguish the deal d_1 from deal d_2). Figure 1 shows just four of the states with one connection of each kind, but the actual number of different card deals in our example is 30 (note that different styles of lines connect states indistinguishable for different agents in this figure — reflexive arcs are not drawn).

Dynamic transitions are represented in figure 2. A choice is available only in the initial situation: agent A has two options in the depicted situation; the third possible statement, $a_3 \vee a_4$, is not true in under the initial card deal and therefore this is not a possible action for A.

Fig. 1. epistemic relations

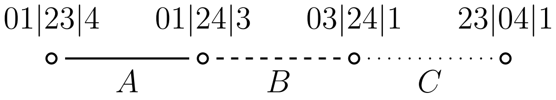


Fig. 2. dynamical transitions

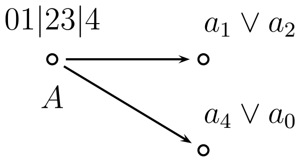
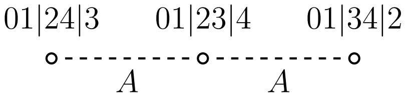


Fig. 3. epistemic model T



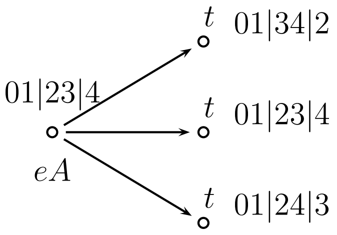
We need to combine the information contained in the model describing the choices (figure 2) and the epistemic information of figure 1. The idea is to *explicitly encode the epistemic transitions as a special kind of choice.* Since these transitions must not be confused with the transitions corresponding to actions, we introduce additional *epistemic agents*, for whom the choices will be the epistemic alternatives of the "real" agents. Thus in addition to the agents $\{A, B, C\}$ we use $\{eA, eB, eC, actA, actB, actC, E\}$: we will call eA, eB, eC the epistemic agents. Each agent eX can choose a next state that is indistinguishable for agent X.

To better understand this idea, consider the formula $\phi = K_A p$, where p is some proposition regarding the card deal. We will evaluate this formula in the epistemic situation of figure 1. We consider 01|23|4 the current card deal. We call this model T. The three states that agent A considers possible are shown in figure 3 — the dashed lines suggest the equivalence relation for agent A. (The possible choices of agent A are not in the picture because the formula ϕ does not use the choices of A.)

The procedure we propose will produce a new model $T' = t(T, \phi)$ and a new formula $\phi' = f(\phi)$. The new model T' must encode information about the knowledge of A. Therefore from the state with the real card deal, choices are available to agent eA to states with card deal that A considers possible. This new model is shown in figure 4. In this model the proposition t has been introduced. It indicates that a state is a final state.

The new formula ϕ' is derived from ϕ by replacing knowledge operators by

Fig. 4. System T'



strategic operators. The intended meaning of the new formula is 'Whatever choice eA makes, p will hold'. This can be expressed by the formula $\langle\langle\rangle\rangle\diamond(t\wedge p)$.

This intuitive idea is formalized in the next sections. The title of this paper refers to the interpretation of knowledge properties in ATEL as ATL formulas referring to strategic ability: ATL formulas are used to express the existence of 'winning' strategies — strategies which allow players to reach or avoid certain situations.

4 Definitions

In this section we define the languages that are used in the remainder of the paper, and the semantic models over which these languages are interpreted. The most important languages we deal with are ATL and ATEL. A common subset of these languages is propositional logic, which will be defined as well.

Definition 4.1 [Turn-Based System] A *turn-based system* (TBS) is a tuple $(S, P, \pi, \Sigma, a, \mathit{label})$, where S is a set of states, P a set of propositions, $\pi : P \times S \to \{\mathsf{true}, \mathsf{false}\}$ is the function interpreting all propositions, Σ is the set of agents, $a \subseteq S \times S$ is a transition relation and $\mathit{label} : a \to \Sigma$ labels all transitions with one of the agents. It is required that transitions with the same source must have the same label: $\forall s, t, t'\ \mathit{label}(s, t) = \mathit{label}(s, t')$.

The intended meaning of labels on transitions is that these labels indicate the agent that can decide which transition will be the next step in the computation. Thus this kind of system models a game-like situation, such as chess or checkers. We will use $\mathcal{T}$ as a variable for TBS.

It is possible to attach labels to positions instead of states. This does not result in important differences but some details will be different. In our formalisation no labels are attached to nodes from which no transitions leave. We see this as an advantage. The domain of the label function is the relation a, so only pairs (s, t) between which there is a transition get a label attached.

Definition 4.2 [Turn-Based Epistemic System] A *turn-based epistemic system* (TBES) is a tuple $((S, P, \pi, \Sigma, a, \mathit{label}), \{\sim_X | X \in \Gamma\})$. The first element,

$(S, P, \pi, \Sigma, a, label)$, is a TBS, while the second set contains for each agent X an equivalence relation $\sim_X \subseteq S \times S$.

The variable $\mathcal{B}$ will be used to denote TBES.

Definition 4.3 [Pointed System] Let M be either a TBS or a TBES and let S be the set of states of M. For each state $s \in S$ in the system M, the tuple (M, s) is a *pointed system.*

We will write TBS^* for the set of all pointed TBS and $TBES^*$ for the set for all pointed TBES.

Definition 4.4 [Propositional Logic] Let P be a set of propositions. Standard propositional logic PL is the smallest language $\mathcal{L}$ such that $P \subset \mathcal{L}$ and such that for all $\phi \in \mathcal{L}$ and $\psi \in \mathcal{L}$ it is the case that $\neg\phi \in \mathcal{L}$ and $\phi \vee \psi \in \mathcal{L}$.

Given an interpretation function $\pi : P \times S \to \{\mathsf{true}, \mathsf{false}\}$ and a state $s \in S$ we can interpret propositional logic formulas as follows.

$\pi, s \models p$ if and only if $\pi(p, s)$ is *true*

$\pi, s \models \neg\phi$ if and only if not $\pi, s \models \phi$

$\pi, s \models \phi \vee \psi$ if and only if $\pi, s \models \phi$ or $\pi, s \models \psi$

Alternating-time Temporal Logic extends propositional logic with two strategic operators.

Definition 4.5 [Alternating-time Temporal Logic] Let P be a set of propositions and Σ a set of agents. The language ATL with propositions P is the smallest language L such that $P \subset L$ and for any $\phi, \psi \in L$ and $\Gamma \subseteq \Sigma$ it is the case that $\neg\phi \in L$, $\phi \vee \psi \in L$,$\langle\langle\Gamma\rangle\rangle \diamond \phi \in L$ and $\langle\langle\Gamma\rangle\rangle\Box\phi \in L$.

Let $\mathcal{T} = (S, P, \pi, \Sigma, a, label)$ be a TBS and $s \in S$. We will define, for each formula $\phi \in ATL$, the circumstances under which $\mathcal{T}, s \models \phi$. The semantics for ATL is the semantics of propositional logic extended with two new clauses for the two new operators. These operators make use of strategies. A strategy is a function $f : S^+ \to S$: given a sequence of states $(\ldots, s)$ ending in state s the strategy selects a next state t such that $(s, t) \in a$.

A *run* r is a sequence of states $r = (s_0, s_1, \ldots, s_n)$ such that all pairs (s_i, s_{i+1}) are element of a and there is no t such that $(s_n, t) \in a$. We say that in a certain run $r = (s_0, s_1, \ldots)$ the agent X used strategy f if for all $(s_i, s_{i+1}) \in r$ with $label(s_i, s_{i+1}) = X$ it is the case that $f((s_0, \ldots, s_i)) = s_{i+1}$. It is not important what $f(s)$ is for states s in which it is not X's turn. This idea of a strategy is used in the following definition, which gives the semantics of ATL.

$\mathcal{T}, s \models p$ if and only if $\pi(p, s)$ is *true*

$\mathcal{T}, s \models \neg\phi$ if and only if not $\mathcal{T}, s \models \phi$

$\mathcal{T}, s \models \phi \vee \psi$ if and only if $\mathcal{T}, s \models \phi$ or $\mathcal{T}, s \models \psi$

$\mathcal{T}, s_0 \models \langle\langle\Gamma\rangle\rangle \diamond \phi$ if and only if there is a strategy for each agent in Γ such that in all runs $r = (s_0, s_1, \ldots)$ in which the agents in Γ used their strategy, a state s_i exists such that $\mathcal{T}, s_i \models \phi$

$\mathcal{T}, s_0 \models \langle\langle\Gamma\rangle\rangle\Box\phi$ if and only if there is a strategy for each agent in Γ such that in all runs $r = (s_0, s_1, \ldots)$ in which the agents in Γ used their strategy, in every state s_i it holds that $\mathcal{T}, s_i \models \phi$.

It is possible to define a semantics for ATL for systems that are not turn based. However, a slightly different notion of a strategy must be used [2].

In a turn based system, we can define $\langle\langle\Gamma\rangle\rangle\Box\phi := \neg\langle\langle\Sigma\backslash\Gamma\rangle\rangle\diamond\neg\phi$. This is not true in all ATL semantics, but for turn based systems we can use the minimax statement from game theory. The minimax theorem for games states that any zero-sum game has a unique value and that both players have a strategy to reach that value. Instead of applying this theorem we will provide a proof in terms of TBS.

Theorem 4.6 *For any TBS $\mathcal{T} = (S, P, \pi, \Sigma, a, label)$ and any state s*

$$\mathcal{T}, s \models \langle\langle\Gamma\rangle\rangle \diamond \phi \leftrightarrow \neg\langle\langle\Sigma \setminus \Gamma\rangle\rangle\Box\neg\phi$$

Proof. *Let $\mathcal{T} = (S, P, \pi, \Sigma, a, label)$ be a TBS and ϕ an ATL formula. We will show that for any state $s \in S$ either $\mathcal{T}, s \models \langle\langle\Gamma\rangle\rangle\diamond\phi$ or $\mathcal{T}, s \models \langle\langle\Sigma\backslash\Gamma\rangle\rangle\Box\neg\phi$. For this proof we construct a function $v : S \to \{1, \ldots, \infty\}$. The function is defined by the next rules. The definition is recursive: we first define when it is 1. Then we define for any value n, when the function takes the value $n+1$. Finally we assign the value ∞ to the states for which the function did not have a value yet.*

- *If $\mathcal{T}, s \models \phi$ then $v(s) = 1$*
- *If there is no transition $(s, t) \in a$ then $v(s) = \infty$*
- *If there is a transition $(s, t) \in a$ and $label(s, t) \in \Gamma$, then $v(s) = 1 + \min_{(s,t') \in a} v(t')$.*
- *If there is a transition $(s, t) \in a$ and $label(s, t) \notin \Gamma$, and for all $(s, t') \in a$ $v(t') < \infty$, then $v(s) = 1 + \max_{(s,t') \in a} v(t')$.*
- *$v(s) = \infty$ otherwise*

Since v is a function for any s either $v(s) < \infty$ or $v(s) = \infty$, and never both. We will show that $v(s) < \infty$ implies $\mathcal{T}, s \models \langle\langle\Gamma\rangle\rangle \diamond \phi$ and $v(s) = \infty$ implies $\mathcal{T}, s \models \langle\langle\Sigma \setminus \Gamma\rangle\rangle\Box\neg\phi$.

Assume that $s \in S$. We will show by induction for any natural number n, that $v(s) = n$ implies $\mathcal{T}, s \models \langle\langle\Gamma\rangle\rangle \diamond \phi$. The strategy we propose for coalition Γ is to choose the next state t for which $v(t)$ is minimal. This is a valid strategy. Assume that $(s_0, s_1, s_2, \ldots)$ is a run in which Γ uses this strategy. If $v(s_i) > 0$ then $v(s_{i+1}) < v(s_i)$. This can be seen from the second and third line of the definition. Therefore eventually a state t_j will be reached with $v(s_j) = 0$, and then $\mathcal{T}, s_j \models \phi$. This proofs that there is a strategy such that $\mathcal{T}, s \models \langle\langle\Gamma\rangle\rangle \diamond \phi$.

Now we show that $v(s) = \infty$ implies $\mathcal{T}, s \models \langle\langle\Sigma \setminus \Gamma\rangle\rangle\Box\neg\phi$. Assume a strategy for coalition $\Sigma \setminus \Gamma$ such that they choose the next state s which maximizes $v(s)$. it is clear that $v(s) = \infty$ implies that $\mathcal{T}, s \not\models \phi$. We will show that in any run $(s_0, s_1, s_2, \ldots)$ in which the agents follow the strategy, $v(s_j) = \infty$ implies $v(s_{j+1}) = \infty$. First assume that label$(s_j, s_{j+1}) \in \Gamma$. If $v(s_{j+1}) < \infty$, then $v(s_j)$ would have been less than or equal to $v(s_{j+1})$. Next assume that label$(s_j, s_{j+1}) \notin \Gamma$. The strategy we assumed tells us that s_{j+1} was choses to maximize $v(s_{j+1})$. If $v(s_{j+1})$ would be finite, then $v(s_j)$ would be $1 + v(s_{j+1})$ hence finite. Therefore $v(s_{j+1}) = \infty$. This shows that ϕ will not become true in this run and therefore $\mathcal{T}, s \models \langle\langle\Sigma \setminus \Gamma\rangle\rangle\Box\neg\phi$. This concludes the proof. □

It may seem that the fact that we add agents might interfere with our appliation of the above theorem. This is not the case because the reduction method carefully constrains the actions of the new agents. This proof and the correctness proof of the reduction method do not depend on each other.

Definition 4.7 [Alternating-time Temporal Epistemic Logic] Let P be a set of propositions and Σ a set of agents. The language $ATEL$ with propositions P is the smallest language L such that $P \subset L$ and for any $\phi, \psi \in L$, $\Gamma \subseteq \Sigma$ and $X \in \Sigma$ it is the case that $\neg\phi \in L$, $\phi \vee \psi \in L$, $K_X\phi \in L$ $\langle\langle\Gamma\rangle\rangle \diamond \phi \in L$ and $\langle\langle\Gamma\rangle\rangle\Box\phi \in L$.

ATEL can be interpreted over TBES. The interpretation for all connectives and operators also appearing in ATL is similar to the interpretation in ATL. We will omit these similar definitions and only define the interpretation of the knowledge operator.

$\mathcal{B}, s \models K_X\phi$ if and only if for each state s' with $s' \sim_X s$ it holds that $\mathcal{B}, s' \models \phi$

5 The Reduction

We now present our main contribution, which permits a reduction of the ATEL model checking problem to ATL model checking. The reduction consists of two functions

$$f : ATEL \rightarrow ATL$$
$$t : TBES^* \times ATEL \rightarrow TBS^*$$

such that

$$t((\mathcal{B}, s), \phi) \models f(\phi) \qquad \text{if and only if} \qquad \mathcal{B}, s \models \phi.$$

The definition of f makes use of an auxiliary function f_1, while the definition of t makes use of auxiliary functions t_1, t_2, t_3.

Throughout this section, the formula $\mu = K_A b_1$ evaluated under card deal 01|23|4 is used as an example to show how all functions work. See figure 3 on page 5 for the epistemic accessability relations for this example and figure 4 (page 6) for an illustration of the resulting system. In this figure only the states reachable from the initial state are shown.

The next table shows the domain and range of the functions we will define next.

function	domain	range
number	$ATEL$	$ATEL^n$
f	$ATEL$	ATL
f_1	$ATEL^n$	ATL
t	$TBES^* \times ATEL$	TBS^*
t_1	$TBES^* \times ATEL^n$	TBS^*
t_2	$TBES^* \times ATEL^n$	trees
t_3	$TBES^* \times ATEL^n \times \{0, 1, \ldots\}$	trees

The first step in the transformation is that in the input formula ϕ, all subformulas (including the whole formula) are assigned a number from the set $\{0, 1, 2, \ldots\}$. The whole formula gets the number 0. The numbers are written as in $\phi^i \vee \psi^j, K_x\phi^i, \langle\langle\Gamma\rangle\rangle \diamond \phi^i$. Subformulas caused by negation, as in $\phi = \neg\psi$ get the same number, as in $(\neg\psi^1)^1$, otherwise every subformula receives a unique number. For instance the input formula $p \vee K_A q$ could be numbered as $(p^1 \vee (K_A q^3)^2)^0$. These numbers are used for matching steps introduced by t to operators introduced by f. The order in which subformulas are numbered does not matter. The set of all formulas with numbered connectives is called $ATEL^n$. We define the function *number* as numbering all subformulas top-down, left to right. We define $f(\phi) = f_1(number(\phi))$ and $t(\mathcal{E}, \phi) = t_1(\mathcal{E}, number(\phi))$.

For the example formula, we have:

$$number(\mu) = (K_A(b_1)^1)^0$$

The function f_1 is defined recursively, as follows.

$$f_1(\chi^k) = \begin{cases} \chi & \text{if } \chi \in PL \\ \langle\langle E\rangle\rangle \diamond (done_i \wedge f_1(\phi)) \vee (done_j \wedge f_1(\psi))) & \text{if } \chi = \phi^i \vee \psi^j \\ \neg f(\phi^i) & \text{if } \chi = \neg\phi^i \\ \langle\langle\rangle\rangle \diamond (done_i \wedge f_1(\phi)) & \text{if } \chi = K_X\phi^i \\ \langle\langle\Gamma \cup \{E\}\rangle\rangle \diamond (done_i \wedge f_1(\phi)) & \text{if } \chi = \langle\langle\Gamma\rangle\rangle \diamond \phi^i \end{cases}$$

It is possible that both the first and either the second or the third clause can be used. In this case, the first rule takes precedence because this yields the smallest translation.

The reason for the use of the environment agent E is that in case we want to determine whether $\mathcal{B}, s \models K_A p \vee K_B p$, the intuition is to define a model in which both transitions eA and transitions labeled eB begin in s. In the definition of a TBS it is required that transitions from the same node must have the same label. In order to overcome this, in the definition of t we will define that from s one transitions leads to a new state t_1, where all the needed eA transition can start, and one transition leads to t_2. All transitions from t_2 are labeled eB. The environment plays a similar 'administrative' role in the translation of the strategic operators.

The translation of our example formula is

$$f(\mu) = f_1(\mathit{number}(\mu)) = \langle\!\langle\rangle\!\rangle \diamond (\mathit{done}_1 \wedge b_1)$$

Suppose $\mathcal{B} = ((S, P, \pi, \Sigma, a, \mathit{label}), \{\sim_i\}_{i\in\Sigma})$ and that $\mathcal{B}, s$ is a pointed TBES. Let ϕ be a numbered $ATEL$ formula and let N be the set of all numbers used in ϕ. Assume $((S', P', \pi', \Sigma', a', \mathit{label}'), s') = t((\mathcal{B}, s), \phi)$. The next definitions apply.

$$\begin{aligned}
S' &= \{\mathit{state}(s,j) | s \in S, j \in N\} \\
P' &= P \cup \{\mathit{done}_i | i \in N \wedge i > 0\} \\
\Sigma' &= \{X | X \in \Sigma\} \cup \{eX | X \in \Sigma\} \cup \{E\} \\
\pi'(p, \mathit{state}(s,j)) &= \pi(p,s) \text{ for all } p \in P \\
\pi'(\mathit{done}_i, \mathit{state}(s,j)) &= \mathit{true} \text{ if } i = j, \mathit{false} \text{ otherwise}
\end{aligned}$$

There is no proposition done_0. The number 0 is assigned to the whole formula, but the function f is designed such that there is no need to refer to done_0. The intuition behind the definition of P' is that all old propositions can be used and the propositions done_i which can be used as labels. In order to interpret these label propositions every state must encode which labels are valid. Luckily at most one label is applied to any state, therefore we can attach to any state the number of the done_i proposition that is true. This explains the definition of π' and S'.

For the example model $((S', P', \pi', \Sigma', a', \mathit{label}'), s') = t(T, \mu)$ we have:

$$\begin{aligned}
&N = \{0, 1\} \\
&P' = \{a_i | 0 < i < 6\} \cup \{b_i | 0 < i < 6\} \cup \{c_i | 0 < i < 6\} \cup \{\mathit{done}_1\} \\
&\Sigma' = \{A, B, C, eA, eB, eC, E\} \\
&S' = \{\mathit{state}(ab|cd|e, x) | \{a, b, c, d, e\} = \{1, 2, 3, 4, 5\}, \\
&\qquad a < b, c < d, x = 0 \vee x = 1\}
\end{aligned}$$

and

$$\pi' : \pi'(done_1, state(ab|cd|e, j)) = true \text{ if } j = 1, false \text{ if } j = 0$$
$$\pi' : \pi'(p, state(ab|cd|e, j)) = \pi(p, ab|cd|e) \text{ for all other } p$$

We define the remaining components, a', $label'$ and s' by the means of the tree $T = t_2((\mathcal{B}, s), \phi)$. First, we explain what is meant by a tree and how these parts can be derived from a tree. Then we define the function t_2 that constructs such a tree.

The format of a tree is $node(state(s, j), C)$. The first element $state(s, j)$ must be in S' and is a state from the new model. The set C encodes where we can go from this state. C is a collection of pairs (X, U). The X is the agent which can make this transition. For any two elements (X, U) and (Y, V), X and Y must be equal. The second element of the pair, U, is itself a tree.

Suppose we have a tree $T = t_2((\mathcal{B}, s), \phi)$, and we wish to know the initial state s', the transition relation a', and the label relation $label'$ of the model $((S', P', \pi', \Sigma', a', label'), s') = t(\mathcal{B}, \phi)$. The initial state s' is the state at the root of the tree, so if $T = node(state(s_0, j_0), C_0)$ for some s_0, j_0 and C_0, then $s' = state(s_0, j_0)$. The set a' consist of all pairs $(state(s, j), state(t, k))$ such that $node(state(s, j), C)$ appears in the tree and $(X, node(state(t, k), D)) \in C$, so transitions are from a subtree to a child of that subtree. For the pair $state(s, j), state(t, k)$, the label of the transition would be X, since X is the first element of the pair $(X, node(state(t, k), D)) \in C$.

The function t_2 is defined in terms of $t3$. We will abuse notation slightly by writing $t_2(s, p)$ instead of $t_2((\mathcal{B}, s), p)$, and by writing $t_3(s, p, j)$ instead of $t_3((\mathcal{B}, s), p, j)$. In the following definition, the symbol S refers to the set of states of $\mathcal{B}$ and a refers to the transition relation of $\mathcal{B}$.

$$\begin{aligned}
t_2(s, \phi^i) &= t_3(s, \phi^i, i) \\
t_3(s, \phi^i, l) &= node(state(s, l), \emptyset) \text{ for any } \phi \in PL \\
t_3(s, (\phi^i \vee \psi^j)^k, l) &= node(state(s, l), \{(E, t_3(s, \phi^i, i)), (E, t_3(s, \psi^j, j))\}) \\
t_3(s, (\neg\phi^i)^i, l) &= t_3(s, \phi^i, l) \\
t_3(s, (K_X\phi^i)^j, l) &= node(state(s, l), \{(eX, t_3(s', \phi^i, i)) | s \sim_X s'\}) \\
t_3(s, (\langle\langle\Gamma\rangle\rangle \diamond \phi^i)^j, l) &= node(state(s, l), \{C_1, C_2\})
\end{aligned}$$

where

$$\begin{aligned}
C_1 &= (E, t_3(s, \phi^i, i)), \\
C_2 &= (E, node(state(s, 0), Z))), \quad \text{and} \\
Z &= \{(X, t_3(t, (\langle\langle\Gamma\rangle\rangle \diamond \phi^i)^j, 0)) | (s, t) \in a \wedge X = label(s, t)\}
\end{aligned}$$

The last clause of this definition defines the translation of the strategic operator. We have modelled this with steps of the original system in set Z, which are interleaved by steps of the environment. The environment can choose whether it wants to continue selecting steps from Z, or move on to evaluating the formula to know ϕ. In the corresponding translation of the formula we see that indeed the environment and the coalition must work together in the new ATL system, instead of only the coalition as in the original ATEL system.

The tree would be infinite if ϕ contained a strategic operator and the transition relation a of the TBES $\mathcal{B}$ we started with contained cycles. We have thus assumed that this relation is acyclic.

Applying this reduction to the example formula μ and model T, we get the following

$$t_2(01|23|4, \mu) \quad = \quad \mathit{node}(\mathit{state}(01|23|4, 0), \{X, Y, Z\})$$

where:

$$\begin{aligned}
X &= (eA, \mathit{node}(\mathit{state}(01|24|3, 1), \emptyset)),\\
Y &= (eA, \mathit{node}(\mathit{state}(01|23|4, 1), \emptyset)),\\
Z &= (eA, \mathit{node}(\mathit{state}(01|34|2, 1), \emptyset)),\\
s' &= \mathit{state}(01|23|4, 0), \quad \text{and}\\
a' &= \{(\mathit{state}(01|23|4, 0), \mathit{state}(01|23|4, 1)),\\
&\quad\ (\mathit{state}(01|24|3, 0), \mathit{state}(01|24|3, 1)),\\
&\quad\ (\mathit{state}(01|34|2, 0), \mathit{state}(01|34|2, 1))\}
\end{aligned}$$

The outcome of $t_3(T, \mu)$ is depicted in figure 4, where t is the same as done_1. The picture shows only four states. The two states not shown, $\mathit{state}(01|24|3, 0)$ and $\mathit{state}(01|34|2, 0)$ cannot be reached from the initial state ($\mathit{state}(01|23|4, 0)$ and have been omitted to simplify the figure.

Theorem 5.1 *For any pointed TBES $(\mathcal{B}, s)$ and any ATEL formula δ, $\mathcal{B}, s \models \delta$ if and only if $t((\mathcal{B}, s), \delta) \models f((\mathcal{B}, s), \delta)$.*

Proof. *Let $(\mathcal{B}, s)$ be a pointed TBES and δ an ATEL formula. Let$(\mathcal{T}, t) = t((\mathcal{B}, s), \delta)$ and let $\phi' = f((\mathcal{B}, s), \delta)$. We must show that $\mathcal{B}, s \models \delta$ if and only if $(\mathcal{T}, t) \models \phi'$. We will prove this by proving the stronger claim that for any subformula ψ^j of $\mathit{number}(\delta)$ and state $s \in \mathcal{B}$ such that $\mathit{state}(s, j)$ occurs in the tree $t_2((\mathcal{B}, \delta), 0)$, it holds that $(\mathcal{B}, s) \models \psi^j$ if and only if $(\mathcal{T}, \mathit{state}(s, j)) \models f_1(\psi)$. We will prove this last claim by induction.*

Let ψ^j be a subformula of $\mathit{number}(\delta)$. Suppose first that $\psi \in PL$. In that case $f_1(\psi^j) = \psi$. The interpretation of a formula in propositional logic only depends on the current state. π' is constructed such that for all p occurring in ψ, $\pi'(p, \mathit{state}(s, j)) = \pi(p, s)$, and therefore $(\mathcal{B}, s) \models \psi^j$ if and only if $(\mathcal{T}, \mathit{state}(s, j)) \models f_1(\psi)$, which we had to prove. This provides us with the inductive base.

For the next step suppose $\psi^j = (\neg\chi^j)^j$. We will show that $(\mathcal{B}, s) \models (\neg\chi^j)^j$ if and only if $(\mathcal{T}, state(s,j)) \models f_1((\neg\chi^j)^j)$.

$$\begin{aligned}
(\mathcal{B}, s) \models (\neg\chi^j)^j &\quad \Leftrightarrow \textit{(ATEL semantic)} \\
\textit{not } (\mathcal{B}, s) \models \chi^j) &\quad \Leftrightarrow \textit{(induction hypothesis)} \\
\textit{not } (\mathcal{T}, state(s,j)) \models \chi^j &\quad \Leftrightarrow \textit{(ATL semantic)} \\
(\mathcal{T}, state(s,j)) \models f_1((\neg\chi^j)^j) &
\end{aligned}$$

Now suppose that $\psi^j = (\phi^i \vee \chi^k)^j$.

$$\begin{aligned}
\mathcal{B}, s \models \psi^j &\quad \Leftrightarrow \textit{(ATEL sem.)} \\
\mathcal{B}, s \models \phi^i \textit{ or } (\mathcal{B}, s) \models \chi^k &\quad \Leftrightarrow \textit{(induction hyp.)} \\
\mathcal{T}, state(s,i) \models f_1(\phi^i) \textit{ or } (\mathcal{T}, state(s,k)) \models f_1(\chi^k) &\quad \Leftrightarrow \textit{(structure } t_3\textit{)} \\
\mathcal{T}, state(s,j) \models \langle\langle E \rangle\rangle \diamond (done_i \wedge f_1(\phi^i)) \vee (done_k \wedge f_1(\chi^k))) &\quad \Leftrightarrow \textit{(definition } f_1\textit{)} \\
\mathcal{T}, state(s,j) \models f_1(\psi^j) &
\end{aligned}$$

The argument given above depends on the definition of t_3, which is applied below. $t_3(s, (\phi^i \vee \chi^k)^j, l) = node(state(s,l), \{(E, t_3(s, \phi^i, i)), (E, t_3(s, \psi^j, j))\})$. One can see from this formula that the strategy for E can only make one choice out of two options. One options will be succesful if $(\mathcal{T}, state(s,i)) \models f_1(\phi^i)$, the other if $(\mathcal{T}, state(s,k)) \models f_1(\chi^k)$.

Next we deal with the strategic oprator. Let $\psi^j = (\langle\langle\Gamma\rangle\rangle \diamond \phi^i)^j$. We will first proof that $(\mathcal{B}, s) \models \psi^j \Rightarrow (\mathcal{T}, state(s,j)) \models f_1(\psi^j)$, and then that $(\mathcal{B}, s) \models \psi^j \Leftarrow (\mathcal{T}, state(s,j)) \models f_1(\psi^j)$.

Assume that $(\mathcal{B}, s) \models \psi^j$. This means that there is a strategy S for Γ such that $\forall r = (s, s_1, \ldots)$ in which agents in Γ use S a state s_i exists such that $(\mathcal{B}, s_i) \models \phi$.

The structure of $t_3(\psi^j)$ is given next.

$$\begin{aligned}
t_3(\psi^j) &= node(state(s,j), \{A, B\}) \\
A &= (E, t_3(s, \phi^i, i)) \\
B &= (E, node(state(s,0), Z))) \\
Z &= \{(label(s,t), t_3(t, \psi^j, 0)) | (s,t) \in a\}
\end{aligned}$$

Suppose that E decides to take option B for some amount of time and that Γ uses S. In that case for any run w we will arive in a state $state(w_x, l)$ for some l such that $(\mathcal{B}, w_x) \models \phi$. Assume that here the environment chooses option A. This brings us to a state $state(w_x, i)$ where $done_i$ will hold. Furthermore, by induction hypothesis, we know that $\mathcal{T}, state(w_x, i)) \models f_1(\phi^i)$. Therefore we know that $\Gamma \cup \{E\}$ has a strategy for bringing about in some state in every run $(done_i \wedge f_1(\phi^i))$, which means that $(\mathcal{T}, state(s,j)) \models \langle\langle\Gamma \cup \{E\}\rangle\rangle \diamond (done_i \wedge$

$f_1(\phi^i)) = f_1(\psi^j)$.

For the second part assume that $(\mathcal{T}, state(s,j)) \models f_1(\psi^j)$. Therefore $\Gamma \cup \{E\}$ has a strategy S such that for any run v starting in $state(s,j)$ there is a state v_x such that $\mathcal{T}, v_x \models (done_i \wedge f_1(\phi^i))$. Let w be any run of $\mathcal{B}$ starting in s in which all agents of Γ use the strategy which corresponds to their strategy S. We can translate this run w to a run v in $\mathcal{T}$ where the environment will repeatedly choose option B. Since $done_i$ is never true on this run, the strategy S must at a certain point v_x specify a different move for the environment. So assume that that v' is a run in which $\Gamma \cup \{E\}$ uses strategy S such that $v'_0 \dots v'_x = v_0 \dots v_x$ and at v'_{x+1} is a state $state(s',i)$. This is the only state of v' in which $done_i$ is true. Since we have assumed that the strategy S works for every run, we have $\mathcal{T}, state(s',i) \models (done_i \wedge f_1(\phi^i))$ and thus $\mathcal{T}, state(s',i) \models f_1(\phi^i)$. Using the induction hypothesis we know that $\mathcal{B}, s' \models \phi$. From the definition of t_3 we see that $v'_x = v_x = state(s',l)$ and therefore $w_x = s'$. Since we have shown the existence of such a state w_x for any run w in which Γ uses the strategy corresponding to S, we can conclude $(\mathcal{B}, s) \models \psi^j$.

Finally, suppose that $\psi^j = (K_X\phi^i)^j$. Then $f_1((K_X\phi^i)^j) = \langle\langle\rangle\rangle \diamond (done_i \wedge f_1(\phi^i))$, and $t_3(s, K_X\phi^i, j) = node(state(s,j), \{(eX, t_3(s',\phi,i)) | s \sim_X s'\})$. We must prove that $\mathcal{B}, s \models \psi$ if and only if $\mathcal{T}, state(s,j) \models f_1(\psi)$.

$$\begin{array}{rl}(\mathcal{B}, s) \models \psi^j & \Leftrightarrow \textit{(ATEL semantic)} \\ \forall t \sim_X s : (\mathcal{B}, t) \models \phi^i & \Leftrightarrow \textit{(induction hypothesis)} \\ \forall t \sim_X s : (\mathcal{T}, state(t,i)) \models f_1(\phi^i) & \end{array}$$

For any state t with $s \sim_X t$ there is in $\mathcal{T}$ a transition from $state(s,j)$ to $state(t,i)$. A statement regarding the empty coalition, such as $\langle\langle\rangle\rangle \diamond (done_i \wedge f_1(\phi^i))$, is only true if on all paths there is a state in which $done_i \wedge f_1(\phi)$ is true. Therefore the next equivalences hold.

$$\begin{array}{rl}\forall t \sim_X s : (\mathcal{T}, state(t,i)) \models f_1(\phi^i) & \Leftrightarrow \\ \mathcal{T}, state(t,j) \models \langle\langle\rangle\rangle \diamond (done_i \wedge f_1(\phi^i)) & \Leftrightarrow \\ \mathcal{T}, state(t,j) \models f_1(\psi^j) & \end{array}$$

This concludes our the proof of our statement. □

6 Implementation in Mocha

In this section we demonstrate how the technique we have described can be applied to the Russian cards scenario. The example code has two goals. The first is to show how a turn-based system can be modelled in Mocha [1] using a controller or scheduler module. Secondly it shows how the reduction method from this paper can be implemented in Mocha. The first part of this section explains how the protocol has been modelled in the first place. The second part shows which modifications and additions are being made when applying the reduction we propose. The Mocha code for this example can be downloaded from http://www.csc.liv.ac.uk/∼sieuwert/.

Modelling the Scenario in Mocha

Mocha is a intended for the modular verification of heterogeneous systems. It accepts a transition system described in the ReactiveModules language as input, and allows one to either simulate the system or else to check ATL properties on the system. The system is composed of several modules that run in parallel. Each variable can only be controlled by one module, but can be read by any module. Modules communicate by means of shared variables or by passing "events".

The parallel setup is convenient for instance in the case of modelling digital circuits, where each path between components is translated into a variable. Our example can be seen as a turn-based synchronous system [2] and this requires a different approach. To make sure only one module acts at any time and all the modules act in the appropriate order, a separate module named `Control` has been introduced. This module signals all processes if they are expected to do something. One part of this module encodes the order of steps in the system, making it easy to change the order of steps or introduce new steps.

The state of the world in the example consists of the deal of cards and all announcements made. The deal of cards can be stored in five variables d, e, f, g, h each containing one card. Agent A holds the cards contained in d, e, B holds f, g and C holds h.

In turn-based systems it is possible that more than one module is in certain situations allowed to change the value of a variable. In ReactiveModules this cannot be implemented directly. This issue of write-shared variables can be solved by giving `Control` write-access to these variables. Modules X that need to alter the value of such a variable y are in control of their own copies of this variables, called xy. The x is some identifier unique for module X to make sure all variables have a unique name. When the module `Control` ends the turn of a certain module X, it copies the value of xy into the real variable y.

The technique for implementing turn-based systems described above has been used to obtain the implementation of the protocol. Below the full program is given, followed by details about the role of each variable.

```
type cards :{zero,one,two,three,four}
type modules:{epA,epB,epC,none,actA,actB}

module actionA
 interface u1,u2:cards
 external focus:modules;d,e:cards
 atom step
  controls u1,u2
  reads u1,u2,d,e
  awaits focus
 init
  [] true -> u1':=one;u2':=one
 update
  [] focus'=actA&(d=four|d=zero|e=four|e=zero) ->
    u1':=four;u2':=zero
  [] focus'=actA&(d=one|d=two|e=one|e=two) ->
    u1':=one;u2':=two
  [] focus'=actA&(d=four|d=three|e=four|e=three) ->
    u1':=three;u2':=four
  [] default ->
 endatom
endmodule
```

```
module Control
 interface d,e,f,g,h:cards;
  stp:(0..10);
  focus:modules;
  finish:bool
 external af,ag,ah,bd,be,bh,cd,ce,cf,cg:cards

 atom step
  controls d,e,f,g,h,stp,focus,finish
  reads d,e,f,g,h,stp,focus,finish,af,ag,ah,bd,be,bh,cd,ce,cf,cg
 init
  [] true ->
    d':=zero;e':=one;f':=two;g':=three;h':=four;
    stp':=0;
    finish':=false;
    focus':=none
 update

-- insert here
[] focus=none & stp=0 ->   focus':=actA
[] focus=none & stp=1 -> finish':=true
-- end of insert

--receive feedback from processes
  [] focus=actA ->
   focus':=none;
   stp':=stp+1
 endatom
endmodule
```

The module that is allowed to make a transition is always stored in `focus`. It contains the name of a module or the value `none`. The `none` value indicates that all variables wait for the module `Control` to do its work.

The module `Control` sets the boolean variable `finished` to `true` if a run is finished. This variable can be used as a label is formulas, indicating a terminal state. Checks on the variable *stp* can be used as labels for other positions in the protocol.

The basic setup of a turn-based system has been implemented with one module `actionA`. This module encodes the actions agent A of our example can do. This agent can make one out of three statements, provided the statement is true. The statement is stored in the variables `u1,u2`. Initially, these variables have the same value. This indicates that no statement has been made yet. When these variable have a different value, the interpretation variables is that agent A makes the statement $a_{u1} \wedge a_{u2}$.

The proposition a_0 is expressed in the code as `d=zero|e=zero` ("d is zero or e is zero"), because `d,e` are A's cards.

The specification given contains some redundant variables and values. These variables will be used later on. In order to avoid having to include almost similar code later on we have not stripped the code from unused variables (`af,ag,ah,bd,be,bh,cd,ce,cf,cg`) or values (`epA,epB,epC,actB`).

Applying the reduction

In this section, we describe how an ATEL formula can be verified by checking a corresponding ATL formula on a modified model. Some of the modifications to the module are specific to the formula. First, we describe the addition of a module for each agent encoding that agent's epistemic accessibility relation. These modules corresponds to the epistemic agents of section 5. They do not depend on the specific formula.

For each of agent X an extra module called `epistemicX` was created. From the definition of knowledge in an interpreted system [6] it follows that an agent knows its own variables. Each epistemic module reassigns all other variables in any consistent manner. For most systems, including the example system this is straightforward to implement.

Agent A has access to its own cards. The module `epistemicA` should therefore rearrange the cards of B and C. The next module implements the rearrangements. The variables af, ag, ah are copies of f, g, h allowing this module to suggest new values for these variables.

```
module epistemicA
 interface af,ag,ah:cards
 external focus:modules;f,g,h:cards
 atom step
  controls af,ag,ah
  reads f,g,h,af,ag,ah
  awaits focus
 init
  []true->af':=one;ag':=one;ah':=one
 update
  []focus'=epA -> af':=f;ag':=g;ah':=h
  []focus'=epA -> af':=h;ag':=g;ah':=f
  []focus'=epA -> af':=f;ag':=h;ah':=g
  [] default ->
 endatom
```

```
endmodule
```

In the example, agent A can make one of three statements. These statements do not affect the knowledge of agent A, so they have not been used in the definition of `epistemicA`. For the knowledge of B and C, the statements are of course relevant. The module `epistemicB` is similar to `epistemicA`, except that it depends on the statement made by A. Extra conditions have been prefixed to the possible swaps. One of the conditions is "No statements has been made yet". In code, this is `u1=u2`, since by convention `u1,u2` are only equal if no statement has been made. The proposition encoded in `u1,u2` is true if one of the cards is indeed held by A:`(u1=d|u1=e|u2=d|u2=e)`.

```
module epistemicB
 interface bd,be,bh:cards
 external focus:modules;d,e,h,u1,u2:cards
 atom step
  controls bd,be,bh
  reads d,e,h,bd,be,bh,u1,u2
  awaits focus
 init
  []true->bd':=one;be':=one;bh':=one
 update
  []focus'=epB & (u1=u2|u1=d|u1=e|u2=d|u2=e)-> bd':=d;be':=e;bh':=h
  []focus'=epB & (u1=u2|u1=h|u1=e|u2=h|u2=e) -> bd':=h;be':=e;bh':=d
  []focus'=epB  &(u1=u2|u1=d|u1=h|u2=d|u2=h)-> bd':=d;be':=h;bh':=e
  [] default ->
 endatom
endmodule

module epistemicC
 interface cd,ce,cf,cg:cards
 external focus:modules;d,e,f,g,u1,u2:cards
atom step
 controls cd,ce,cf,cg
 reads d,e,f,g,cd,ce,cf,cg,u1,u2
 awaits focus
 init
  []true->cd':=one;ce':=one;cf':=one;cg':=one
 update
  []focus'=epC & (u1=u2|u1=d|u1=e|u2=d|u2=e) -> cd':=d;ce':=e;cf':=f;cg':=g
  []focus'=epC & (u1=u2|u1=d|u1=f|u2=d|u2=f)-> cd':=d;ce':=f;cf':=e;cg':=g
  []focus'=epC & (u1=u2|u1=d|u1=g|u2=d|u2=g)-> cd':=d;ce':=g;cf':=f;cg':=e
  []focus'=epC & (u1=u2|u1=f|u1=e|u2=f|u2=e)-> cd':=f;ce':=e;cf':=d;cg':=g
  []focus'=epC & (u1=u2|u1=g|u1=e|u2=g|u2=e)-> cd':=g;ce':=e;cf':=f;cg':=d
  []focus'=epC & (u1=u2|u1=f|u1=g|u2=f|u2=g)-> cd':=f;ce':=g;cf':=d;cg':=e
  [] default ->
 endatom
endmodule
```

The introduction of new modules imposes some additional bookkeeping requirements. The next lines must appear in the `Control` module. The lines are responsible for transferring the control back from each module to the

Control module. These lines are similar for the action and the epistemic modules. They replace the old part, starting with the comment --receive feedback from processes.

```
--receive feedback from processes
  [] focus=epA ->
   f':=af;g':=ag;h':=ah;
   focus':=none;
   stp':=stp+1
  [] focus=epB ->
   d':=bd;e':=be;h':=bh;
   focus':=none;
   stp':=stp+1
  [] focus=epC ->
   d':=cd;e':=ce;f':=cf;g':=cg;
   focus':=none;
   stp':=stp+1
  [] focus=actA ->
   focus':=none;
   stp':=stp+1
```

For each formula to check, we must alter the game structure: the epistemic modules of the agents involved must be used. In the program prior to modification, a comment appears saying `-- insert here`. At this point, the turns of the game are determined. In the original scenario, the module `actionA` was allowed to do one action (to make an announcement). This is modified to include epistemic transitions, matching the order of epistemic operators in the formula to check.

In section 2, several epistemic and ATEL formulas are given. Here we give, for each formula, the translation in ATL, together with a new set of lines to be inserted in the program. The ATL formulas are given in the input format of Mocha. For each example formula ϕ a statement `atl "anyname"` ϕ`;` is given. This assigns the name `anyname` to the ATL formula ϕ. The difference between the logical notation and the Mocha notation for formulas is that the diamond $\diamond$ is replaced by `F`. The propositions referring to the cards are translated into simple tests on the five variables `d,e,f,g,h`. The next table illustrates the translation of propositions.

proposition	Mocha test
a_0	`d=zero|e=zero`
b_0	`f=zero|g=zero`
c_0	`h=zero`

For the translation of epistemic operators we need labels, so that the formula can refer to the point in each run after the epistemic transitions have been made. These correspond to the $done_i$ propositions in the proof. The boolean variable `finish` was introduced to serve as a label for the final state of each run. Tests on the variable stp can be used to refer to other points in each run.

The next ATL formulas and modifications correspond to the first three example formulas (page 3). The first line is the original formula. The following lines must be included in the Control module. The line starting with atl contains the resulting ATL formula, and the last line indicates the model checking result (either passed or failed).

```
-- Ka a0
-- [] focus=none & stp=0 ->   focus':=epA
-- [] focus=none & stp=1 -> finish':=true
atl "simple1" <<>> F (finish & (d=zero|e=zero));
--passed

-- Kb a0
-- [] focus=none & stp=0 ->   focus':=epB
-- [] focus=none & stp=1 -> finish':=true
atl "simple2" <<>> F (finish & (d=zero|e=zero));
-- failed

-- Ka-Kb a0
-- [] focus=none & stp=0 ->   focus':=epA
-- [] focus=none & stp=1 ->   focus':=epB
-- [] focus=none & stp=2 -> finish':=true
atl "simple3" <<>> F (stp=1 & ~(<<>> F (finish & (d=zero|e=zero))));
-- passed
```

The results are correct, just as we expected. Since these examples are formulas in epistemic logic, this only shows that an ATL model checker can be used for checking epistemic properties.

The next examples illustrate the state of knowledge after A makes the statement $a_1 \vee a_2$. These examples are still only epistemic, but they must be evaluated at a specific point, in only some runs of the system. In order to express in ATEL that a formula must be true after a certain move, an ad hoc notation must be introduced. The original ATEL formula for the next example can be written down using t as a proposition which is true in a terminal state, and u as a proposition true if A has announced $a_1 \vee a_2$. The original ATEL formula for the first example is $\langle\langle\rangle\rangle \diamond (t \wedge (\neg u \vee K_B a_1))$. In a run in which A makes a different statement to $a_1 \vee a_2$, the value of u1 will not be equal to one. Therefore the test (∼u1=one) can be used in the code instead of u.

```
-- [after saying a1 or a2] Kb a1
-- [] focus=none & stp=0 ->   focus':=actA
-- [] focus=none & stp=1 ->   focus':=epB
-- [] focus=none & stp=2 -> finish':=true
atl "dyn1" <<>> F  (finish & ((~u1=one)|(d=one|e=one)) );
-- passed

-- [after saying a1 or a2] Ka Kb a1
-- [] focus=none & stp=0 ->   focus':=actA
-- [] focus=none & stp=1 ->   focus':=epA
-- [] focus=none & stp=2 ->   focus':=epB
-- [] focus=none & stp=3 -> finish':=true
atl "dyn2" <<>> F  (finish & ((~u1=one)|(d=one|e=one)) );
-- failed
```

Again the true formula passes and the false formula fails. This illustrates how one can check knowledge at different points than the initial situation. The selection of tests to use in the formulas seem somewhat ad hoc. We have opted not to introduce extra variables to act as fresh propositions, but to use tests on existing variables. Enough variables are already available to create a simple test for any interesting point in a run.

The main question is of course whether we can also evaluate the final formulas, which combine knowledge and strategies. The next examples show translations for the last two example formulas which mix epistemic and strategic operators.

```
-- <<A>> KB a1
-- [] focus=none & stp=0 ->   focus':=actA
-- [] focus=none & stp=1 ->   focus':=epB
-- [] focus=none & stp=2 -> finish':=true
atl "atel1" <<actionA>> F (finish&(d=one|e=one));
-- passed

-- KA <<A>> KB a1
-- [] focus=none & stp=0 ->   focus':=epA
-- [] focus=none & stp=1 ->   focus':=actA
-- [] focus=none & stp=2 ->   focus':=epB
-- [] focus=none & stp=3 -> finish':=true
atl "atel2" <<actionA>> F (finish&(d=one|e=one));
-- failed
```

The formulas given in this section have been simplified compared to the outcome of the procedure defined in section 5. In our example, knowledge monotonically increases. For instance in the formula $K_A\langle\langle A\rangle\rangle \diamond K_B a_1$ the environment has not been explicitly introduced in the translation of the strategic operator. In this example that is not necessary since knowledge only increases in this scenario. Another simplification used is that we have replaced $\langle\langle\Gamma_1\rangle\rangle \diamond (l_1 \wedge \langle\langle\Gamma_2\rangle\rangle \diamond (l_2 \wedge \phi))$ in certain cases by $\langle\langle\Gamma_1 \cup \Gamma_2\rangle\rangle \diamond (l_2 \wedge \phi))$. This is equivalent if all turns of Γ_1 precede the turns of Γ_2, which is the case in our example. In an automated translation process there is no need for these simplifications, but we hope that the resulting formulas are easier to read when they are simplified.

Again, the results are correct. We hope this example has convinced the reader that explicit encoding of epistemic relations is a feasible method for model checking ATEL.

7 Conclusion

We have demonstrated a technique for model checking ATEL, a logic for expressing properties involving both knowledge and strategies, using an ATL model checker. The approach can be used for arbitrary ATEL formulas on any turn-based acyclic transition system. Since ATL model checkers already exist, we believe this is a useful result of immediate practical value.

Application of the method involves explicit encoding of transitions corresponding to epistemic relations. The example given shows how to derive such code for a small turn-based acyclic system.

An interesting question for future research is whether the approach can be extended to incorporate multi-agent notions of knowledge, such as common and distributed knowledge [6].

References

[1] Alur, R., L. de Alfaro, T. A. Henzinger, S. C. Krishnan, F. Y. C. Mang, S. Qadeer, S. K. Rajamani and S. Taşiran, MOCHA *user manual*, University of Berkeley Report (2000).

[2] Alur, R., T. A. Henzinger and O. Kupferman, *Alternating-time temporal logic*, in: *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, Florida, 1997, pp. 100–109.

[3] Alur, R., T. A. Henzinger and O. Kupferman, *Alternating-time temporal logic*, Journal of the ACM **49** (2002), pp. 672–713.

[4] Blackburn, P., M. de Rijke and Y. Venema, "Modal Logic," Cambridge University Press: Cambridge, England, 2001.

[5] Clarke, E. M., O. Grumberg and D. A. Peled, "Model Checking," The MIT Press: Cambridge, MA, 2000.

[6] Fagin, R., J. Y. Halpern, Y. Moses and M. Y. Vardi, "Reasoning about knowledge," The MIT Press: Cambridge, MA, 1995.

[7] Jamroga, W. and W. van der Hoek, *Some remarks on alternating-time temporal epistemic logic* (2003).

[8] Jonker, G., *Feasible strategies in alternating-time temporal epistemic logic* (2003), universiteit Utrecht Master Thesis.

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

[10] van der Hoek, W. and M. Wooldridge, *Model checking knowledge and time*, in: D. Bošnački and S. Leue, editors, *Model Checking Software, Proceedings of SPIN 2002 (LNCS Volume 2318)* (2002), pp. 95–111.

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

[12] van Ditmarsch, H. P., "Knowledge Games," Ph.D. thesis, University of Groningen, Groningen (2000).

[13] van Ditmarsch, H. P., *The russian cards problem*, Studia Logica **75** (2003), pp. 31–62.