Verifying Multi-agent Systems via Unbounded Model Checking*

M. Kacprzak[1], A. Lomuscio[2], T. Łasica[3], W. Penczek[3,4], and M. Szreter**[3]

[1] Białystok University of Technology
Institute of Mathematics and Physics
15-351 Białystok, ul. Wiejska 45A, Poland
email: mdkacprzak@wp.pl
[2] Department of Computer Science
King's College London, London WC2R 2LS, United Kingdom
email: alessio@dcs.kcl.ac.uk
[3] Institute of Computer Science, PAS
01-237 Warsaw, ul. Ordona 21, Poland
email: {tlasica,penczek,mszreter}@ipipan.waw.pl
[4] Podlasie Academy
Institute of Informatics, Siedlce, Poland

Abstract. We present an approach to the problem of verification of epistemic properties in multi-agent systems by means of symbolic model checking. In particular, it is shown how to extend the technique of unbounded model checking from a purely temporal setting to a temporal-epistemic one. In order to achieve this, we base our discussion on interpreted systems semantics, a popular semantics used in multi-agent systems literature. We give details of the technique and show how it can be applied to the well known train, gate and controller problem.
Keywords: model checking, unbounded model checking, multi-agent systems

1 Introduction

Verification of reactive systems by means of model-checking techniques [3] is now a well-established area of research. In this paradigm one typically models a system S in terms of automata (or by a similar transition-based formalism), builds an implementation P_S of the system by means of a model-checker friendly language such as the input for SMV or PROMELA, and finally uses a model-checker such as SMV or SPIN to verify some temporal property ϕ the system: $M_P \models \phi$, where M_P is a temporal model representing the executions of P_S. As it is well known, there are intrinsic difficulties with the naive approach of

* The authors acknowledge support from the Polish National Committee for Scientific Research (grant No 4T11C 01325, a special grant supporting ALFEBIITE), the Nuffield Foundation (grant NAL/00690/G), and EPSRC (GR/S49353/01).
** M. Szreter acknowledges support from the US Navy via grant N00014-04-1-4063 issued by the Office of Naval Research International Field Office.

performing this operation on an explicit representation of the states, and refinements of symbolic techniques (based on OBDD's, and SAT [1] translations) are being investigated to overcome these hurdles. Formal results and corresponding applications now allow for the verification of complex systems that generate tens of thousands of states.

The field of multi-agent systems (MAS) has also recently become interested in the problem of verifying complex systems. In MAS the emphasis is on the autonomy, and rationality of the components, or agents [22]. In this area, modal logics representing concepts such as knowledge, beliefs, intentions, norms, and the temporal evolution of these are used to specify high level properties of the agents. Since these modalities are given interpretations that are different from the ones of the standard temporal operators, it is not straightforward to apply existing model checking tools developed for standard *Linear Temporal Logic* (LTL) (or *Computation Tree Logic*, CTL) temporal logic to the specification of MAS. One further problem is the fact that the modalities that are of interest are often not given a precise interpretation in terms of the computational states of the system, but simply interpreted on classes of Kripke models that guarantee (via frame-correspondence) that some intuitive properties of the system are preserved[1]. This makes it hard to use the semantics to model any actual computation performed by the system [21]. For the case of knowledge, the semantics of interpreted systems [8], popularized by Halpern and colleagues in the 90's, can be used to give an interpretation to the modalities that maintains the traditional S5 properties, while, at the same time, is appropriate for model checking [9]. Indeed, a considerable amount of literature now exists on the application of interpreted systems and epistemic logic to the application areas of security, modelling of synchronous, asynchronous systems, digital rights, etc. It is fair to say that this area constitutes the most thoroughly explored, and technically advanced sub-discipline among the formal studies of multi-agent systems available at the moment.

1.1 State of the art and related literature

The recent developments in the area of model checking MAS can broadly be divided into streams: in the first category standard predicates are used to interpret the various intensional notions and these are paired with standard model checking techniques based on temporal logic. Following this line is for example [23] and related papers. In the other category we can place techniques that make a genuine attempt at extending the model checking techniques by adding other operators. Works along these lines include [19, 20, 12, 17, 16, 15, 14, 10].

In [19] local propositions are used to translate knowledge modalities on LTL structures. Once this process is done, the result can be fed into a SPIN model

[1] For example, in epistemic logic it is customary to use equivalence models to interpret a knowledge modality K so that it inherits the properties of the logical systems S5 [2]; in particular axioms T, 4, and 5 (which are considered to be intuitively correct for knowledge) result valid.

checker. Unfortunately, in this approach local propositions need to be computed by the user.

These works were preceded by [12], where van der Meyden and Shilov presented theoretical properties of the model checking problems for epistemic linear temporal logics for interpreted systems with perfect recall. In particular, it was shown that the problem of checking a language that includes "until" and "common knowledge" on perfect recall systems is undecidable, and decidable fragments were identified.

In [17, 16, 15] an extension of standard temporal verification via model checking on obdd's to epistemic and deontic operators is presented and studied.

In [14, 10] an extension of the method of bounded model checking (one of the main SAT-based techniques) to CTLK a language comprising both CTL and knowledge operators, was defined, implemented, and evaluated. While preliminary results appear largely positive, any bounded model checking algorithm is mostly of use when the task is either to check whether a universal CTLK formula is actually false on a model, or to check that an existential CTLK formula is valid. This is a severe limitation in MAS as it turns out that many of the most interesting properties one is interested in checking actually involve universal formulas. For example, in a security setting one may want to check whether it is true that forever in the future a particular secret, perhaps a key, is mutually known by two participants.

1.2 Aim of this paper

The aim of this paper is to contribute to the line of SAT-based techniques, by overcoming the intrinsic limitation of any bounded model checking algorithm, and provide a method for model checking the full language of CTLK. The SAT-based method we introduce and discuss here is an extension to knowledge and time of a technique introduced by McMillan [11] called *unbounded model checking (UMC)*. A byproduct of the work presented here is the definition of fixed point semantics for a logic $\mathrm{CTL_pK}$, which extends CTLK by past operators.

Like any SAT-based method, UMC consists in translating the model checking problem of what is in this case a $\mathrm{CTL_pK}$ formula into the problem of satisfiability of a propositional formula. UMC exploits the characterization of the basic modalities in terms of *Quantified Boolean Formulas* (QBF), and the algorithms that translate QBF and fixed point equations over QBF into propositional formulas. In order to adapt UMC for checking $\mathrm{CTL_pK}$, we use three algorithms. The first one, implemented by the procedure *forall* [11] (based on the Davis-Putnam-Logemann-Loveland approach [4]) eliminates the universal quantifier from a QBF formula representing a $\mathrm{CTL_pK}$ formula, and returns the result in *conjunctive normal form* (CNF). The remaining algorithms, implemented by the procedures *gfp* and *lfp* calculate the greatest and the least fixed points for the modal formulas in use here. Ultimately, the technique allows for a $\mathrm{CTL_pK}$ for-

mula α to be translated into a propositional formula $[\alpha](w)$ [2] in CNF, which characterizes all the states of the model, where α holds.

For the case of CTL it was shown by McMillan [11] that model checking via UMC can be exponentially more efficient than approaches based on BDD's in two situations:

- whenever the resulting fixed points have compact representations in CNF, but not via BDD's;
- whenever the SAT-based image computation step proves to be faster than the BDD-based one.

Although we do not prove it here, we expect a similar increase in efficiency for model checking of $\mathrm{CTL_pK}$ over interpreted systems.

The rest of the paper is structured in the following manner. Section 2 introduces interpreted systems semantics, the semantics on which we ground our investigation. The logic $\mathrm{CTL_pK}$ is defined in Section 3. Section 4 summarize the basic definitions that we need for CNF and QBF formulas, and fixes the notation we use throughout the paper. A fixed point characterization of $\mathrm{CTL_pK}$ formulas is presented in Section 5. The main idea of symbolic model checking $\mathrm{CTL_pK}$ is described in section 6, where algorithms for computing propositional formulas equivalent to $\mathrm{CTL_pK}$ formulas are also given. Two examples on the use of the algorithms of this paper are given in Section 7. Preliminary experimental results are shown in Section 8, whereas conclusions are given in Section 9.

2 Interpreted systems semantics

Any transition-based semantics allows for the representation of temporal flows of time by means of the successor relation. For example, UMC for CTL uses plain Kripke models [11]. To work on a temporal epistemic language, we need to consider a semantics that allows for an automatic representation of the epistemic relations between computational states [21]. The mainstream semantics that allows to do so is the one of interpreted systems [8].

Interpreted systems can be succinctly defined as follows (we refer to [8] for more details). Assume a set of agents $A = \{1, \ldots, n\}$, a set of local states L_i and possible actions Act_i for each agent $i \in A$, and a set L_e and Act_e of local states and actions for the environment. The set of possible global states for the system is defined as $G = L_1 \times \ldots \times L_n \times L_e$, where each element $(l_1, \ldots, l_n, l_e)$ of G represents a computational state for the whole system (note that, as it will be clear below, some states in G may actually be never reached by any computation of the system). Further assume a set of protocols $P_i : L_i \to 2^{Act_i}$, for $i = 1, \ldots, n$, representing the functioning behaviour of every agent, and a function $P_e : L_e \to 2^{Act_e}$ for the environment. We can model the computation taking place in the system by means of a transition function $t : G \times Act \to G$,

[2] Note that w is a vector of propositional variables used to encode the states of the model.

where $Act \subseteq Act_1 \times \ldots \times Act_n \times Act_e$ is the set of joint actions. Intuitively, given an initial state ι, the sets of protocols, and the transition function, we can build a (possibly infinite) structure that represents all the possible computations of the system. Many representations can be given to this structure; since in this paper we are only concerned with temporal epistemic properties, we shall find the following to be a useful one.

Definition 1 (Models). *Given a set of agents $A = \{1, \ldots, n\}$, a temporal epistemic* model *(or simply a* model*) is a pair $\mathrm{M} = (\mathcal{K}, \mathcal{V})$ with $\mathcal{K} = (G, W, T, \sim_1, \ldots, \sim_n, \iota)$, where*

- *G is the set of the* global states *for the system (henceforth called simply* states*);*
- *$T \subseteq G \times G$ is a total binary (successor) relation on G;*
- *W is a set of* reachable global states *from ι, i.e., $W = \{s \in G \mid (\iota, s) \in T^*\}$*[3]*,*
- *$\sim_i \subseteq G \times G$ $(i \in A)$ is an* epistemic accessibility relation *for each agent $i \in A$ defined by $s \sim_i s'$ iff $l_i(s') = l_i(s)$, where the function $l_i : G \longrightarrow L_i$ returns the local state of agent i from a global state s; obviously $\sim_i$ is an equivalence relation,*
- *$\iota \in W$ is the* initial state*;*
- *$\mathcal{V} : G \longrightarrow 2^{\mathcal{PV}_K}$ is a* valuation function *for a set of propositional variables $\mathcal{PV}_K$ such that* **true** *$\in \mathcal{V}(s)$ for all $s \in G$. $\mathcal{V}$ assigns to each state a set of propositional variables that are assumed to be true at that state.*

Note that in the definition above we include both all possible states and the subset of reachable states. The reason for this follows from having past modalities in the language (see the next section), which are defined over any possible global state so that a simple fixed point semantics for them can be given. Still, note that, if required, it is possible to restrict the range of the past modalities to reachable states only by insisting that the target state is itself reachable from the initial state.

By $|\mathrm{M}|$ we denote the number of states of M, by $\mathbb{N} = \{0, 1, 2, \ldots\}$ the set of natural numbers and by $\mathbb{N}_+ = \{1, 2, \ldots\}$ the set of positive natural numbers.

Epistemic relations. When we consider a group of agents, we are often interested in situations in which *everyone in the group* knows a fact α. In addition to this it is sometimes useful to consider other kinds of group knowledge. One of these is the one of *common knowledge*. A group of agents has common knowledge about α if everyone knows that α, and everyone knows that everyone knows α, and everyone knows that everyone knows that everyone knows that α, and so on. For example common knowledge is achieved following information broadcasting with no faults. A different notion is the one of *distributed knowledge* (sometimes referred to as "implicit knowledge", or "wise-man" knowledge). A fact α is distributed knowledge in a group of agents if it could be inferred by pooling together the information the agents have. We refer to [8] for an introduction to these concepts.

[3] T^* denotes the reflexive and transitive closure of T.

Let $\Gamma \subseteq A$. Given the epistemic relations for the agents in Γ, the union of Γ's accessibility relations defines the epistemic relation corresponding to the modality of everybody knows: $\sim_\Gamma^E = \bigcup_{i\in\Gamma} \sim_i$. $\sim_\Gamma^C$ denotes the transitive closure of $\sim_\Gamma^E$, and corresponds to the relation used to interpret the modality of common knowledge. Notice that from reflexivity of $\sim_\Gamma^E$ follows that $\sim_\Gamma^C$ is, in fact, the transitive and reflexive closure of $\sim_\Gamma^E$. The relation used to interpret the modality of distributed knowledge is given by taking the intersection of the relations corresponding to the agents in Γ.

Computations. A *computation* in M is a possibly infinite sequence of states $\pi = (s_0, s_1, \ldots)$ such that $(s_i, s_{i+1}) \in T$ for each $i \in \mathbb{N}$. Specifically, we assume that $(s_i, s_{i+1}) \in T$ iff $s_{i+1} = t(s_i, act_i)$, i.e., s_{i+1} is the result of applying the transition function t to the global state s_i, and a joint action act_i. All the components of act_i are prescribed by the corresponding protocols P_j for the agents at s_i. In the following we abstract from the transition function, the actions, and the protocols, and simply use T, but it should be clear that this is uniquely determined by the interpreted system under consideration. Indeed, these are given explicitly in the example in the last section of this paper. In interpreted systems terminology a computation is a *part* of a run; note that we do not require s_0 to be an initial state. For a computation $\pi = (s_0, s_1, \ldots)$, let $\pi(k) = s_k$, and $\pi_k = (s_0, \ldots, s_k)$, for each $k \in \mathbb{N}$. By $\Pi(s)$ we denote the set of all the infinite computations starting at s in M.

3 Computation Tree Logic of Knowledge with Past ($\mathbf{CTL_pK}$)

Interpreted systems are traditionally used to give a semantics to an epistemic language enriched with temporal connectives based on linear time [8]. Here we use *Computation Tree Logic* (CTL) by Emerson and Clarke [7] as our basic temporal language and add an epistemic and past component to it. We call the resulting logic *Computation Tree Logic of Knowledge with Past* ($\mathrm{CTL_pK}$).

Definition 2 (Syntax of $\mathrm{CTL_pK}$). *Let $\mathcal{PV}_K$ be a set of propositional variables containing the symbol* **true**. *The set of $\mathrm{CTL_pK}$ formulas $\mathcal{FORM}$ is defined inductively by using the following rules only:*

- *every member p of $\mathcal{PV}_K$ is a formula,*
- *if α and β are formulas, then so are $\neg\alpha$, $\alpha \wedge \beta$ and $\alpha \vee \beta$,*
- *if α and β are formulas, then so are $\mathrm{AX}\alpha$, $\mathrm{AG}\alpha$, and $\mathrm{A}(\alpha \mathrm{U} \beta)$,*
- *if α is formula, then so are $\mathrm{AY}\alpha$ and $\mathrm{AH}\alpha$,*
- *if α is formula, then so is $\mathrm{K}_i\alpha$, for $i \in A$,*
- *if α is formula, then so are $\mathrm{D}_\Gamma\alpha$, $C_\Gamma\alpha$, and $\mathrm{E}_\Gamma\alpha$, for $\Gamma \subseteq A$.*

The other modalities are defined by duality as follows:

– $\mathrm{EF}\alpha \stackrel{def}{=} \neg\mathrm{AG}\neg\alpha$, $\mathrm{EP}\alpha \stackrel{def}{=} \neg\mathrm{AH}\neg\alpha$, $\mathrm{EZ}\alpha \stackrel{def}{=} \neg\mathrm{AZ}\neg\alpha$, for $\mathrm{Z} \in \{X, Y\}$,

– $\overline{\mathrm{K}}_i\alpha \stackrel{def}{=} \neg\mathrm{K}_i\neg\alpha$, $\overline{\mathrm{D}}_\Gamma\alpha \stackrel{def}{=} \neg\mathrm{D}_\Gamma\neg\alpha$, $\overline{C}_\Gamma\alpha \stackrel{def}{=} \neg C_\Gamma\neg\alpha$, $\overline{\mathrm{E}}_\Gamma\alpha \stackrel{def}{=} \neg\mathrm{E}_\Gamma\neg\alpha$.

Moreover, $\alpha \Rightarrow \beta \stackrel{def}{=} \neg\alpha\vee\beta$, $\alpha \Leftrightarrow \beta \stackrel{def}{=} (\alpha \Rightarrow \beta)\wedge(\beta \Rightarrow \alpha)$, and $\mathbf{false} \stackrel{def}{=} \neg\mathbf{true}$. We omit the subscript Γ for the epistemic modalities if $\Gamma = A$, i.e., Γ is the set of all the agents. As customary X, G stand for respectively "at the next step", and "forever in the future". Y, H are their past counterparts "at the previous step", and "forever in the past". The *Until* operator U, precisely $\alpha\mathrm{U}\beta$, expresses that β occurs eventually and α holds continuously until then.

Definition 3 (Interpretation of $\mathrm{CTL_pK}$). *Let $\mathrm{M} = (\mathcal{K}, \mathcal{V})$ be a model with $\mathcal{K} = (G, W, T, \sim_1, \ldots, \sim_n, \iota)$, $s \in G$ a state, π a computation, and α, β formulas of $\mathrm{CTL_pK}$. $\mathrm{M}, s \models \alpha$ denotes that α is true at the state s in the model M. M is omitted, if it is implicitly understood. The relation $\models$ is defined inductively as follows:*

$s \models p$ *iff* $p \in \mathcal{V}(s)$,
$s \models \neg\alpha$ *iff* $s \not\models \alpha$,
$s \models \alpha \vee \beta$ *iff* $s \models \alpha$ *or* $s \models \beta$,
$s \models \alpha \wedge \beta$ *iff* $s \models \alpha$ *and* $s \models \beta$,
$s \models \mathrm{AX}\alpha$ *iff* $\forall\pi \in \Pi(s)\ \pi(1) \models \alpha$,
$s \models \mathrm{AG}\alpha$ *iff* $\forall\pi \in \Pi(s)\ \forall_{m\geq 0}\ \pi(m) \models \alpha$,
$s \models \mathrm{A}(\alpha\mathrm{U}\beta)$ *iff* $\forall\pi \in \Pi(s)\ (\exists_{m\geq 0}\ [\pi(m) \models \beta$ *and* $\forall_{j<m}\ \pi(j) \models \alpha])$,
$s \models \mathrm{AY}\alpha$ *iff* $\forall s' \in G\ (if\ (s', s) \in T,\ then\ s' \models \alpha)$,
$s \models \mathrm{AH}\alpha$ *iff* $\forall s' \in G\ (if\ (s', s) \in T^*,\ then\ s' \models \alpha)$,
$s \models \mathrm{K}_i\alpha$ *iff* $\forall s' \in W\ (if\ s \sim_i s',\ then\ s' \models \alpha)$,
$s \models \mathrm{D}_\Gamma\alpha$ *iff* $\forall s' \in W\ (if\ s \sim_\Gamma^D s',\ then\ s' \models \alpha)$,
$s \models \mathrm{E}_\Gamma\alpha$ *iff* $\forall s' \in W\ (if\ s \sim_\Gamma^E s',\ then\ s' \models \alpha)$,
$s \models C_\Gamma\alpha$ *iff* $\forall s' \in W\ (if\ s \sim_\Gamma^C s',\ then\ s' \models \alpha)$.

Definition 4. (Validity) *A $\mathrm{CTL_pK}$ formula φ is valid in* M *(denoted* $\mathrm{M} \models \varphi$*) iff* $\mathrm{M}, \iota \models \varphi$*, i.e., φ is true at the initial state of the model M.*

Notice that the past component of $\mathrm{CTL_pK}$ does not contain the modality *Since*, which is a past counterpart of the modality *Until* denoted by *U*. Extending the logic by *Since* is possible, but complicates the semantics, so this is not discussed in this paper.

4 Formulas in Conjunctive Normal Form and Quantified Boolean Formulas

In this section, we shortly describe Davis-Putnam-Logemann-Loveland approach [4] to checking satisfiability of formulas in conjunctive normal form (CNF), and show how to construct a CNF formula that is unsatisfiable exactly when a propositional formula α is valid. Having done so, we apply these two methods to compute a propositional formula equivalent to the quantified boolean formula $\forall v.\alpha$, where v is a vector of propositions. In order to do this we first give some basic

definitions. The formalism in this section is from [11] and is reported here for completeness.

Let $\mathcal{PV}$ be a finite set of propositional variables. A *literal* is a propositional variable $p \in \mathcal{PV}$ or the negation of one: $\neg p, p \in \mathcal{PV}$. A *clause* is a disjunction of a set of zero or more literals $l[1] \vee \ldots \vee l[n]$. A disjunction of zero literals is taken to mean the constant **false**. A formula is in a *conjunctive normal form* (CNF) if it is a conjunction of a set of zero or more clauses $c[1] \wedge \ldots \wedge c[n]$. A conjunction of zero clauses is taken to mean the constant **true**. An *assignment* is a partial function from $\mathcal{PV}$ to $\{\mathbf{true}, \mathbf{false}\}$. An assignment is said to be *total* when its domain is $\mathcal{PV}$. A total assignment A is said to be *satisfying* for a formula α when $\alpha(A) = \mathbf{true}$, i.e., the value of α given by A is **true** (under the usual interpretation of the boolean connectives). We equate an assignment A with the conjunction of a set of literals, specifically the set containing $\neg p$ for all $p \in dom(A)$ such that $A(p) = \mathbf{false}$, and p for all $p \in dom(A)$ such that $A(p) = \mathbf{true}$.

For a given CNF formula α and an assignment A, an *implication graph* $\mathrm{IG}(A, \alpha)$ is a maximal directed acyclic graph (V, E), where V is a set of vertices, and E is a set of edges, such that:

- V is a set of literals,
- every literal in A is a root,
- for every vertex l not in A, the CNF formula α contains the clause $cl(l, A, \alpha) \stackrel{def}{=} l \vee \bigvee_{m \in \{l' \in V : (l', l) \in E\}} \neg m$,
- for all $p \in \mathcal{PV}$, V does not contain both p and $\neg p$.

Notice that the above conditions do not uniquely define the implication graph. We denote by A_α the assignment induced by the implication graph $\mathrm{IG}(A, \alpha)$, i.e., $A_\alpha = \bigwedge_{v \in V} v$, where V is a set of vertices of $\mathrm{IG}(A, \alpha)$. Observe that A_α is an extension of A. Furthermore, $\alpha \wedge A$ implies A_α.

Given two clauses of the form $c[1] = p \vee C_1$ and $c[2] = \neg p \vee C_2$, where C_1 and C_2 are disjunctions of literals, we say that the *resolvent* of $c[1]$ and $c[2]$ is $C_1 \vee C_2$, provided that $C_1 \vee C_2$ contains no contradictory literals, i.e., it does not contain a variable p and its negation $\neg p$. If this happens, the resolvent does not exist. Note that the resolvent of $c[1]$ and $c[2]$ is a clause that is implied by $c[1] \wedge c[2]$.

CNF formulas satisfy useful properties to check their satisfiability. Indeed, notice that a CNF formula is satisfied only when each of its clauses is satisfied individually. Thus, given a CNF formula α and an assignment A, if a clause in α has all its literals assigned value **false**, then A cannot be extended to a satisfying assignment. A clause that has all its literals assigned to value **false** is called a *conflicting* clause. We also say that a clause is in *conflict* when all of its literals are assigned the value **false** under A_α. If there exists a clause in α such that the all but one of its literals have been assigned the value **false**, then the remaining literal must be assigned the value **true** for this clause to be satisfied. In particular, in every satisfying assignment which is an extension

of the assignment A, the unassigned literal must be **true**. Such an unassigned literal is called *unit literal*, and the clause it belongs to is called a *unit clause*.

There are several algorithms for determining satisfiability of CNF formulas. Here, we use the algorithm proposed by Davis and Putnam and later modified by Davis, Logemann and Loveland [4]. The algorithm is based on the methods of *Boolean constraint propagation* (BCP) and *conflict-based learning* (CBL) and it is aimed at building a satisfying assignment for a given formula α in an incremental manner. The BCP technique is the most important part of the algorithm; it determines a logical consequence of the current assignment by building an implication graph and detecting unit clauses, and conflicting clauses. When a conflict is detected, as we mentioned above, the current assignment cannot be extended to a satisfying one. In this case, the technique of conflict-based learning is used to deduce a new clause that prevents similar conflicts from reoccurring. This new clause is called a *conflict clause* and is deduced by resolving the existing clauses using the implication graph as a guide.

The following is a generic conflict-based learning procedure that takes an assignment A, a CNF formula α, and a conflicting clause c and produces a conflict clause by repeatedly applying resolution steps until either a termination condition T is satisfied, or no further steps are possible. We elaborate on the condition T below when we discuss how the procedure *deduce* is used by the procedure *forall*.

procedure *deduce*(c, A, α),
`while` $\neg T$ `and exists` $l \in c$ `such that` $\neg l \notin A$
  `let` $c =$ `resolvent of` $cl(\neg l, A, \alpha)$ `and` c
`return` c

The resulting clause c is implied by α. Thus it can be added to α without changing its satisfiability.

In the following we show a polynomial-time algorithm that, given a propositional formula α, constructs a CNF formula which is unsatisfiable exactly when α is valid. The procedure works as follows. First, for every β subformula of the formula α (including α itself) we introduce a distinct variable l_β. If β is a propositional variable, then $l_\beta = \beta$. Next we assign a formula $\mathcal{CNF}(\beta)$ to every subformula β according to the following rules:

- if β is a variable then $\mathcal{CNF}(\beta) =$ **true**,
- if $\beta = \neg\phi$ then $\mathcal{CNF}(\beta) = \mathcal{CNF}(\phi) \wedge (l_\beta \vee l_\phi) \wedge (\neg l_\beta \vee \neg l_\phi)$,
- if $\beta = \phi \vee \varphi$ then $\mathcal{CNF}(\beta) = \mathcal{CNF}(\phi) \wedge \mathcal{CNF}(\varphi) \wedge (l_\beta \vee \neg l_\phi) \wedge (l_\beta \vee \neg l_\varphi) \wedge (\neg l_\beta \vee l_\phi \vee l_\varphi)$,
- if $\beta = \phi \wedge \varphi$ then $\mathcal{CNF}(\beta) = \mathcal{CNF}(\phi) \wedge \mathcal{CNF}(\varphi) \wedge (\neg l_\beta \vee l_\phi) \wedge (\neg l_\beta \vee l_\varphi) \wedge (l_\beta \vee \neg l_\phi \vee \neg l_\varphi)$,
- if $\beta = \phi \rightarrow \varphi$ then $\mathcal{CNF}(\beta) = \mathcal{CNF}(\phi) \wedge \mathcal{CNF}(\varphi) \wedge (l_\beta \vee l_\phi) \wedge (l_\beta \vee \neg l_\varphi) \wedge (\neg l_\beta \vee \neg l_\phi \vee l_\varphi)$.

It can be shown [11] that the formula α is valid when the CNF formula $\mathcal{CNF}(\alpha) \wedge \neg l_\alpha$ is unsatisfiable. This follows from the fact that there is a unique satisfying assignment A' of $\mathcal{CNF}(\alpha)$ consistent with A such that $A'(l_\alpha) = \alpha(A)$.

In our method, in order to have a more succinct notation for complex operations on boolean formulas, we also use *Quantified Boolean Formulas* (QBF), an extension of propositional logic by means of quantifiers ranging over propositions. In BNF: $\alpha ::= p \mid \neg\alpha \mid \alpha \wedge \alpha \mid \exists p.\alpha \mid \forall p.\alpha$. The semantics of the quantifiers is defined as follows:

- $\exists p.\alpha$ iff $\alpha(p \leftarrow \mathbf{true}) \vee \alpha(p \leftarrow \mathbf{false})$,
- $\forall p.\alpha$ iff $\alpha(p \leftarrow \mathbf{true}) \wedge \alpha(p \leftarrow \mathbf{false})$,

where $\alpha \in$ QBF, $p \in \mathcal{PV}$ and $\alpha(p \leftarrow \psi)$ denotes substitution with the formula ψ of every occurrence of the variable p in formula α.

We will use the notation $\forall v.\alpha$, where $v = (v[1], \ldots, v[m])$ is a vector of propositional variables, to denote $\forall v[1].\forall v[2] \ldots \forall v[m].\alpha$.

What is important here, is that for a given QBF formula $\forall v.\alpha$, we can construct a CNF formula equivalent to it by using the algorithm *forall* [11].

procedure *forall*(v, α), where $v = (v[1], ..., v[m])$ and α is a propositional formula
let $\phi = \mathcal{CNF}(\alpha) \wedge \neg l_\alpha$, $\chi = \mathbf{true}$, and $A = \emptyset$
repeat
  if ϕ contains **false**, return χ
  else if some c in ϕ is in conflict
    add clause *deduce*(c, A, ϕ) to ϕ
    remove some literals from A
  else if A_ϕ is total
    choose a blocking clause c'
    remove literals of form $v[i]$ or $\neg v[i]$ from c'
    add c' to ϕ and χ
  else
    choose a literal l such that $l \notin A$ and $\neg l \notin A$
    add l to A

The procedure works as follows. Initially it assumes an empty assignment A, a formula χ to be **true** and ϕ to be a CNF formula $\mathcal{CNF}(\alpha) \wedge \neg l_\alpha$. The algorithm aims at building a satisfying assignment for the formula ϕ, i.e., an assignment that falsifies α. The search for an appropriate assignment is based on the Davis-Putnam-Logemann-Loveland approach. The following three cases may happen:

- A conflict is detected, i.e., there exists a clause in ϕ such that all of its literals are false in A_ϕ. So, the assignment A can not be extended to a satisfying one. Then, the procedure *deduce* is called to generate a conflict clause, which

is added to ϕ, and the algorithm backtracks, i.e., it changes the assignment A by withdrawing one of the previously assigned literals.
- A conflict does not exist and A_ϕ is total, i.e., the satisfying assignment is obtained. In this case we generate a new clause which is false in the current assignment A_ϕ and whose complement characterizes a set of assignments falsifying the formula α. This clause is called a *blocking clause* and it must have the following properties:
  - it contains only input variables, i.e., the variables over which the input formula α is built,
  - it is false in the current assignment,
  - it is implied by $l_\alpha \wedge \mathcal{CNF}(\alpha)$.

  A blocking clause could be generated using the conflict-based learning procedure, but we require the blocking clause to contain only input variables. To do this we use an implication graph, in which all the roots are input literals. Such a graph can be generated in the following way. Let A_ϕ be a satisfying assignment for ϕ, $A' = A_\phi \downarrow V$, i.e., A' is the projection of A_ϕ onto the input variables and let $\phi' = \mathcal{CNF}(\alpha) \wedge \chi$. It is not difficult to show that $A'_{\phi'} = A_\phi$, i.e., both the graphs $\mathrm{IG}(A', \phi')$ and $\mathrm{IG}(A, \phi)$ induce the same assignments. Furthermore, the variable l_α is in conflict in $\mathrm{IG}(A', \phi')$, since ϕ contains the clause $\neg l_\alpha$. Thus, a clause $deduce(l_\alpha, A', \phi')$ is a blocking clause providing that it contains only input variables, what can be ensured by a termination condition T.

  Next, in order to quantify universally over the variables $v[1], \ldots, v[m]$, the blocking clause is deprived of the variables either of the form $v[i]$ or the negation of these. This is sufficient as the blocking clause is a formula in CNF. Then, what remains is added to the formulas ϕ and χ and the algorithm continues, i.e., again finds a satisfying assignment for ϕ.
- The first two cases do not apply. Then, the procedure makes a new assignment A by giving a value to a selected variable.

On termination, when ϕ becomes unsatisfiable, χ is a conjunction of the blocking clauses and precisely characterizes $\forall v.\alpha$.

Theorem 1. *Let α be a propositional formula and $v = (v[1], \ldots, v[m])$ be a vector of propositions, then the QBF formula $\forall v.\alpha$ is logically equivalent to the CNF formula $forall(v, \alpha)$.*

The proof of the above theorem follows from the correctness of the algorithm *forall* (see [11]).

Example 1. We illustrate in a quite detailed way (as performed by a solver) some basic operations of the procedure $forall$. To make it simple, we explain these operations for a formula in CNF. So, let $\phi = (\neg v_1) \wedge (v_1 \vee v_4 \vee \neg v_5) \wedge (\neg v_2 \vee v_3) \wedge (v_4 \vee v_5)$ and assume that $\phi = \mathcal{CNF}(\alpha) \wedge \neg l_\alpha$ for some formula α. The aim of the procedure $forall(v_1, \alpha)$ is to find a formula in CNF equivalent to $\forall v_1.\alpha$. We will only show how one blocking clause is generated and added to ϕ and χ. Notice that at the start of the procedure the assignment of v_1 is implied as

this variable is the only literal in a clause of ϕ and must be followed in order for the clause to be satisfied. Thus, we have $A = \{\neg v_1\}$. Now, the algorithm decides the assignment for another unassigned variable, say $A(v_2) = \mathbf{true}$. This implies the assignment of v_3, namely $A(v_3) = \mathbf{true}$, so that the clause $(\neg v_2 \vee v_3)$ is satisfied. Next, an assignment $A(v_4) = \mathbf{false}$ is decided, but notice that this implies both v_5 (because of the clause $(v_4 \vee v_5)$) and $\neg v_5$ (because of the clause $(v_1 \vee v_4 \vee \neg v_5)$) – *a conflict*. The implication graph is analysed (several algorithms can be applied [13]) and a *learned clause* $(v_1 \vee v_4)$ is generated and added to the working set of clauses (i.e., ϕ). Notice, that the variables v_2 and v_3 are not responsible for this conflict. The learned clause greatly reduces the number of assignments to be examined as the partial assignment $\{\neg v_1, \neg v_4\}$ is excluded from the future search irrespectively on valuations of the remaining variables. Next, the algorithm withdraws from the assignment of v_4. Notice that the learned clause implies $A(v_4) = \mathbf{true}$. Thus, a satisfying assignment that is found is $A_\varphi = \{\neg v_1, v_2, v_3, v_4, v_5\}$.

A *blocking clause* $(v_1 \vee \neg v_4)$ is generated and the literal v_1 is removed from this clause. We obtain the *blocking clause* $c' = (\neg v_4)$ and c' is added to ϕ and χ. The procedure keeps on going until ϕ does not contain $false$.

5 Fixed point characterization of $\mathrm{CTL_pK}$

In this section we show how the set of states satisfying any $\mathrm{CTL_pK}$ formula can be characterized by a fixed point of an appropriate function. We follow and adapt, when necessary, the definitions given in [3].

Let $\mathrm{M} = ((G, W, T, \sim_1, \ldots, \sim_n, \iota), \mathcal{V})$ be a model. Notice that the set 2^G of all subsets of G forms a lattice under the set inclusion ordering. Each element $G' \subseteq Q$ of the lattice can also be thought of as a *predicate* on G, where the predicate is viewed as being true for exactly the states in G'. The least element in the lattice is the empty set, which corresponds to the predicate **false**, and the greatest element in the lattice is the set G, which corresponds to **true**. A function τ mapping 2^G to 2^G is called a *predicate transformer*. A set $G' \subseteq G$ is a *fixed point* of a function $\tau : 2^G \to 2^G$ if $\tau(G') = G'$.

Whenever τ is monotonic (i.e., when $P \subseteq Q$ implies $\tau(P) \subseteq \tau(Q)$), τ has a least fixed point denoted by $\mu Z.\tau(Z)$, and a greatest fixed point, denoted by $\nu Z.\tau(Z)$. When τ is monotonic and $\bigcup$-continuous (i.e., when $P_1 \subseteq P_2 \subseteq \ldots$ implies $\tau(\bigcup_i P_i) = \bigcup_i \tau(P_i)$), then $\mu Z.\tau(Z) = \bigcup_{i \geq 0} \tau^i(\mathbf{false})$. When τ is monotonic and $\bigcap$-continuous (i.e., when $P_1 \supseteq P_2 \supseteq \ldots$ implies $\tau(\bigcap_i P_i) = \bigcap_i \tau(P_i)$), then $\nu Z.\tau(Z) = \bigcap_{i \geq 0} \tau^i(\mathbf{true})$ (see [18]).

In order to obtain fixed point characterizations of the modal operators, we identify each $\mathrm{CTL_pK}$ formula α with the set $\langle\alpha\rangle_\mathrm{M}$ of states in M at which this formula is true, formally $\langle\alpha\rangle_\mathrm{M} = \{s \in G \mid M, s \models \alpha\}$. If M is clear from the context we omit the subscript M. Furthermore, we define functions $\mathrm{AX}, \mathrm{AY}, \mathrm{K}_i, \mathrm{E}_\Gamma, \mathrm{D}_\Gamma$ from 2^G to 2^G as follows:

- $\mathrm{AX}(Z) = \{s \in G \mid \text{for every } s' \in G \text{ if } (s, s') \in T, \text{ then } s' \in Z\}$,
- $\mathrm{AY}(Z) = \{s \in G \mid \text{for every } s' \in G \text{ if } (s', s) \in T, \text{ then } s' \in Z\}$,

- $\mathrm{K}_i(Z) = \{s \in G \mid \text{for every } s' \in G \text{ if } (\iota, s') \in T^* \text{ and } s \sim s', \text{ then } s' \in Z\}$,
- $\mathrm{E}_\Gamma(Z) = \{s \in G \mid \text{for every } s' \in G \text{ if } (\iota, s') \in T^* \text{ and } s \sim^E_\Gamma s', \text{ then } s' \in Z\}$,
- $\mathrm{D}_\Gamma(Z) = \{s \in G \mid \text{for every } s' \in G \text{ if } (\iota, s') \in T^* \text{ and } s \sim^D_\Gamma s', \text{ then } s' \in Z\}$.

Observe that $\langle O\alpha \rangle = O(\langle \alpha \rangle)$, for $O \in \{\mathrm{AX}, \mathrm{AY}, \mathrm{K}_i, \mathrm{E}_\Gamma, D_\Gamma\}$. Then, the following temporal and epistemic operators may be characterized as the least or the greatest fixed point of an appropriate monotonic ($\bigcap$-continuous or $\bigcup$-continuous) predicate transformer.

- $\langle \mathrm{AG}\alpha \rangle = \nu Z.\langle \alpha \rangle \cap \mathrm{AX}(Z)$,
- $\langle \mathrm{A}(\alpha \mathrm{U} \beta) \rangle = \mu Z.\langle \beta \rangle \cup (\langle \alpha \rangle \cap \mathrm{AX}(Z))$,
- $\langle \mathrm{AH}\alpha \rangle = \nu Z.\langle \alpha \rangle \cap \mathrm{AY}(Z)$,
- $\langle \mathcal{C}_\Gamma \alpha \rangle = \nu Z.\mathrm{E}_\Gamma(Z \cap \langle \alpha \rangle)$

The first three equations are standard (see [6], [3]), whereas the fourth one is defined analogously taking account that $\sim^C_\Gamma$ is the transitive, and reflexive closure of $\sim^E_\Gamma$.

6 Symbolic model checking on $\mathrm{CTL_pK}$

Let $M = (\mathcal{K}, \mathcal{V})$ with $\mathcal{K} = (G, W, T, \sim_1, ..., \sim_n, \iota)$. Recall that the set of global states $G = \times_{i=1}^n L_i$ is the Cartesian product of the set of local states (without loss of generality we treat the environment as one of the agents).

We assume $L_i \subseteq \{0,1\}^{n_i}$, where $n_i = \lceil \log_2(|L_i|) \rceil$ and let $n_1 + \ldots + n_n = m$, i.e., every local state is represented by a sequence consisting of 0's and 1's. Moreover, let D_i be a set of the indexes of the bits of the local states of each agent i of the global states, i.e., $D_1 = \{1, \ldots, n_1\}, \ldots, D_n = \{m - n_n + 1, \ldots, m\}$.

Let $\mathcal{PV}$ be a set of fresh propositional variables such that $\mathcal{PV} \cap \mathcal{PV}_K = \emptyset$, $\mathrm{F}_{\mathcal{PV}}$ be a set of propositional formulas over $\mathcal{PV}$, and $lit : \{0,1\} \times \mathcal{PV} \to \mathrm{F}_{\mathcal{PV}}$ be a function defined as follows: $lit(0,p) = \neg p$ and $lit(1,p) = p$. Furthermore, let $w = (w[1], \ldots, w[m])$, where $w[i] \in \mathcal{PV}$ for each $i = 1, \ldots, m$, be a *global state variable*. We use elements of G as valuations[4] of global state variables in formulas of $\mathrm{F}_{\mathcal{PV}}$. For example $w[1] \wedge w[2]$ evaluates to *true* for the valuation $q = (1, \ldots, 1)$, and it evaluates to *false* for the valuation $q = (0, \ldots, 0)$.

Now, the idea consists in using propositional formulas of $F_{\mathcal{PV}}$ to encode sets of states of G. For example, the formula $w[1] \wedge \ldots \wedge w[m]$ encodes the state represented by $(1, \ldots, 1)$, whereas the formula $w[1]$ encodes all the states, the first bit of which is equal to 1.

Next, the following propositional formulas are defined:

- $I_s(w) := \bigwedge_{i=1}^m lit(s_i, w[i])$.
  This formula encodes the state $s = (s_1, \ldots, s_m)$ of the model, i.e., $s_i = 1$ is encoded by $w[i]$, and $s_i = 0$ is encoded by $\neg w[i]$.
- $H(w,v) := \bigwedge_{i=1}^m w[i] \Leftrightarrow v[i]$.
  This formula represents logical equivalence between global state encodings, representing the fact that they represent the same state.

[4] We identify 1 with *true* and 0 with *false*.

- $T(w,v)$ is a formula, which is true for a valuation $(s_1,\ldots,s_m)$ of $(w[1],\ldots,w[m])$ and a valuation $(s'_1,\ldots,s'_m)$ of $(v[1],\ldots,v[m])$ iff $((s_1,\ldots,s_m),(s'_1,\ldots,s'_m)) \in T$.

Our aim is to translate $\mathrm{CTL_pK}$ formulas into propositional formulas. Specifically, for a given $\mathrm{CTL_pK}$ formula β we compute a corresponding propositional formula $[\beta](w)$, which encodes those states of the system that satisfy the formula. Operationally, we work outwards from the most nested subformulas, i.e., the atoms. In other words, to compute $[\mathrm{O}\alpha](w)$, where O is a modality, we work under the assumption of already having computed $[\alpha](w)$. To calculate the actual translations we use either the fixed point or the QBF characterization of $\mathrm{CTL_pK}$ formulas. For example, the formula $[\mathrm{AX}\alpha](w)$ is equivalent to the QBF formula $\forall v.(T(w,v) \Rightarrow [\alpha](v))$. We can use similar equivalences for formulas $\mathrm{AY}\alpha, \mathrm{K}_i\alpha, \mathrm{D}_\Gamma\alpha, \mathrm{E}_\Gamma\alpha$. More specifically, we use the following three basic algorithms. The first one, implemented by the procedure *forall*, is used for formulas $\mathrm{O}\alpha$ such that $\mathrm{O} \in \{\mathrm{AX}, \mathrm{AY}, \mathrm{K}_i, \mathrm{D}_\Gamma, \mathrm{E}_\Gamma\}$. This procedure eliminates the universal quantifier from a QBF formula representing a $\mathrm{CTL_pK}$ formula, and returns the result in a conjunctive normal form. The second algorithm, implemented by the procedure gfp_O, is applied to formulas $\mathrm{O}\alpha$ such that $\mathrm{O} \in \{\mathrm{AG}, \mathrm{AH}, \mathcal{C}_\Gamma\}$. This procedure computes the greatest fixed point. For the formulas of the form $\mathrm{A}(\alpha\mathrm{U}\beta)$ we use a third procedure, called lfp_{AU}, which computes the least fixed point. In so doing, given a formula β we obtain a propositional formula $[\beta](w)$ such that β is valid in the model M iff the conjunction $[\beta](w) \wedge I_\iota(w)$ is satisfiable, i.e., $\iota \in \langle\beta\rangle$. Below, we formalize the above discussion.

Definition 5 (Translation for UMC**).** *Given a* $\mathrm{CTL_pK}$ *formula* φ*, the propositional translation* $[\varphi](w)$ *is inductively defined as follows:*

- $[p](w) := \bigvee_{s\in\langle p\rangle} I_s(w)$, *for* $p \in \mathcal{PV}_\mathcal{K}$,
- $[\neg\alpha](w) := \neg[\alpha](w)$,
- $[\alpha \wedge \beta](w) := [\alpha](w) \wedge [\beta](w)$,
- $[\alpha \vee \beta](w) := [\alpha](w) \vee [\beta](w)$,
- $[\mathrm{AX}\alpha](w) := \mathit{forall}(v, (T(w,v) \Rightarrow [\alpha](v)))$,
- $[\mathrm{AY}\alpha](w) := \mathit{forall}(v, (T(v,w) \Rightarrow [\alpha](v)))$,
- $[\mathrm{K}_i\alpha](w) := \mathit{forall}(v, ((H_i(w,v) \wedge \neg\ \mathit{gfp}_{AH}(\neg I_\iota(v))) \Rightarrow [\alpha](v)))$,
- $[\mathrm{D}_\Gamma\alpha](w) := \mathit{forall}(v, ((\bigwedge_{i\in\Gamma} H_i(w,v) \wedge \neg\ \mathit{gfp}_{AH}(\neg I_\iota(v))) \Rightarrow [\alpha](v)))$,
- $[\mathrm{E}_\Gamma\alpha](w) := \mathit{forall}(v, ((\bigvee_{i\in\Gamma} H_i(w,v) \wedge \neg\ \mathit{gfp}_{AH}(\neg I_\iota(v))) \Rightarrow [\alpha](v)))$,
- $[\mathrm{AG}\alpha](w) := \mathit{gfp}_{AG}([\alpha](w))$,
- $[\mathrm{A}(\alpha\mathrm{U}\beta)](w) := \mathit{lfp}_{AU}([\alpha](w), [\beta](w))$,
- $[\mathrm{AH}\alpha](w) := \mathit{gfp}_{AH}([\alpha](w))$,
- $[\mathcal{C}_\Gamma\alpha](w) := \mathit{gfp}_{C_\Gamma}([\alpha](w))$.

The algorithms *gfp* and *lfp* are based on the standard procedures computing fixed points.

procedure $gfp_{AG}([\alpha](w))$, `where` α `is an` $\mathrm{CTL_pK}$ `formula`
`let` $Q(w) = [\mathbf{true}](w)$, $Z(w) = [\alpha](w)$

while $\neg(Q(w) \Rightarrow Z(w))$ is satisfiable
    let $Q(w) = Z(w)$,
    let $Z(w) = forall(v, (T(w,v) \Rightarrow Z(v))) \wedge [\alpha](w)$
return $Q(w)$

The procedure gfp_{AH} is obtained by replacing in the above $forall(v, (T(w,v) \Rightarrow Z(v)))$ with $forall(v, (T(v,w) \Rightarrow Z(v)))$.

procedure $gfp_{C_\Gamma}([\alpha](w))$, where α is an $\mathrm{CTL_pK}$ formula
let $Q(w) = [\mathbf{true}](w)$, $Z(w) = forall(v, ((\bigvee_{i\in\Gamma} H_i(w,v) \wedge \neg\ gfp_{AH}(\neg I_\iota(v))) \Rightarrow [\alpha](v)))$
while $\neg(Q(w) \Rightarrow Z(w))$ is satisfiable
    let $Q(w) = Z(w)$,
    let $Z(w) = forall(v, (\bigvee_{i\in\Gamma} H_i(w,v) \wedge \neg\ gfp_{AH}(\neg I_\iota(v)) \Rightarrow (Z(v) \wedge [\alpha](v))))$
return $Q(w)$

procedure $lfp_{AU}([\alpha](w), [\beta](w))$, where α, β are $\mathrm{CTL_pK}$ formulas
let $Q(w) = [\mathbf{false}](w)$, $Z(w) = [\beta](w)$
while $\neg(Z(w) \Rightarrow Q(w))$ is satisfiable
    let $Q(w) = Q(w) \vee Z(w)$,
    let $Z(w) = forall(v, (T(w,v) \Rightarrow Q(v))) \wedge [\alpha](w)$
return $Q(w)$

We now have all the ingredients in place to state the main result of this paper: modal satisfaction of a $\mathrm{CTL_pK}$ formula can be rephrased as propositional satisfaction of an appropriate conjunction. Note that the translation is sound and complete (details of the proof are not given here).

Theorem 2 (UMC for $\mathrm{CTL_pK}$). *Let* M *be a model and* φ *be a* $\mathrm{CTL_pK}$ *formula. Then,* $\mathrm{M} \models \varphi$ *iff* $[\varphi](w) \wedge I_\iota(w)$ *is satisfiable.*

Proof. Notice that $I_\iota(w)$ is satisfied only by the valuation $\iota = (\iota_1, \ldots, \iota_m)$ of $w = (w[1], \ldots, w[m])$. Thus $[\varphi](w) \wedge I_\iota(w)$ is satisfiable iff $[\varphi](w)$ is true for the valuation ι of w. On the other hand for a model M, $\mathrm{M} \models \varphi$ iff $\mathrm{M}, \iota \models \varphi$, i.e., $\iota \in \langle\varphi\rangle$. Hence, we have to prove that $\iota \in \langle\varphi\rangle$ iff $[\varphi](w)$ is true for the valuation ι of w. The proof is by induction on the complexity of φ. The theorem follows directly for the propositional variables. Next, assume that the hypothesis holds for all the proper sub-formulas of φ. If φ is equal to either $\neg\alpha$, $\alpha \wedge \beta$, or $\alpha \vee \beta$, then it is easy to check that the theorem holds.

For the modal formulas, let P be a set of states and $\alpha_P(w)$ a propositional formula such that $\alpha_P(w)$ is true for the valuation $s = (s_1, \ldots, s_m)$ of $w = (w[1], \ldots, w[m])$ iff $s \in P$. Note that given any P, α_P is well defined: since the set G of all states is finite, and one can take $\bigvee_{s\in P} I_s(w)$ as $\alpha_P(w)$. Consider φ to be of the following forms:

- $\varphi = \mathrm{AY}\alpha$. We will prove that $\iota \in \langle \mathrm{AY}\alpha \rangle$ iff the formula $[\mathrm{AY}\alpha](w)$ is true for the valuation ι of w.
  First we prove that:
  (*) $s \in \mathrm{AY}(P)$ iff the formula $\forall v.(T(v,w) \Rightarrow \alpha_P(v))$ is true for the valuation s of w.
  $s \in \mathrm{AY}(P)$ iff $s \in \{s' \in G |$ for every $s'' \in G$ if $(s'', s') \in T$, then $s'' \in P\}$. On the one hand, $(s'', s') \in T$ iff $T(v,w)$ is true for the valuation s' of w and the valuation s'' of v. Moreover, $s'' \in P$ iff the formula $\alpha_P(v)$ is true for the valuation s'' of v. Thus $s \in \mathrm{AY}(P)$ iff the formula $T(v,w) \Rightarrow \alpha_P(v)$ is true for the valuation s of w and every valuation s'' of v. Hence, $s \in \mathrm{AY}(P)$ iff the QBF formula $\forall v.(T(v,w) \Rightarrow \alpha_P(v))$ is true for the valuation s of w.
  Therefore, $\iota \in \langle \mathrm{AY}\alpha \rangle$ iff $\iota \in \mathrm{AY}(\langle \alpha \rangle)$ iff (by the inductive assumption and (*)) the formula $(\forall v.(T(v,w) \Rightarrow [\alpha](v)))$ is true for the valuation ι of w iff (by Theorem 1) the propositional formula $forall(v, T(v,w) \Rightarrow [\alpha](v))$ is true for the valuation ι of w iff $[\mathrm{AY}\alpha](w)$ is true for the valuation ι of w.
- $\varphi = \mathrm{AX}\alpha$. The proof is analogous to the former case.
- $\varphi = \mathrm{AH}\alpha$ We will show that $\iota \in \langle \mathrm{AH}\alpha \rangle$ iff formula $[\mathrm{AH}\alpha](w)$ is true for the valuation ι of w.
  First we prove that:
  (*) $s \in \nu Z.P \cap \mathrm{AY}(Z)$ iff the formula $gfp_{AH}(\alpha_P(w))$ is true for the valuation s of w.
  Let $\tau(Z) = P \cap \mathrm{AY}(Z)$, then $s \in \nu Z.\tau(Z)$ iff $s \in \bigcap_{i \geq 0} \tau^i(G)$ (as $s \in \bigcap_{i \geq 0} \tau^i(\mathbf{true})$). Thus, $s \in \nu Z.\tau(Z)$ iff $s \in \tau^i(G)$ for the least i such that $\tau^i(G) \subseteq \tau^{i+1}(G)$ since for every $i \geq 0$ we have $\tau^{i+1}(G) \subseteq \tau^i(G)$. On the other hand, $s \in \tau(Z)$ iff formula $\alpha_P(w) \wedge \forall v.(T(v,w) \Rightarrow \alpha_Z(v))$ is true for the valuation s of w iff (by Theorem 1) formula $\alpha_P(w) \wedge forall(v, T(v,w) \Rightarrow \alpha_Z(v))$ is true for the valuation s of w.
  Let $Z^0(w) = \alpha_P(w)$ and $Z^i(w) = \alpha_P(w) \wedge forall(v, (T(v,w) \Rightarrow Z^{i-1}(v)))$ for $i > 0$. Notice that $s \in \tau^i(G)$ iff $Z^i(w)$ is true for the valuation s of w. Moreover, $Q_i(w) = Z^{i-1}(w)$ and $Z_i(w) = Z^i(w)$ are invariants of the while-loop of the procedure $gfp_{AH}(\alpha_P(w))$. Hence on the termination, when $Q_{i_0}(w) \Rightarrow Z_{i_0}(w)$, where i_0 is the least i such that $Q_i(w) \Rightarrow Z_i(w)$, $gfp_{AH}(\alpha_P(w)) = Q_{i_0}(w)$ is a formula that is true for the valuation s of w iff $s \in \nu Z.\tau(Z)$.
  Therefore, $\iota \in \langle \mathrm{AH}\alpha \rangle$ iff $\iota \in \nu Z.\langle \alpha \rangle \cap \mathrm{AY}(Z)$ iff (by the inductive assumption and (*)) the propositional formula $gfp_{AH}([\alpha](w))$ is true for the valuation ι of w iff propositional formula $[\mathrm{AH}\alpha](w)$ is true for the valuation ι of w.
- $\varphi = \mathrm{AG}\alpha \mid \mathcal{C}_\Gamma\alpha \mid \mathrm{A}(\alpha \mathrm{U} \beta)$. The proof is analogous to the former case.
- $\varphi = \mathrm{K}_i\alpha$. In order to show that $\iota \in \langle \mathrm{K}_i\alpha \rangle$ iff formula $[\mathrm{K}_i\alpha](w)$ is true for the valuation ι of w, first we prove that:
  (*) $s \in \mathrm{K}_i(P)$ iff the formula $\forall v.(\neg gfp_{AH}(\neg I_\iota(v)) \wedge H_i(w,v) \Rightarrow \alpha_P(v))$ is true for the valuation s of w.
  To this aim we prove the following two facts:
  (**) $(\iota, s'') \in T^*$ iff $\neg gfp_{AH}(\neg I_\iota(v))$ is true for the valuation s'' of v.
  Observe that $s'' \in G \backslash \{\iota\}$ iff $\neg I_\iota(v)$ is true for the valuation s'' of v. On the other hand $(\iota, s'') \notin T^*$ iff $s'' \in \nu Z.(G \backslash \{\iota\}) \cap \mathrm{AY}(Z)$. Hence $(\iota, s'') \in T^*$ iff

$s'' \notin \nu Z.(G\backslash\{\iota\}) \cap \mathrm{AY}(Z)$ iff $gfp_{AH}(\neg I_\iota(v))$ is false for the valuation s'' of v iff $\neg gfp_{AH}(\neg I_\iota(v))$ is true for the valuation s'' of v.
(***) $s' \sim_i s''$ iff $H_i(w,v)$ is true for the valuation s' of w and the valuation s'' of v.
$s' \sim_i s''$ iff $l_i(s') = l_i(s'')$ iff $\bigwedge_{j \in D_i} s'_j = s''_j$ iff formula $\bigwedge_{j \in D_i} w[j] \Leftrightarrow v[j]$ is true for the valuation s' of w and the valuation s'' of v iff $H_i(w,v)$ is true for the valuation s' of w and the valuation s'' of v.
Thus by (**) and (***), $s \in \mathrm{K}_i(P)$ iff for the valuation s of w and every valuation s'' of v formula $\neg gfp_{AH}(\neg I_\iota(v)) \wedge H_i(w,v) \Rightarrow \alpha_P(v)$ is true iff the QBF formula $\forall v.(\neg gfp_{AH}(\neg I_\iota(v)) \wedge H_i(w,v) \Rightarrow \alpha_P(v))$ is true for the valuation s of w.
Therefore, $\iota \in \langle \mathrm{K}_i\alpha \rangle$ iff $\iota \in \mathrm{K}_i(\langle\alpha\rangle)$ iff (by the inductive assumption and (*)) the formula $\forall v.(\neg gfp_{AH}(\neg I_\iota(v)) \wedge H_i(w,v) \Rightarrow [\alpha](v))$ is true for the valuation ι of w iff (by Theorem 1) the propositional formula $forall(v, (\neg gfp_{AH}(\neg I_\iota(v)) \wedge H_i(w,v) \Rightarrow [\alpha](v)))$ is true for the valuation ι of w iff $[\mathrm{K}_i\alpha](w)$ is true for the valuation ι of w.

- $\varphi = \mathrm{D}_\Gamma\alpha \mid \mathrm{E}_\Gamma\alpha$. The proof is analogous to the former case.

6.1 Optimizations of algorithms

In our implementation we apply some optimizations to the fixed point computing algorithms described above. Precisely, we compute $[\mathrm{AG}\alpha](w)$ and $[\mathrm{AH}\alpha](w)$ by using the following *frontier set simplification method* [11]. Define the formula $(\forall v.\alpha) \downarrow \delta$, representing some propositional formula such that $\delta \wedge (\forall v.\alpha) \downarrow \delta$ is equivalent to $\delta \wedge \forall v.\alpha$. The formula $(\forall v.\alpha) \downarrow \delta$ is computed using the procedure *forall* with a slight modification. Next, we compute $[\mathrm{AG}\alpha](w)$ as the conjunction of the following sequence: $Z_1(w) = [\alpha](w)$, $Z_{i+1}(w) = (\forall v.(T(w,v) \Rightarrow Z_i(v))) \downarrow \bigwedge_{j=1}^{i} Z_j(w)$. The sequence converges when $\bigwedge_{j=1}^{i} Z_j(w) \Rightarrow forall(v, (T(w,v) \Rightarrow Z_i(v)))$, in which case $Z_{i+1}(w)$ is the constant **true**. The procedure $fssm_{AG}$ for computing $[\mathrm{AG}\alpha](w)$ is as follows.

procedure $fssm_{AG}([\alpha](w))$, `where` α `is an` $\mathrm{CTL_pK}$ `formula`
`let` $Z(w) = Q(w) = [\alpha](w)$
`while` $Z(w) \neq$ **true**
  `let` $Z(w) = (\forall v.(T(w,v) \Rightarrow Z(v))) \downarrow Q(w)$
  `let` $Q(w) = Q(w) \wedge Z(w)$
`return` $Q(w)$

The procedure $fssm_{AH}$ for computing $[\mathrm{AH}\alpha](w)$ is obtained by replacing in the above $(\forall v.(T(w,v) \Rightarrow Z(v))) \downarrow Q(w)$ with $(\forall v.(T(v,w) \Rightarrow Z(v))) \downarrow Q(w)$. Similar procedure can be obtained for computing formulas $[C_\Gamma\alpha](w)$.

7 Example of Train, Gate and Controller

In this section we exemplify the procedure above by discussing the scenario of the train controller system (adapted from [20]). The system consists of three

agents: two trains (agents 1 and 3), and a controller (agent 2). The trains, one Eastbound, the other Westbound, occupy a circular track. At one point, both tracks pass through a narrow tunnel. There is no room for both trains to be in the tunnel at the same time. Therefore the trains must avoid this to happen. There are traffic lights on both sides of the tunnel, which can be either red or green. Both trains are equipped with a signaller, that they use to send a signal when they approach the tunnel. The controller can receive signals from both trains, and controls the colour of the traffic lights. The task of the controller is to ensure that the trains are never both in the tunnel at the same time. The trains follow the traffic lights signals diligently, i.e., they stop on red.

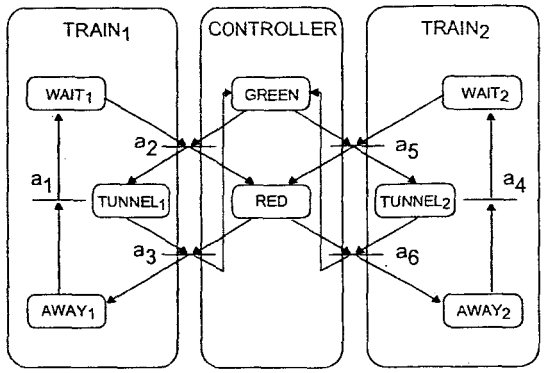

Fig. 1. The local transition structures for the two trains and the controller

We can model the example above with an interpreted system as follows. The local states for the agents are:

- $L_{train_1} = \{away_1, wait_1, tunnel_1\}$,
- $L_{controller} = \{red, green\}$,
- $L_{train_2} = \{away_2, wait_2, tunnel_2\}$.

The set of global states is defined as $G = L_{train_1} \times L_{controller} \times L_{train_2}$. Let $\iota = (away_1, green, away_2)$ be the initial state. We assume that the local states are numbered in the following way: $away_1 := 1$, $wait_1 := 2$, $tunnel_1 := 3$, $red; = 4$, $green := 5$, $away_2 := 6$, $wait_2 := 7$, $tunnel_2 := 8$ and the agents are numbered as follows: $train_1 := 1$, $controller := 2$, $train_2 := 3$. Thus we assume a set of agents A to be the set $\{1, 2, 3\}$.

Let $Act = \{a_1, ..., a_6\}$ be a set of joint actions. For $a \in Act$ we define the preconditions $pre(a)$, postconditions $post(a)$, and the set $agent(a)$ containing the numbers of the agents that may change local states by executing a.

- $pre(a_1) = \{1\}, post(a_1) = \{2\}, agent(a_1) = \{1\}$,

- $pre(a_2) = \{2,5\}, post(a_2) = \{3,4\}$, $agent(a_2) = \{1,2\}$,
- $pre(a_3) = \{3,4\}, post(a_3) = \{1,5\}$, $agent(a_3) = \{1,2\}$,
- $pre(a_4) = \{6\}, post(a_4) = \{7\}$, $agent(a_4) = \{3\}$,
- $pre(a_5) = \{5,7\}, post(a_5) = \{4,8\}$, $agent(a_5) = \{2,3\}$,
- $pre(a_6) = \{4,8\}, post(a_6) = \{5,6\}$, $agent(a_6) = \{2,3\}$.

In our formulas we use the following two propositional variables in_tunnel_1 and in_tunnel_2 such that $in\_tunnel_1 \in \mathcal{V}(s)$ iff $l_{train_1}(s) = tunnel_1$, $in\_tunnel_2 \in \mathcal{V}(s)$ iff $l_{train_2}(s) = tunnel_2$, for $s \in G$.

We now encode the local states in binary form in order to use them in the model checking technique. Given that agent $train_1$ can be in 3 different local states we shall need 2 bits to encode its state; in particular we shall take: $(0,0) = away_1$, $(1,0) = wait_1$, $(0,1) = tunnel_1$. Similarly for the agent $train_2$: $(0,0) = away_2$, $(1,0) = wait_2$, $(0,1) = tunnel_2$. The modelling of the local states of the controller requires only one bit: $(0) = green$, $(1) = red$. In view of this a global state is modelled by 5 bits. For instance the initial state $\iota = (away_1, green, away_2)$ is represented as a tuple of 5 0's. Notice that the first two bits of a global state encode the local state of agent 1, the third bit encodes the local state of agent 2, and two remaining bits encode the local state of agent 3. We represent this by taking: $D_1 = \{1,2\}$, $D_2 = \{3\}$, $D_3 = \{4,5\}$.

Let $w = (w[1], ..., w[5])$, $v = (v[1], ..., v[5])$ be two global state variables. We define the following propositional formulas over w and v:

- $I_\iota(w) := \bigwedge_{j \in D_1 \cup D_2 \cup D_3} \neg w[j]$,
  this formula encodes the initial state,
- $H_i(w,v) := \bigwedge_{j \in D_i} w[j] \Leftrightarrow v[j]$,
  the formula $H_i(w,v)$, where $i \in A$, represents logical equivalence between local states of agent i at two global states represented by variables w and v,
- $p_1(w) := \neg w[1] \wedge \neg w[2]$, $p_2(w) := w[1] \wedge \neg w[2]$, $p_3(w) := \neg w[1] \wedge w[2]$, $p_4(w) := w[3]$, $p_5(w) := \neg w[3]$, $p_6(w) := \neg w[4] \wedge \neg w[5]$, $p_7(w) := w[4] \wedge \neg w[5]$, $p_8(w) := \neg w[4] \wedge w[5]$,
  the formula $p_j(w)$, for $j = 1, \ldots, 8$, encodes a particular local state of an agent.

For $a \in Act$, let $B_a := \bigcup_{i \in A \setminus agent(a)} D_i$ be the set of the labels of the bits that are not changed by the action a, then

- $T(w,v) := \bigvee_{a \in Act} \left(\bigwedge_{j \in pre(a)} p_j(w) \wedge \bigwedge_{j \in post(a)} p_j(v) \wedge \bigwedge_{j \in B_a} (w[j] \Leftrightarrow v[j]) \right) \vee \left(\bigwedge_{a \in Act} \bigvee_{j \in pre(a)} (\neg p_j(w)) \wedge \bigwedge_{j \in D_1 \cup D_2 \cup D_3} (w[j] \Leftrightarrow v[j]) \right)$.
  Intuitively, $T(w,v)$ encodes the set of all couples of global states s and s' represented by variables w and v respectively, such that s' is reachable from s, i.e., either there exists a joint action which is available at s and s' is the result of execution a at s or there is not such an action and s' equals s. Notice that the above formula is composed of two parts. The first one encodes the transition relation of the system whereas the second one adds self-loops to all the states without successors. This is necessary in order to satisfy the assumption that T is total.

Consider now the following formulas:

- $\alpha_0 = \neg \mathrm{AX}(\neg in\_tunnel_1)$,
- $\alpha_1 = \mathrm{AG}(in\_tunnel_1 \Rightarrow \mathrm{K}_{train_1}(\neg in\_tunnel_2))$,
- $\alpha_2 = \mathrm{AG}(\neg in\_tunnel_1 \Rightarrow (\neg \mathrm{K}_{train_1} in\_tunnel_2 \wedge \neg \mathrm{K}_{train_1}(\neg in\_tunnel_2)))$,

where in_tunnel_1 (respectively in_tunnel_2) is a proposition true whenever the local state of $train_1$ is equal to $tunnel_1$ (respectively the local state of $train_2$ is equal to $tunnel_2$).

The first formula states that agent $train_1$ may at the next step be in the tunnel. The second formula expresses that when the agent $train_1$ is in the tunnel, it knows that agent $train_2$ is not in the tunnel. The third formula expresses that when agent $train_1$ is away from the tunnel, it does not know whether or not agent $train_2$ is in the tunnel.

As discussed above, the translation of propositions in_tunnel_1 and in_tunnel_2 is as follows:

- $[in\_tunnel_1](w) = \neg w[1] \wedge w[2]$,
- $[in\_tunnel_2](w) = \neg w[4] \wedge w[5]$.

Next, we show how to translate the formula α_0:
$[\alpha_0](w) = [\neg \mathrm{AX}(\neg in\_tunnel_1)](w) = \neg[AX(\neg in\_tunnel_1)](w)$.
The formula $[AX(\neg in\_tunnel_1)](w)$ is computed as follows:
$[AX(\neg in\_tunnel_1)](w) = forall(v, T(w,v) \Rightarrow [\neg in\_tunnel_1](v)) =$
$forall(v, T(w,v) \Rightarrow (\neg(\neg v[1] \wedge v[2]))) = forall(v, T(w,v) \Rightarrow (v[1] \vee \neg v[2]))$.
Consequently $[\alpha_0](w) = \neg forall(v, T(w,v) \Rightarrow (v[1] \vee \neg v[2]))$ and $[\alpha_0](w) \wedge I_\iota(w)$
$= \neg forall(v, T(w,v) \Rightarrow (v[1] \vee \neg v[2])) \wedge I_\iota(w) = ((w[1] \wedge \neg w[2] \wedge \neg w[3]) \vee (\neg w[1] \wedge$
$w[2] \wedge \neg w[3] \wedge \neg w[5]) \vee (\neg w[1] \wedge w[2] \wedge w[3] \wedge \neg w[4]) \vee (\neg w[1] \wedge w[2] \wedge \neg w[3] \wedge$
$\neg w[4] \vee w[5])) \wedge I_\iota(w) =$ **false**. Therefore α_0 is not valid in the model.
But, both the formulas α_1 and α_2 are valid in the model since
$[\alpha_1](w) \wedge I_\iota(w) = true \wedge I_\iota(w) = \neg w[1] \wedge \neg w[2] \wedge \neg w[3] \wedge \neg w[4] \wedge \neg w[5]$ and
$[\alpha_2](w) \wedge I_\iota(w) = (\neg w[1] \vee \neg w[2]) \wedge I_\iota(w) = \neg w[1] \wedge \neg w[2] \wedge \neg w[3] \wedge \neg w[4] \wedge \neg w[5]$.
This corresponds to our intuition.

8 Preliminary Experimental Results

In this section we describe an implementation of the UMC algorithm and present some preliminary experimental results for selected benchmark examples.

Our tool, unbounded model checking for interpreted systems, is a new module of the verification environment VerICS [5]. The tool takes as input an interpreted system and a $\mathrm{CTL_pK}$ formula φ and produces a set of states (encoded symbolically), in which the formula holds. The implementation consists of two main parts: the translation module and the *forall* module. According to the detailed description in former sections, each subformula ψ of φ is encoded (by the translation module) by a QBF formula which characterizes all the states at which ψ holds. In case of checking a modal formula, the corresponding QBF formula

is then evaluated by the *forall* module, which is implemented on the top of the SAT solver `Zchaff` [13]. The whole tool is written in C++ making intensive use of STL libraries.

The tests presented below have been performed on a workstation equipped with the AMD Athlon XP+ 2400 MHz processor and 2 GB RAM running under Linux Redhat. For each of the results we present the time (in seconds) used by `VerICS` and `Zchaff`, and give RAM (in kB) consumed during the computation.

8.1 Train, Gate and Controller - example parameterized

The first example we have tested is the train, gate and controller system presented in Section 7. In order to show how the algorithm copes with the combinatorial explosion, this example is parameterized with the number of trains N. For a given $N \in \{2, 4, 6\}$, we have generalized the property α_2 of Section 7 to N trains: $\alpha_2(N) = \mathrm{AG}(\neg in\_tunnel_1 \Rightarrow (\neg \mathrm{K}_{train_1} \bigwedge_{i=2..N} \neg in\_tunnel_i \wedge \neg \mathrm{K}_{train_1} \bigvee_{i=2..N} in\_tunnel_i))$.

The results (time and memory consumption) are presented in the Table 1. *SAT-time* denotes the amount of time necessary to determine by means of unmodified `Zchaff` whether the obtained set of states contains an initial state (this is a SAT problem).

$\alpha_2(N)$				
N	CNF clauses	UMC-mem	UMC-time	SAT-time
2	557	2260 kB	0.12 s	0.01 s
4	5214	8376 Mb	1.51 s	0.01 s
6	58489	64 MB	46.55 s	0.01 s

Table 1. Experimental results for Train-Gate-Controller

8.2 Attacking Generals

The second analyzed example is a scenario of the coordinated attack problem, often discussed in the area of MAS, distributed computing as well as epistemic logic. It concerns coordination of agents in the presence of unreliable communication. It is also known as the coordinated attack problem [8].

For the purpose of this paper, we choose a particular joint protocol for the scenario and verify the truth and falsehood of particular formulas that capture its key characteristics. The variant we analyse is the following (for more detailed protocol description we refer to [10]) :

> After having studied the opportunity of doing so, general A may issue a request-to-attack order to general B. A will then wait to receive an acknowledgment from B, and will attack immediately after having received

it. General B will not issue request-to-attack orders himself, but if his assistance is requested, he will acknowledge the request, and will attack after a suitable time for his messenger to reach A (*assuming no delays*) has elapsed. A joint attack guarantees success, and any non-coordinated attack causes defeat of the army involved (Fig. 2).

Figure 2 presents three scenarios for the agents involved in the coordinated attack problem. The rounded boxes represent locations (local states), while the arrows denote transitions between locations. The beginning location for each agent is in bold. The transitions sharing labels are executed simultaneously (i.e., synchronize). The local states for the agents are listed below:

- $L_{General_A} = \{wait_A, order_A, ack_A, win_A\}$,
- $L_{General_B} = \{wait_B, order_B, ready_B, win_B, fail_B\}$,
- $L_{Environment} = \{wait_E, order_E, ack_E, ack\_lost_E\}$.

In our formulas we use the following propositional variables: $attack_A$ and $attack_B$ meaning that corresponding General has made the decision of attacking the enemy, $success_A$ and $success_B$ meaning the victory of each General and finally $fail_B$ which denotes the defeat of General B (and both Generals). For $s \in G$:

- $attack_A \in \mathcal{V}(s)$ iff $l_{General_A}(s) \in \{win_A, ack_A\}$
- $success_A \in \mathcal{V}(s)$ iff $l_{General_A}(s) \in \{win_A\}$
- $attack_B \in \mathcal{V}(s)$ iff $l_{General_B}(s) \in \{order_B, win_B, ready_B, fail_B\}$
- $success_B \in \mathcal{V}(s)$ iff $l_{General_B}(s) \in \{win_B\}$
- $fail_B \in \mathcal{V}(s)$ iff $l_{General_B}(s) \in \{fail_B\}$

Below we present some properties we test for the coordinated model problem. Results of the tests are listed for each property in the same way as in the previous example.

- $\beta_1 = \mathrm{AG}(attack_B \Rightarrow \mathrm{K}_A \mathrm{K}_B attack_A)$
- $\beta_2 = \mathrm{EF}(C_{\{AB\}}(attack_A \wedge attack_B))$

The property β_1 states that if the general B decides to attack, then the general A knows that B knows that A will attack the enemy. The property β_2 expresses that there is a possibility of achieving common knowledge about the decision of attacking the enemy. The experimental results for this example are given in the Table 2.

Property	CNF clauses	UMC-memory	UMC-time	SAT-time
β_1	917	1488 kB	1.08 s	0.02 s
β_2	971	2300 kB	1.54 s	0.01 s

Table 2. Experimental results for the coordinated attack problem

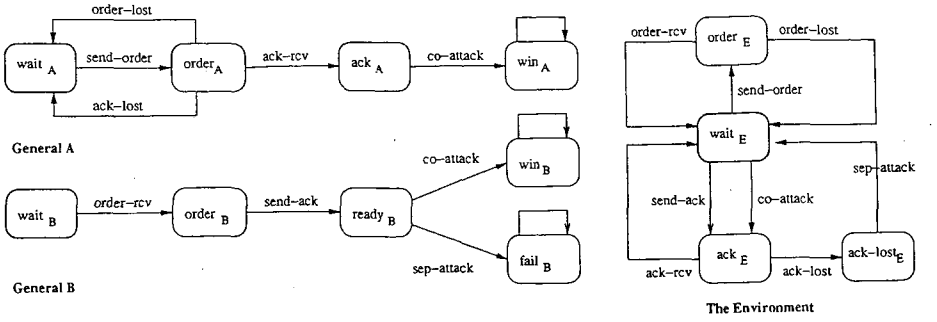

Fig. 2. The attacking generals scenarios

9 Conclusions

Verification of multi-agent systems is quickly becoming an active area of research. In the case of model checking, plain temporal verification is not sufficient because of the variety of modalities that are commonly used to specify multi-agent systems. In this paper we have extended the state-of-the-art of the area by providing a model checking theory to perform unbounded model checking on a temporal epistemic language interpreted on interpreted systems. This surpasses the possibilities available already with other SAT-based approaches, namely bounded model checking, in that it is possible to check the full CTLK language, not just its existential fragment.

It should be noted that our tool provides only a preliminary implementation of UMC. The major problem we found was that blocking clauses are defined only over input variables V. This often seemed to be a too finer description and lead to generating exponentially many clauses (as can be seen in Table 1). We have found that the Alternative Implication Graph $IG(A', \phi')$ usually gives shorter blocking clauses only for simple formulas, while formulas encoding "real" UMC problems produce clauses over all literals of V. In future work we shall investigate the conjecture of K. McMillan stating that by allowing in blocking clauses literals corresponding not only to state vectors, but also to subformulas, one could obtain a dramatic improvement in performance.

References

1. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. of TACAS'99*, volume 1579 of *LNCS*, pages 193–207. Springer-Verlag, 1999.
2. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.
3. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

4. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7):394–397, 1962.
5. P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS: A tool for verifying Timed Automata and Estelle specifications. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 278–283. Springer-Verlag, 2003.
6. E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. of the 7th Int. Colloquium on Automata, Languages and Programming (ICALP'80)*, volume 85 of *LNCS*, pages 169–181. Springer-Verlag, 1980.
7. E. A. Emerson and E. M. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.
8. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.
9. J. Halpern and M. Vardi. *Model checking vs. theorem proving: a manifesto*, pages 151–176. Artificial Intelligence and Mathematical Theory of Computation. Academic Press, Inc, 1991.
10. A. Lomuscio, T. Łasica, and W. Penczek. Bounded model checking for interpreted systems: Preliminary experimental results. In *Proc. of the 2nd NASA Workshop on Formal Approaches to Agent-Based Systems (FAABS'02)*, volume 2699 of *LNAI*, pages 115–125. Springer-Verlag, 2003.
11. K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Proc. of the 14th Int. Conf. on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 250–264. Springer-Verlag, 2002.
12. R. van der Meyden and H. Shilov. Model checking knowledge and time in systems with perfect recall. In *Proceedings of Proc. of FST&TCS*, volume 1738 of *Lecture Notes in Computer Science*, pages 432–445, Hyderabad, India, 1999.
13. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. of the 38th Design Automation Conference (DAC'01)*, pages 530–535, June 2001.
14. W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundamenta Informaticae*, 55(2):167–185, 2003.
15. F. Raimondi and A. Lomuscio. Symbolic model checking of deontic interpreted systems via obdd's. In *Proceedings of DEON04, Seventh International Workshop on Deontic Logic in Computer Science*, volume 3065 of *Lecture Notes in Computer Science*. Springer Verlag, May 2004.
16. F. Raimondi and A. Lomuscio. Towards model checking for multiagent systems via obdd's. In *Proceedings of the Third NASA Workshop on Formal Approaches to Agent-Based Systems (FAABS III)*, Lecture Notes in Computer Science. Springer Verlag, April 2004. This volume.
17. F. Raimondi and A. Lomuscio. Verification of multiagent systems via ordered binary decision diagrams: an algorithm and its implementation. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, July 2004.
18. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
19. W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In *Proc. of the 9th Int. SPIN Workshop (SPIN'02)*, volume 2318 of *LNCS*, pages 95–111. Springer-Verlag, 2002.

20. W. van der Hoek and M. Wooldridge. Tractable multiagent planning for epistemic goals. In *Proc. of the 1st Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, volume III, pages 1167–1174. ACM, July 2002.
21. M. Wooldridge. Computationally grounded theories of agency. In E. Durfee, editor, *Proceedings of ICMAS, International Conference of Multi-Agent Systems*. IEEE Press, 2000.
22. M. Wooldridge. *An introduction to multi-agent systems*. John Wiley, England, 2002.
23. M. Wooldridge, M. Fisher, M. Huget, and S. Parsons. Model checking multiagent systems with mable. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*, Bologna, Italy, July 2002.