Verification of multiagent systems via unbounded model checking [*]

Magdalena Kacprzak
Białystok University of Technology
mdkacprzak@wp.pl

Alessio Lomuscio
King's College London
alessio@dcs.kcl.ac.uk

Wojciech Penczek
ICS PAS and Podlasie Academy
penczek@ipipan.waw.pl

Abstract

We present an approach to the problem of verification of epistemic properties of 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.

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: $\mathrm{M}_P \models \phi$, where M_P is a temporal model representing the executions of P_S. As it is well known, there are intrinsic difficulties with the naïve approach of 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 more than 10^{20} 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 [21]. 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 LTL (or 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 [20]. For the case of knowledge, the semantics of interpreted systems [7], popularised 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 [8]. 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 subdiscipline 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 vari-

* The authors acknowledge support from the Naval Research International (grant N00014-04-1-4080), the Polish National Committee for Scientific Research (grant No 7T11C 00620 and a special grant supporting ALFEBIITE), the Nuffield Foundation (grant NAL/00690/G), and EPSRC (GR/S49353/01).

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.

ous intensional notions and these are paired with standard model checking techniques based on temporal logic. Following this line is for example [22] 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 [9, 10, 15, 18]. In [9] 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 checker. In this approach local propositions need to be computed by the user. In [18] a compiler is given to translate an interpreted system specification into SMV code that is then used to generate the whole state space on which epistemic formulas can be evaluated. The process allows for testing of static epistemic formulas only.

These works were preceded by [15], 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.

The authors of this paper have also contributed to this line. In [16, 17, 12] presented at AAMAS2003, 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 [14] called *unbounded model checking (UMC)*. A byproduct of the work presented here is the definition of a fixed point semantics for the logic $\mathrm{CTL_pK}$, an extension of CTLK by means of 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 $forall$ (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 $\mathrm{CTL_pK}$ formula α to be translated into a propositional formula $[\alpha](w)$ in CNF, which characterizes all the states of the model, where α holds.

For the case of CTL it was shown by McMillan [14] 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; or whenever the SAT-based image computation step proves to be faster than the BDD-based one. Although we do not investigate these results here, similar beneficial effects may occur in the temporal epistemic case discussed here.

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 on. The logic $\mathrm{CTL_pK}$ is defined in Section 3. Section 4 summarises 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 the algorithms for computing CNF formulas equivalent to $\mathrm{CTL_pK}$ formulas are also given. A simple example of using the algorithms for verifying epistemic properties of a train, gate, and controller system is given in Section 7.

2. Interpreted systems semantics

Any transition-based semantics allows for the representation of temporal flows of time by means of a successor relation. For example, CTL is interpreted on plain Kripke models. To work with a temporal epistemic language, we need to consider a semantics that also allows for the automatic representation of the epistemic relations between computational states [20]. The mainstream semantics that allows one to do so is the one of interpreted systems [7].

Interpreted systems can be succinctly defined as follows (we refer to [7] 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 $P_i : L_i \rightarrow 2^{Act_i}$, for $i = 1, \ldots, n$, representing the functioning behaviour of every agent, and a function $P_e : L_e \rightarrow 2^{Act_e}$ for the environment. We can model the computation taking place in the system by means of a transition function $t : G \times Act \rightarrow G$, 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., $W = \{s \in G \mid (\iota, s) \in T^*\}$*[2]*, $\sim_i \subseteq G \times G$ ($i \in A$) 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 $l_i : G \rightarrow L_i$ 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 states 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.

Epistemic relations. 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. The intersection of Γ's accessibility relations defines the epistemic relation corresponding to the modality of distributed knowledge: $\sim_\Gamma^D = \bigcap_{i \in \Gamma} \sim_i$. We refer to [7] for an introduction to these concepts.

2 T^* denotes the reflexive and transitive closure of T.

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 ($\mathrm{CTL_pK}$)

Interpreted systems are traditionally used to give a semantics to an epistemic language enriched with temporal connectives based on linear time [7]. Here we use CTL by Emerson and Clarke [6] 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***, and A a set of agents. The set of $\mathrm{CTL_pK}$ formulas $\mathcal{FORM}$ is defined inductively by using the following BNF syntax: $\phi ::= p \in \mathcal{PV}_K \mid \neg\phi \mid \phi \wedge \phi \mid \mathrm{AX}\phi \mid \mathrm{AG}\phi \mid \mathrm{A}(\phi \mathrm{U} \phi) \mid \mathrm{AY}\phi \mid \mathrm{AH}\phi \mid \mathrm{K}_i\phi, i \in A \mid \mathrm{E}_\Gamma\phi, \Gamma \subseteq A \mid \mathcal{C}_\Gamma\phi, \Gamma \subseteq A \mid \mathrm{D}_\Gamma\phi, \Gamma \subseteq A$.*

Additional Boolean connectives are defined in the usual manner. Moreover, **false** $\overset{def}{=}$ $\neg$**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 and G stand for respectively "at the next step", and "forever in the future". The operators Y and H are their past counterparts "at the previous step", and "forever in the past". U is the *Until* operator: $\alpha \mathrm{U} \beta$ expresses that β occurs eventually and α holds continuously at least until the first occurrence of β.

Definition 3 (Interpretation of $\mathrm{CTL_pK}$) *Let* M *be a model,* $s \in G$ *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 (we omit the definition of the basic propositional connectives):*

$s \models p$ *iff* $p \in \mathcal{V}(s)$,
$s \models \mathrm{AX}\alpha$ *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* $s' \models \alpha$*)*,
$s \models \mathcal{C}_\Gamma\alpha$ *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 the 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. The reason for interpreting the past operators over the states of G rather than of W is strictly technical. In this way, we can easily compute the validity of the past formulas using fixed point equations over the relation T^{-1}. Moreover, the reachable states can be characterized by the formula $\neg \mathrm{AH} \neg init$, where $init$ is a proposition holding in the initial state only. So, past properties over reachable states can be specified as well.

4. Formulas in Conjunctive Normal Form and Quantified Boolean Formulas

The method presented in the next section relies on manipulation of formulas in conjunctive normal form (CNF), and in quantified boolean form (QBF), and related algorithms and techniques for verifying their satisfiability (conflict clauses, block clauses, implication graphs, etc). We are forced to assume familiarity with these concepts and can only report brief definitions to fix the notation. We refer to [11] for more details.

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 $c[1] \wedge \ldots \wedge c[n]$.

The BNF syntax of a QBF formula is given by: $\alpha ::= p \mid \neg\alpha \mid \alpha \wedge \alpha \mid \exists p.\alpha \mid \forall p.\alpha$. The semantics of the quantifiers is defined as follows:

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

where $\alpha \in \mathrm{QBF}$, $p \in \mathcal{PV}$ and $\alpha(p \leftarrow q)$ denotes substitution with the variable q 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 $\forall v[1].\forall v[2] \ldots \forall v[m].\alpha$. For a given QBF formula $\forall v.\alpha$, we can construct a CNF formula equivalent to it by using the algorithm *forall* [14].

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

The procedure deduce (not explicitly given here) 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. The procedure *forall* 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, and it is rather complex. We refer to [11] for details. 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, α).

The proof of the above theorem follows from the correctness of *forall* algorithm (see [14]).

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. To this aim, we follow and adapt definitions given in [19, 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' 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 $\mu Z.\tau(Z) = \bigcup_{i\geq 0} \tau^i(\textbf{false})$. 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 $\nu Z.\tau(Z) = \bigcap_{i\geq 0} \tau^i(\textbf{true})$.
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:

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

Observe that $\langle O\alpha\rangle = O(\langle\alpha\rangle)$, for $O \in \{\mathrm{AX}, \mathrm{AY}, \mathrm{K}_i, \mathrm{E}_\Gamma, \mathrm{D}_\Gamma\}$. 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.

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

The first three equations are standard (see [5], [3]), whereas the fourth one is defined analogously taking account that $\sim^C_\Gamma$ is the transitive closure of $\sim^E_\Gamma$.

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

Let $\mathrm{M} = (\mathcal{K}, \mathcal{V})$ 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 $n_1 + \ldots + n_n = m$. 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., $D_1 = \{1, \ldots, n_1\}, \ldots, D_n = \{m - n_n + 1, \ldots, m\}$. So, each global state $s = (l_1, \ldots, l_n) \in G$ can be represented by a *global state variable* $w = (w[1], \ldots, w[m])$, where each $w[i]$ for $i = 1, \ldots, m$ is a propositional variable in $\mathcal{PV}$. Note that in this way each local state is represented by a tuple of propositional variables.

Let $F_{\mathcal{PV}}$ be the set of propositional formulas over $\mathcal{PV}$, and let *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, v be global state variables. We define the following propositional formulas:

- $I_s(w) := \bigwedge_{i=1}^m \textit{lit}(s_i, w[i])$.
  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 $\neg w[i]$.
- $H_i(w, v) := \bigwedge_{j \in D_i} w[j] \Leftrightarrow v[j]$.
  This formula represents logical equivalence between local state encodings for agent i of two global states encoded by the variables w and v, representing the fact that they represent the same i-local state.
- $T(w, v)$ 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 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 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 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 propositional formula $[\beta](w) \wedge I_\iota(w)$ is satisfiable, i.e., $\iota \in \langle\beta\rangle$. Below, we formalise the above discussion.

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

- $[p](w) := \bigvee_{s \in \langle p \rangle} I_s(w)$*, for* $p \in \mathcal{PV}_\mathcal{K}$*,*
- $[\neg\alpha](w) := \neg[\alpha](w)$, $[\alpha \wedge \beta](w) := [\alpha](w) \wedge [\beta](w)$, $[\alpha \vee \beta](w) := [\alpha](w) \vee [\beta](w)$,
- $[\mathrm{AX}\alpha](w) :=$ *forall*$\big(v, (T(w,v) \Rightarrow [\alpha](v))\big)$,
- $[\mathrm{AY}\alpha](w) :=$ *forall*$\big(v, (T(v,w) \Rightarrow [\alpha](v))\big)$,
- $[\mathrm{K}_i\alpha](w) :=$ *forall*$\big(v, ((H_i(w,v) \wedge \neg gfp_{AH}(\neg I_\iota(v))) \Rightarrow [\alpha](v))\big)$,
- $[\mathrm{D}_\Gamma\alpha](w) :=$ *forall*$\big(v, ((\bigwedge_{i\in\Gamma} H_i(w,v) \wedge \neg gfp_{AH}(\neg I_\iota(v))) \Rightarrow [\alpha](v))\big)$,
- $[\mathrm{E}_\Gamma\alpha](w) :=$ *forall*$\big(v, ((\bigvee_{i\in\Gamma} H_i(w,v) \wedge \neg gfp_{AH}(\neg I_\iota(v))) \Rightarrow [\alpha](v))\big)$,
- $[\mathrm{AG}\alpha](w) :=$*gfp*$_{AG}([\alpha](w))$,
- $[\mathrm{A}(\alpha \mathrm{U} \beta)](w) :=$*lfp*$_{AU}([\alpha](w), [\beta](w))$,
- $[\mathrm{AH}\alpha](w) :=$*gfp*$_{AH}([\alpha](w))$,
- $[\mathcal{C}_\Gamma\alpha](w) :=$*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 $\alpha$ 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 $Z(w) = forall(v, (T(w,v) \Rightarrow Z(v))) \wedge [\alpha](w)$ with $Z(w) = forall(v, (T(v,w) \Rightarrow Z(v))) \wedge [\alpha](w)$. Similarly, the procedure gfp_{C_Γ} is obtained by replacing $Z(w) = [\alpha](w)$ with $Z(w) = forall\big(v, ((\bigvee_{i\in\Gamma} H_i(w,v) \wedge \neg gfp_{AH}(\neg I_\iota(v))) \Rightarrow [\alpha](v))\big)$ and $Z(w) = forall(v, (T(w,v) \Rightarrow Z(v))) \wedge [\alpha](w)$ with $Z(w) = forall(v, ((\bigvee_{i\in\Gamma} H_i(w,v) \wedge \neg gfp_{AH}(\neg I_\iota(v))) \Rightarrow (Z(v) \wedge [\alpha](v))))$.

```
procedure lfp_AU($[\alpha](w), [\beta](w)$),
where $\alpha, \beta$ 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 as we state below. We refer to [11] for a proof.

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.*

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 [10]). 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 need to 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. Traffic lights are placed 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.

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

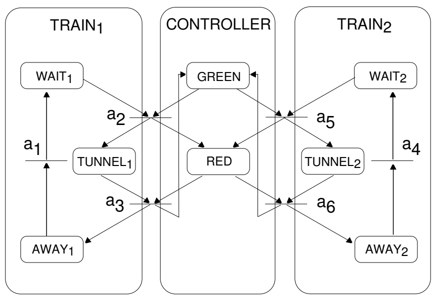

Figure 1. The local transition structures for the two trains and the controller

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 sets $agent(a)$ containing the numbers of the agents that may change local states 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 five 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,
- $H_i(w,v) := \bigwedge_{j \in D_i} w[j] \Leftrightarrow v[j]$, for $i \in \{1,2,3\}$,
- $p_1(w) := \neg w[1] \wedge \neg w[2]$, $p_2(w) := w[1] \wedge \neg w[2]$, $p_3(w) := \neg w[1] \wedge w[2]$, $p_4(w) := w[3]$, $p_5(w) := \neg w[3]$, $p_6(w) := \neg w[4] \wedge \neg w[5]$, $p_7(w) := w[4] \wedge \neg w[5]$, $p_8(w) := \neg w[4] \wedge w[5]$,
  the formula $p_j(w)$ for $j = 1, \ldots, 8$ is a local proposition encoding a particular local state for 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

- $T(w,v) := \bigvee_{a \in Act} \big(\bigwedge_{j \in pre(a)} p_j(w) \wedge \bigwedge_{j \in post(a)} p_j(v) \wedge \bigwedge_{j \in B_a} (w[j] \Leftrightarrow v[j]) \big) \vee (\bigwedge_{a \in Act} \bigvee_{j \in pre(a)} (\neg p_j(w)) \wedge \bigwedge_{j \in D_1 \cup D_2 \cup D_3} (w[j] \Leftrightarrow v[j]))$.
  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 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 local proposition true whenever the local state of $train_1$ is equal to $tunnel_1$ (respectively $train_2$ in state $tunnel_2$).

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:

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

Now we show how to translate the formula α_0: $[\alpha_0](w) = [\neg \text{ AX } (\neg in\_tunnel_1)](w) = \neg[AX\ (\neg in\_tunnel_1)](w)$. The formula $[AX(\neg in\_tunnel_1)](w)$ is computed as follows: $[AX(\neg in\_tunnel_1)](w) = forall(v, T(w,v) \Rightarrow [\neg in\_tunnel_1](v)) = forall(v, T(w,v) \Rightarrow (\neg(\neg v[1] \wedge v[2])))$.

Consequently $[\alpha_0](w) = \neg forall(v, T(w,v) \Rightarrow (v[1] \vee \neg v[2]))$ and $[\alpha_0](w) \wedge I_\iota(w) = \neg forall(v, T(w,v) \Rightarrow (v[1] \vee \neg v[2])) \wedge I_\iota(w) =$ **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 $[\alpha_1](w) \wedge I_\iota(w)$=$I_\iota(w)$ and $[\alpha_2](w) \wedge I_\iota(w)$=$I_\iota(w)$.

This corresponds to our intuition about the scenario.

8. Conclusions

Verification of multiagent 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 multiagent 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.

A description of the implementation of the algorithm presented in this paper and some experimental results are already available [13].

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] 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.

[6] 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.

[7] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.

[8] 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.

[9] 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.

[10] 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.

[11] M. Kacprzak, A. Lomuscio, and W. Penczek. Unbounded model checking for knowledge and time. Technical Report 966, ICS PAS, Ordona 21, 01-237 Warsaw, December 2003. Also available at http://www.ipipan.waw.pl/∼penczek/WPenczek/2003.html.

[12] 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.

[13] T. Łasica, W. Penczek, and M. Szreter. Model checking multi-agent systems with VerICS. In *Proc. of the 3rd NASA Workshop on Formal Approaches to Agent-Based Systems (FAABS III)*, LNCS. Springer-Verlag, 2004. To appear.

[14] 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.

[15] 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.

[16] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundamenta Informaticae*, 55(2):167–185, 2003.

[17] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. In T. Sandholm, editor, *Proc. of the 2nd Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*, pages 209–216. ACM, July 2003.

[18] F. Raimondi and A. Lomuscio. A tool for specification and verification of epistemic and temporal properties of multi-agent system. *Electronic Lecture Notes in Theoretical Computer Science*, 2003. To appear.

[19] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[20] M. Wooldridge. Computationally grounded theories of agency. In E. Durfee, editor, *Proceedings of ICMAS, International Conference of Multi-Agent Systems*. IEEE Press, 2000.

[21] M. Wooldridge. *An introduction to multi-agent systems*. John Wiley, England, 2002.

[22] 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.