Symbolic model checking for temporal-epistemic logic*

A. Lomuscio[1] and W. Penczek[2]

[1] Department of Computing, Imperial College London, UK
[2] ICS PAS Warsaw and UPH Siedlce, Poland

Abstract. We survey some of the recent work in verification via symbolic model checking of temporal-epistemic logic. Specifically, we discuss OBDD-based and SAT-based approaches for epistemic logic built on discrete and real-time branching time temporal logic. The underlying semantical model considered throughout is the one of interpreted system, suitably extended whenever necessary.

1 Introduction

The study of epistemic logic, or logic for the representation of knowledge, has a long and successful tradition in Logic, Computer Science, Economics and Philosophy. Its main motivational thrust is the observation that knowledge of the principals (or *agents*) in an exchange is fundamental in the study not only of the information they have at their disposal, but also in the analysis of their rational actions and, consequently, of the overall behaviour of the system. It is often remarked that the first systematic attempts to develop modal formalisms for knowledge date back to the sixties and seventies and in particular to the works of Hintikka [26] and Gettier [38]. The line of work at the time focussed on the adequacy of particular principles, expressed as axioms of modal logic, representing certain properties of knowledge in a rational setting. The standard framework consisted of the propositional normal modal logic $S5_n$ [6] built on top of the propositional calculus by considering the axioms $K : K_i(p \rightarrow q) \rightarrow K_i p \rightarrow K_i q$, $T : K_i p \rightarrow p$, $4 : K_i p \rightarrow K_i K_i p$, $5 : \neg K_i p \rightarrow K_i \neg K_i p$, together with usual normal rules of necessitation Nec : From φ infer $K_i \varphi$ and modus ponens. Since then several other formalisms have been introduced accounting for weaker notions of knowledge as well as subtly different mental notions such as belief, explicit knowledge and others.

While in the sixties soundness and completeness of these formalisms were shown, the standard semantics considered was the one of plain Kripke models. These are models of the form $M = (W, \{R_i\}_{i \in A}, V)$, where W is a set of "possible worlds", $R_i \subseteq W \times W$ is a binary relation between worlds expressing epistemic indistinguishability between them, and $V : W \rightarrow 2^{PV}$ is an interpretation function for a set of basic propositional variables PV. Indeed, much of

* An earlier version of this paper was published in ACM SIGACT News. Logic Column. 38(3), pp 76-100. 2007. ACM Press.

the theory of modal logic has been developed in this setting up to recent times. However, in the eighties and nineties attention was given to finer grained semantics that accounted for the particular states of computation in a system. In terms of epistemic logic, the challenge was to develop semantics that accounted both to the low-level models of (a-)synchronous actions and protocols, and that at the same time would be amenable to simple yet intuitive notions of knowledge. The key basic semantical concept put forward at the time satisfying these considerations was the one which became popular with the name of "interpreted system". Originally developed independently by Parikh and Ramanujam [51], Halpern and Moses [24] and Rosenscheim [58] and later popularised by [20], the interpreted system model offered a natural yet powerful formalism to represent the temporal evolution of a system as well as the evolution of knowledge of the principals in the run. The development of this model, succinctly described in the next section, triggered a tremendous acceleration in the study of logic for knowledge with several results being produced both in terms of axiomatisations with respect to several different classes of models of agents (synchronous, asynchronous, perfect recall, no learning, etc.) as well as applications of these to standard problems such as coordinated attack, communication, security, and others.

In this setting logic was most often seen as a formal reasoning tool. Attention was given to the exploration of metaproperties of the various formalisms (such as their completeness, decidability, and computational complexity), and axiomatisations developed. Attempts were made to verify systems automatically by exploring the relation $\Gamma_L \vdash \varphi$, where φ is a specification for the system, L is the axiomatised logic representing the system and Γ, a set of formulae expressing the initial conditions. However, partly due to the inherent complexity of some of the epistemic formalisms, verification of concrete systems via theorem proving for epistemic logic did not attract too much attention.

At the same time (the early nineties) the area of verification by model checking [13] began acquiring considerable attention with a stream of results being produced for a variety of temporal logics. The idea of switching attention from theorem proving to model checking became prominent [25]. However, it was not before the very end of the nineties that similar ideas were applied to the verification of multi-agent systems via temporal-epistemic formalisms. The first contribution in the area to our knowledge dates back to a paper by van der Meyden and Shilov [50], where the complexity of model checking perfect recall semantics is analysed. After that attention switched to the possible use of ad-hoc local propositions for translating the verification of temporal-epistemic into plain temporal logic [27]. Following this there were studies on the extension of bounded model checking algorithms [53] and binary-decision diagrams [57]. Several other extensions and algorithms later appeared.

The aim of this paper is to summarise some of the early results obtained by the authors in this area. The area has grown tremendously in recent years and this paper is not intended to provide a survey of the area. The choice of the topics to present is guided by the influence that Prof. Marek Sergot had on

their development. The rest of the paper is organised as follows. In Section 2 we present syntax and semantics of the basic logic. In Section 3 we introduce and discuss an OBDD-based approach to verification of temporal-epistemic logic. In Section 4 an alternative yet complementary approach based on bounded and unbounded model checking is discussed. In Section 5 extensions to real-time are summarised briefly. We conclude in Section 6.

2 Syntax and Semantics

Many model checking approaches differ depending on the syntax supported as a specification language for the properties to be verified by the model checker. We begin here with the basic temporal branching time temporal-epistemic logic.

2.1 Syntax

Given a set of agents $A = \{1, \ldots, n\}$ and a set of propositional variables PV, we define the language $\mathcal{L}$ of CTLK as the fusion between the branching time logic CTL and the epistemic logic $S5_n$ for n modalities of knowledge K_i $(i = 1, \ldots, n)$ and group epistemic modalities E_Γ, D_Γ, and C_Γ $(\Gamma \subseteq A)$:

$$\varphi, \psi ::= p \in PV \mid \neg\varphi \mid \varphi \wedge \psi \mid K_i\varphi \mid E_\Gamma\varphi \mid D_\Gamma\varphi \mid C_\Gamma\varphi \mid AX\varphi \mid AG\varphi \mid A(\varphi U \psi)$$

In addition to the standard Boolean connectives the syntax above defines two fragments: an epistemic and a temporal one. The epistemic part includes formulas of the form $K_i\varphi$ representing "agent i knows that φ", $E_\Gamma\varphi$ standing for "everyone in group Γ knows that φ", $D_\Gamma\varphi$ representing "it is distributed knowledge in group Γ that φ is true", C_Γ formalising "it is common knowledge in group Γ that φ". We refer to [20] for a discussion of these concepts and examples. The temporal fragment defines formulas of the form $AX\varphi$ meaning "in all possible paths, φ holds at next step"; $AG\varphi$ standing for "in all possible paths along φ is always true"; and $A(\varphi U \psi)$ representing "in all possible paths at some point ψ holds true and before then φ is true along the path".

Whenever $\Gamma = A$ we will omit the subscript from the group modalities E, D, and C. As customary we will also use "diamond modalities", i.e., modalities dual to the ones defined. In particular, for the temporal part we use $\mathrm{EF}\varphi = \neg\mathrm{AG}\neg\varphi$, $\mathrm{EX}\varphi = \neg\mathrm{AX}\neg\varphi$ representing "there exists a path where at some point φ is true" and "there exists a path in which at the next step φ is true" respectively. We will also use the $\mathrm{E}(\varphi\mathrm{U}\psi)$ with obvious meaning. For the epistemic part we use overlines to indicate the epistemic diamonds; in particular we use $\overline{\mathrm{K}}_i\varphi$ as a shortcut for $\neg K_i\neg\varphi$, meaning "agent i considers it possible that φ" and similarly for $\overline{\mathrm{E}}_\Gamma$, $\overline{\mathrm{D}}_\Gamma$, and $\overline{\mathrm{C}}_\Gamma$.

Formulas including both temporal and epistemic modalities can represent expressive specifications in particular scenarios, e.g., the evolution of private and group knowledge over time, knowledge about a changing environment as well as knowledge about other agents' knowledge. We refer to [20] for standard examples such as alternating bit protocol, attacking generals, message passing systems, etc.

2.2 Interpreted systems semantics

In what follows the syntax of the specification language supported is interpreted on the multi-agent semantics of Interpreted Systems [20]. Interpreted systems are a fine-grained semantics put forward in [24] to represent temporal evolution and knowledge in multi-agent systems. Although initially developed for linear time, given the applications of this paper we present them in their branching time version. Given the model checking algorithms described later we summarise the formalism below in relation to a branching time model. For more details we refer to [20].

Assume a set of *possible local states* L_i for each agent i in a set $A = \{1, \ldots, n\}$ and a set L_e of possible local states for the environment e. The set of *possible global states* $G \subseteq L_1 \times \cdots \times L_n \times L_e$ is the set of all possible tuples $(l_1, \ldots, l_n, l_e)$ representing a snapshot of the system as a whole. The model stipulates that each agent i performs one of the enabled actions in a given state according to a *protocol function* $P_i : L_i \rightarrow 2^{Act_i}$. P_i maps local states to sets of possible actions for agent i within a repertoire of its actions Act_i. Similarly, the environment e is assumed to be performing actions following its protocol $P_e : L_e \rightarrow 2^{Act_e}$. *Joint actions* $(act_1, \ldots, act_n, act_e)$ are tuples of actions performed jointly by all agents and the environment in accordance with their respective protocol. Joint actions are used to determine the transition function $T \subseteq G \times Act_1 \times \cdots \times Act_n \times Act_e \times G$ which gives the evolution of a system from an initial global state $g^0 \in G$. A *path* $\pi = (g_0, g_1, \ldots)$ is a maximal sequence of global states such that $(g_k, g_{k+1}) \in T$ for each $k \geq 0$ (if π is finite then the range of k is restricted accordingly). For a path $\pi = (g_0, g_1, \ldots)$, we take $\pi(k) = g_k$. By $\Pi(g)$ we denote the set of all the paths starting at $g \in G$.

The model above can be enriched in several ways by expressing explicitly observation functions for the agents in the system or by taking more concrete definitions of the sets of local states thereby modelling specific classes of systems (perfect recall, no learning, etc.). We do not discuss these options here; we simply note that in a later section we will pair this semantics with an automata-based one.

To interpret the formulas of the language $\mathcal{L}$ for convenience we define models simply as tuples $M = (G, g^0, T, \sim_1, \ldots, \sim_n, V)$, where G is the set of the global states reachable from the initial global state g^0 via T; $\sim_i \subseteq G \times G$ is an epistemic relation for agent i defined by $g \sim_i g'$ iff $l_i(g) = l_i(g')$, where $l_i : G \rightarrow L_i$ returns the local state of agent i given a global state; and $V : G \times PV \rightarrow \{true, false\}$ is an interpretation for the propositional variables PV in the language.

The intuition behind the definition of models above is that the global states whose local components are the same for agent i are not distinguishable for the agent in question. This definition is standard in epistemic logic via interpreted systems - again we refer to [20] for more details.

We can use the model above to give a satisfaction relation $\models$ for $\mathcal{L}$ inductively as standard. Let M be a model, $g = (l_1, \ldots, l_n)$ a global state, and φ, ψ formulas in $\mathcal{L}$:

- $(M, g) \models p$ iff $V(g, p) = true$,

- $(M, g) \models \mathrm{K}_i\varphi$ iff for all $g' \in G$ if $g \sim_i g'$, then $(M, g') \models \varphi$,
- $(M, g) \models D_\Gamma\varphi$ iff for all $i \in \Gamma$ and $g' \in G$ if $g \sim_i g'$, then $(M, g') \models \varphi$,
- $(M, g) \models E_\Gamma\varphi$ iff $(M, g) \models \bigwedge_{i\in\Gamma} K_i\varphi$,
- $(M, g) \models C_\Gamma\varphi$ iff for all $k \geq 0$ we have $(M, g) \models E^k_\Gamma\varphi$,
- $(M, g) \models \mathrm{AX}\varphi$ iff for all $\pi \in \Pi(g)$ we have $(M, \pi(1)) \models \varphi$,
- $(M, g) \models \mathrm{AG}\varphi$ iff for all $\pi \in \Pi(g)$ and for all $k \geq 0$ we have $(M, \pi(k)) \models \varphi$,
- $(M, g) \models A(\varphi U \psi)$ iff for all $\pi \in \Pi(g))$ there exists a $k \geq 0$ such that $(M, \pi(k)) \models \psi$ and for all $0 \leq j < k$ we have $(M, \pi(j)) \models \varphi$.

The definitions for the Boolean connectives and the other inherited modalities are given as standard and not repeated here. $E^k\varphi$ is to be understood as a shortcut for k occurrences of the E modality followed by φ, i.e., $E^0\varphi = \varphi$; $E^1\varphi = E\varphi$; $E^{k+1}\varphi = EE^k\varphi$.

2.3 The dining cryptographers problem

The formalism of interpreted systems has been used successfully to model a variety of scenarios ranging from basic communication protocols (e.g., the bit transmission problem, message passing systems), to coordination (e.g., the attacking generals setting), deadlocks (e.g., the train-gate-controller scenario), etc. We refer the reader to the specialised literature; the key consideration here is that in each of these scenarios it is shown that temporal-epistemic languages can be used to express specification for the systems and the individual agents very naturally.

To exemplify this we present a protocol for anonymous broadcast very well-known in the security literature: The dining cryptographers (DC). The DC was introduced by D. Chaum [10] and analysed in a temporal-epistemic setting by Meyden and Su [44]. A reformulation to include cheating cryptographers appears in [32]. We report the original wording here [10] (part of this text was originally cited in [44]).

"Three cryptographers are sitting down to dinner at their favorite three-star restaurant. Their waiter informs them that arrangements have been made with the maitre d'hotel for the bill to be paid anonymously. One of the cryptographers might be paying for dinner, or it might have been NSA (U.S. National Security Agency). The three cryptographers respect each other's right to make an anonymous payment, but they wonder if NSA is paying. They resolve their uncertainty fairly by carrying out the following protocol:

Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer on his right, so that only the two of them can see the outcome. Each cryptographer then states aloud whether the two coins he can see–the one he flipped and the one his left-hand neighbor flipped–fell on the same side or on different sides. If one of the cryptographers is the payer, he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number indicates that NSA is paying (assuming that dinner was paid for only once). Yet if a cryptographer is paying, neither of

the other two learns anything from the utterances about which cryptographer it is."

Temporal-epistemic logic can be used to analyse the specification of the example - we summarise here the description reported in [57, 56]. It is relatively straightforward to model the protocol above by means of interpreted systems. For each agent i we can consider a local state consisting of the triple (l_i^1, l_i^2, l_i^3), representing respectively whether the coins observed are the same or different, whether agent i paid for the bill, and whether the announcements have an even or odd parity. A local state for the environment can be taken as a 4-tuple $(l_e^1, l_e^2, l_e^3, l_e^4)$ where $l_e^1 - l_e^3$ represent the coin tosses for each agent and l_e^4 represents whether or not the agent in question paid for the bill. Actions and protocols for the agents and the environment can easily be given following Chaum's narrative description above and relations for the temporal evolution and the epistemic relation are easily built in this way.

In principle by coding the above we would be able to show on the model for DC that

$$(M_{DC}, g^0) \models \bigwedge_{i \in A} (\mathbf{odd} \wedge \neg \mathbf{paid_i}) \rightarrow AX(K_i(\bigvee_{j \neq i} \mathbf{paid_j}) \bigwedge_{k \neq i} \neg K_i \mathbf{paid_k})$$

The specification above states that if an agent i observes an odd parity and did not cover the bill then in all next states (i.e., when the announcements have been made) she will know that one of the others paid for dinner but without knowing who it was.

Although conceptually easy, the example is already large enough to make it difficult to work out all possible execution traces on the model. Of note is the fact that DC can actually be scaled to any number of cryptographers. By using model checking techniques one can verify DC up to 8 and more cryptographers with resulting state spaces for the model of about 10^{36} states, and considerably more cryptographers if the representation of the model is optimised [32].

Other examples are equally amenable to representation via interpreted systems and model checking via the techniques presented below.

3 OBDD-based Symbolic Model Checking

As it is customary in model checking in the following we analyse systems of finite states only. Given a system S and a property P to be checked, the model checking approach suggests coding S as a logical model M_S, the property P as a logic formula φ_P, and investigating whether $M_S \models \varphi_P$. In the traditional approach the model M_S is finite and represents all the possible computations of system S and φ_P is a formula in temporal logic expressing some property to be checked on the system, e.g., liveness, safety, etc. When φ_P is given in LTL or CTL, checking φ_P on an explicitly given M_S is, of course, a very tractable problem. However, it is impractical to represent M_S explicitly, so M_S is normally implicitly given by means of a dedicated programming language using imperative commands on

sets of variables. This can be convenient for the programmer, but the number of states in the resulting model grows exponentially with the number of variables used in the program describing M_S potentially causing great difficulty (*state explosion problem*).

Much of the model checking literature in plain temporal logic deals with techniques to limit the impact of this, the most prominent being partial order reductions [52, 23], symmetry reductions [12, 18, 19], ordered-binary decision diagrams [8, 47], bounded and unbounded model checking [5, 48], and (predicate) abstraction [16, 3]. By using partial-order reduction techniques the computational tree M_S is pruned and certain provably redundant states are eliminated and/or collapsed with others depending on the formula to be checked thereby reducing the state space. Symmetry reductions are used to reduce the state spaces of distributed systems composed of many similar processes. Predicate abstraction is based on the identification of certain predicates which have no impact on the verification of the formula in question; crucially it is used in verification of infinite-state systems. Binary-decision diagrams (described below) offer a compact representation for Boolean formulas and traditionally constitute one of the leading symbolic approaches. Bounded and unbounded model checking (described in Subsection 4.1 and 4.2, respectively) exploit recent advances in the efficiency of checking satisfiability for appropriate Boolean formulas suitably constructed. Several tools have been developed for model checking temporal logics, including: SPIN [28] for on-the-fly automata-based approach combined with partial-order reductions for LTL, SMV and NuSMV [47, 11] for OBDDs and bounded model checking for LTL and CTL, and POEM [43] for a partial-order semantics. Several other tools exist for other varieties of temporal logic, e.g., real-time logic, probabilistic temporal logic, and indeed other implementations are available for the same or slightly different techniques.

Even if all tools mentioned above are nowadays very sophisticated and support ad-hoc input languages they are traditionally limited to temporal logic only. In the rest of the paper we discuss work by the authors towards techniques and tools supporting temporal-epistemic logic.

3.1 The OBDD approach

The two main model checking platforms for temporal-epistemic logic based on binary-decision diagrams are the MCK and the MCMAS toolkits. They both implement model checking of temporal-epistemic logic on interpreted systems semantics via ordered-binary decision diagrams. MCK [21, 45] implements a variety of different semantics (observational, perfect recall, etc), supports a concise and specialised input language, and was the first model checker available supporting temporal-epistemic logic. MCMAS [40] implements standard interpreted systems semantics and a number of extensions, including deontic modalities, explicit knowledge, ATL, etc. In terms of implementations the two tools are rather different. MCK is implemented in Haskell using Long's BDD library (written in C), whereas MCMAS is implemented in C++ and relies on Somenzi's [59]

BDD package (also in C). MCMAS and its theoretical background is succinctly described in the rest of this section; we refer to [56] for an in-depth description.

Irrespective of the implementation details, the crux of the ordered-binary decision diagrams (OBDDs) approx lies in the symbolic representation of sets and functions paired with the observation that to assess whether $(M, g) \models \varphi$ it is sufficient to evaluate whether $g \in SAT(\varphi)$ where $SAT(\varphi)$ is the set of states in the model M satisfying φ. To introduce the main ideas of the approach we proceed in three stages: first, we observe we can encode sets as Boolean formulas; second, we show how OBDDs offer a compact representation to Boolean functions; third we give algorithms for the calculation of $SAT(\varphi)$.

First of all observe that given a set G of size $|G|$ it is straightforward to associate uniquely a vector of Boolean variables $(w_1, \ldots, w_m)$ to any element $g \in G$ where $m = \lceil log_2|G| \rceil$ (note that a tuple of m places can represent 2^m different elements). Any subset $S \subseteq G$ can be represented by using a characteristic function $f_S : (g_1, \ldots, g_m) \rightarrow \{0, 1\}$, expressing whether the element (as encoded) is in S or otherwise. Note that functions and relations can also be encoded as Boolean functions; for instance to encode that two states are related by some relation we can simply consider a vector of Boolean functions comprising of two copies of the representation of the state to which we add a further Boolean variable expressing whether or not the states are related. Vectors designed in this way represent conjunctions of Boolean atoms or their negation and as such constitute a simple (albeit possibly long) Boolean formula.

In the construction of OBDD-based model checking for plain temporal logic it is normally assumed that the propositions themselves (appropriately ordered) constitute the basis for the encoding of the states of the model. In the MCMAS approach Boolean functions first and then OBDDs are constructed iteratively by considering all aspects of the interpreted system given. These involve building the:

- Boolean functions for the sets of local, global states, actions, and initial global states;
- Boolean functions representing the protocols for each agent, the local evaluation function for each agent, and the valuation for the atoms;
- Boolean functions representing the global temporal relation and the n epistemic relations for the agents. The Boolean formula coding the temporal relation needs to encode that joint actions correspond to enabled actions for all agents: $f_T(g, g') = \bigvee_{a \in JointAct}(g, a, g') \in T \bigwedge_{i \in A} a_i \in P_i(l_i(g))$, where $a = (a_1, \ldots, a_n)$ is a joint action for the system and all individual action components a_i are enabled by the local protocols at the corresponding local state $l_i(g)$ in g. The epistemic relations for the agents can be represented simply by imposing equality on the corresponding local state component.
- A Boolean formula representing the set of reachable states for the interpreted system. This can be encoded as standard by calculating the fix-point of the operator $\tau(Q) = (I(g) \vee \exists g'(T(g, a, g') \wedge Q(g'))$.

Boolean functions are a convenient representation to perform certain logical operations on them (e.g., $\wedge, \vee$); however it is well known that working out their

satisfiability and validity can be expensive. Truth tables themselves do not offer any advantage in this respect: for instance checking satisfiability on them may involve checking 2^n rows of the table where n is the number of atoms present. OBDDs constitute a symbolic representation for Boolean functions and are normally much cheaper to handle. Before introducing OBDDs observe that for every Boolean function we can associate a binary decision tree (BDT), in which each level represents a different atom appearing in the Boolean function. Taking a different path along the tree corresponds to selecting a particular combination of values for the atoms (see Figure 1), thereby determining the truth value of the formula.

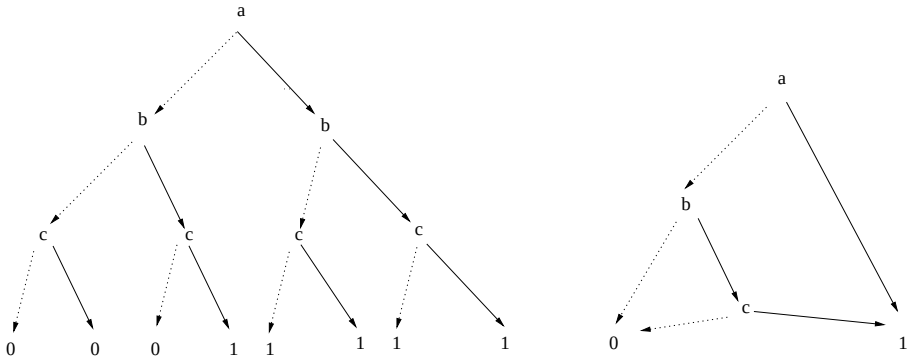

Fig. 1. A BDT for the Boolean function $a \vee (b \wedge c)$ (left) and its corresponding BDD (right). The dotted lines correspond to assigning the value false to the atom whose name the edge leaves from. Conversely solid lines represent assignments to true.

In most instances a BDT is not an efficient representation of its corresponding Boolean function. However, a series of operations can be performed on it to reduce it to a binary decision diagram (BDD). A BDD is a directed acyclic graph with an initial node, and in which each node (representing a Boolean atom) has two edges (corresponding to decision points true and false) originating from it with the final leaves being either "true" or "false" (see Figure 1). There are several algorithms for producing BDDs from BDTs; however the order of the operations on the initial BDT affects the resulting BDD and, most crucially, comparing BDDs turns out to be an expensive operation. What makes the whole approach useful is the provable assertion that there exist sets of algorithms for computing canonical BDDs once the ordering of the variables is fixed. In other words, as long as the ordering of the variables is fixed the resulting BDD is unique for a given Boolean function. This is a remarkable result and leads to an alternative technique to compare Boolean functions: compute their canonical BDDs; if they are the same they represent the same function, if not they are

the result of different functions. The canonical BDDs produced by this set of algorithms are normally referred to as OBDDs and constitute one of the leading data structures in symbolic model checking. We do not discuss algorithms to manipulate BDDs here and refer to [30] for details; but of particular significance is the fact that Boolean operations on Boolean functions can be done directly on the corresponding OBDDs without a very significant loss in performance. Other model-checking specific set operations such as computing pre-images (see below) may also be coded in terms of the corresponding BDDs. For more details on OBDDs and related techniques we refer to [30] Chapter 6 and references, notably [7].

We now present the algorithms for the calculation of the set of states $SAT(\varphi)$ satisfying a formula φ in $\mathcal{L}$. In the OBDD approach all sets of states below are computed symbolically on the corresponding OBDDs.

```
SAT(φ) {
  φ is an atomic formula: return {g | V(g, φ) = true};
  φ is ¬φ1: return S \ SAT(φ1);
  φ is φ1 ∧ φ2: return SAT(φ1) ∩ SAT(φ2);
  φ is EXφ1: return SAT_EX(φ1);
  φ is E(φ1 U φ2): return SAT_EU(φ1, φ2);
  φ is EFφ1: return SAT_EF(φ1);
  φ is K_i φ1: return SAT_K(φ1, i);
  φ is Eφ: return SAT_E(φ);
  φ is Cφ: return SAT_C(φ);
}
```

In the algorithm above, the auxiliary procedures $SAT_{EX}, SAT_{EU}, SAT_{EF}$ follow the standard algorithms used in temporal logic[3]. For instance the set of global states satisfying $EX\varphi$ is computed as follows (in what follows G is the set of reachable states).

```
SAT_EX(φ) {
  X = SAT(φ);
  Y = {g ∈ G | ∃g' ∈ X and T(g, a, g')}
  return Y;
}
```

Note that the calculation of EX involves working out the pre-image of T. The set of states satisfying the epistemic modalities are defined as follow (note that below we use $\sim_\Gamma^E = \bigcup_{i\in\Gamma} \sim_i$ and $\sim_\Gamma^D = \bigcap_{i\in\Gamma} \sim_i$).

[3] For efficiency reasons the CTL modalities implemented are typically EX, AF, and EU.

```
SAT_K(φ, i) {
  X = SAT(¬φ);
  Y = {g ∈ S | ∃g' ∈ X and ∼_i (g, g')}
  return ¬Y ∩ G;
}
```

```
SAT_E(φ, Γ) {
  X = SAT(¬φ);
  Y = {g ∈ G |∼^E_Γ (g, g') and g' ∈ X}
  return ¬Y ∩ G;
}
```

```
SAT_D(φ, Γ) {
  X = SAT(¬φ);
  Y = {g ∈ G |∼^D_Γ (g, g') and g' ∈ X}
  return ¬Y ∩ G;
}
```

```
SAT_C(φ, Γ) {
  Y = SAT(¬φ);
  X = G;
  while ( X ≠ Y ) {
    X = Y;
    Y = {g ∈ G |∼^E_Γ (g, g') and g' ∈ X}
  return ¬Y ∩ G;
}
```

The algorithm for $K_i\varphi$ is similar in spirit to the CTL algorithm for computing $AX\varphi$: essentially we compute the pre-image under the epistemic relation of the set of formulas not satisfying φ and negate the result. $E_\Gamma\varphi$ (respectively $D_\Gamma\varphi$ is done similarly but on the $\sim_E^\Gamma$ ($\sim_D^\Gamma$, respectively). For C we need to use a fix-point construction (fix-point constructions already appear in the algorithm to compute the satisfiability of the until operator). All sets operations above are implemented on the corresponding OBDDs thereby producing the OBDD for $SAT(\varphi)$. We can then solve $(M, g^0) \models \varphi$ by answering the query $g^0 \in SAT(\varphi)$ on the corresponding OBDD.

3.2 MCMAS

MCMAS [40, 46] is a toolkit released under GNU GPL that implements the OBDD-based procedures described in the previous subsection. Input to the model checker is a program describing the evolutions of a multi-agent system. The program is given in ISPL (Interpreted Systems Programming Language), a modelling language specialised for the specifications of interpreted systems and some extensions. An ISPL program consists of a sequence of declarations for agents in the system, valuation for the atomic propositions, and formulas in CTLK (other languages are also supported - see extensions). An agent is given by explicitly listing the local states it may be in, the local actions, protocols, and the local evolution function. Note that the local evolution function $T_i : L_i \times Act_1 \times \cdots \times Act_n \to L_i$ gives a set of rules specifying the target local state when a certain combination of actions is performed. An example of an ISPL fragment describing a very simple agent is given in Figure 2.

Upon invocation the tool parses the input, builds the OBDD for transition relation and the OBDD for the set of reachable states. This is then used in the calculation of the OBDD for the sets of states satisfying the formula to be verified. By comparing whether the initial state belongs to this set the output is displayed. A graphical and a web-interface are available for the tool.

Through MCMAS several scenarios from the areas of web-services, cache-coherence protocols, diagnosis, and security protocols, have been verified. In

```
Agent SampleAgent
 Lstate = {s0,s1,s2};
 Action = {a1,a2}
 Protocol:
   s0: {a1};
   s1: {a2};
   s2: {a1,a2};
 end Protocol
 Ev:
   s1 if ((AnotherAgent.Action=a7);
   s2 if Lstate=s1;
 end Ev
end Agent
```

Fig. 2. A fragment of ISPL code describing an agent.

line with other BDD-based checkers, the size of the model that can be usefully verified depends on the specific example and ranges from 10^6 to 10^{26} reachable global states.

4 SAT-based Symbolic Model Checking

SAT-based model checking is the most recent symbolic approach for modal logic. It was motivated by a dramatic increase in efficiency of SAT-solvers, i.e., algorithms solving the satisfiability problem for propositional formulas [65]. The main idea of SAT-based methods consists in translating the model checking problem for a temporal-epistemic logic to the problem of satisfiability of a formula in propositional logic. This formula is typically obtained by combining an encoding of the model and of the temporal-epistemic property. In principle, the approaches to SAT-based symbolic verification can be viewed as bounded (BMC) or unbounded (UMC). BMC applies to an existential fragment of a logic (here ECTLK) on a part of the model, whereas UMC is for an unrestricted logic (here CTLK) on the whole model.

4.1 Bounded Model Checking

BMC was originally introduced for verification of LTL [5, 4] as an alternative to approaches based on OBDDs. Then, BMC was defined for the existential fragment of the logic CTL [55, 64] and then extended to ECTLK [53]. BMC is based on the observation that some properties of a system can be checked over a part of its model only. In the simplest case of reachability analysis, this approach consists in an iterative encoding of a finite symbolic path as a propositional formula. The satisfiability of the resulting propositional formula is then checked using an external SAT-solver. We present here the main definitions of BMC

for ECTLK and later discuss extensions to more expressive logics. We refer the reader to the literature cited above for more details.

To explain how the model checking problem for an ECTLK formula is encoded as a propositional formula, we first define k-models, bounded semantics over k-models, and then propositional encodings of k-paths in the k-model and propositional encodings of the formulas. In order to define a bounded semantics for ECTLK we define k-models. Let $M = (G, g^0, T, \sim_1, \ldots, \sim_n, \mathcal{V})$ be a model and $k \in \mathbb{N}_+$. The $k-$model for M is defined as a structure $M_k = (G, g^0, P_k, \sim_1 , \ldots, \sim_n, \mathcal{V})$, where P_k is the set of all the k-paths of M over G, where a k-path is the prefix of length k of a path.

We need to identify k-paths that represent infinite paths so that satisfaction of EG formulas in the bounded semantics implies their satisfaction on the unbounded one. To this aim define the function $loop : P_k \rightarrow 2^{\mathbb{N}}$ as: $loop(\pi) = \{l \mid 0 \leq l \leq k \; and \; (\pi(k), \; \pi(l)) \in T\}$, which returns the set of indices l of π for which there is a transition from $\pi(k)$ to $\pi(l)$.

Let M_k be a $k-$model and α, β be ECTLK formulas. $(M_k, g) \models \alpha$ denotes that α is true at the state g of M_k. The bounded semantics is summarised as follows. $(M_k, g) \models \mathrm{EX}\alpha$ has the same meaning as for unbounded models. $(M_k, g) \models \mathrm{EG}\alpha$ states that there is a k-path π, which starts at g, all its states satisfy α and π is a loop, which means that g is a T-successor of one of the states of π. $loop(\pi)$ returns the indexes of such states. For the other modalities the bounded semantics is the same as unbounded, but insisting on reachability of the state satisfying α on a path of length k.
Model checking over models can be reduced to model checking over k-models. The main idea of BMC for ECTLK is that checking φ over M_k is replaced by checking the satisfiability of the propositional formula $[M, \varphi]_k := [M^{\varphi, g^0}]_k \wedge [\varphi]_{M_k}$. $[M^{\varphi, g^0}]_k$ represents (a part of) the model under consideration whereas $[\varphi]_{M_k}$ - a number of constraints that must be satisfied on M_k for φ to be satisfied. Checking satisfiability of an ECTLK formula can be done by means of a SAT-solver. Typically, we start with $k := 1$, test satisfiability for the translation, and increase k by one until either $[M^{\varphi, g^0}]_k \wedge [\varphi]_{M_k}$ becomes satisfiable, or k reaches the maximal depth of M, which is bounded by $|G|$. It can be shown that if $[M^{\varphi, g^0}]_k \wedge [\varphi]_{M_k}$ is satisfiable for some k, then $(M, g^0) \models \varphi$, where M is the full unbounded model.

Translation to SAT We provide here some details of the translation. The states and the transitions of the system under consideration are encoded similarly as for BDDs in Section 3. Let $w = (w[1], \ldots, w[m])$ be sequence of propositions (called a *global state variable*) for encoding global states. A sequence $w_{0,j}, \ldots, w_{k,j}$ of global state variables is called a symbolic k-path j. Since a model for a branching time formula is a tree (a set of paths), we need to use a set of symbolic k-paths to encode it. The number of them depends on the value of k and the formula φ, and it is computed using the function f_k. This function determines the number of k-paths sufficient for checking an ECTLK formula, see [63] for more details. Intuitively, each nesting of an epistemic or temporal

formula in φ increases the value of $f_k(\varphi)$ by 1, whereas subformulas EU, EG and $\overline{\mathrm{C}}_\Gamma$ add more $k-$paths.

The propositional formula $[M^{\varphi,g^0}]_k$, representing the k-paths in the k-model, is defined as follows:

$$[M^{\varphi,g^0}]_k := I_{g^0}(w_{0,0}) \wedge \bigwedge_{j=1}^{f_k(\varphi)} \bigwedge_{i=0}^{k-1} T(w_{i,j}, w_{i+1,j}),$$

where $w_{0,0}$ and $w_{i,j}$ for $0 \leq i \leq k$ and $1 \leq j \leq f_k(\varphi)$ are global state variables, and $T(w_{i,j}, w_{i+1,j})$ is a formula encoding the transition relation T.

An intuition behind this encoding is as follows. The vector $w_{0,0}$ encodes the initial state g^0 and for each symbolic k-path, numbered $1 \ldots f_k(\varphi)$, each pair of the consecutive vectors on this path encodes pairs of states that are in the transition relation T. The formula $T(w, v)$ is typically a logical disjunction of the encodings of all the actions corresponding to the transitions of the model M. This way, one symbolic k-path encodes all the (concrete) k-paths.

The next step of the algorithm consists in translating an ECTLK formula φ into a propositional formula. Let w, v be global state variables. We make use of the following propositional formulas in the encoding:

- $p(w)$ encodes a proposition p of ECTLK over w.
- $H(w, v)$ represents logical equivalence between global state encodings u and v (i.e., encodes that u and v represent the same global states).
- $HK_i(w, v)$ represents logical equivalence between i-local state encodings u and v, (i.e., encodes that u and v share i-local states).
- $L_{k,j}(l)$ encodes a backward loop connecting the k-th state to the l-th state in the symbolic $k-$path j, for $0 \leq l \leq k$.

The translation of each ECTLK formula is directly based on its bounded semantics. The translation of φ at the state $w_{m,n}$ into the propositional formula $[\varphi]_k^{[m,n]}$ is as follows (we give the translation of selected formulas only):

$$\begin{aligned}
[\mathrm{EX}\alpha]_k^{[m,n]} &:= \bigvee_{i=1}^{f_k(\varphi)} \Big(H(w_{m,n}, w_{0,i}) \ \wedge \ [\alpha]_k^{[1,i]} \Big), \\
[\mathrm{EG}\alpha]_k^{[m,n]} &:= \bigvee_{i=1}^{f_k(\varphi)} \Big(H(w_{m,n}, w_{0,i}) \ \wedge \ \big(\bigvee_{l=0}^{k} L_{k,i}(l)\big) \ \wedge \ \bigwedge_{j=0}^{k} [\alpha]_k^{[j,i]} \Big), \\
[\mathrm{E}(\alpha \mathrm{U} \beta)]_k^{[m,n]} &:= \bigvee_{i=1}^{f_k(\varphi)} \Big(H(w_{m,n}, w_{0,i}) \ \wedge \ \bigvee_{j=0}^{k} \big([\beta]_k^{[j,i]} \ \wedge \ \bigwedge_{t=0}^{j-1} [\alpha]_k^{[t,i]} \big) \Big), \\
[\overline{\mathrm{K}}_l \alpha]_k^{[m,n]} &:= \bigvee_{i=1}^{f_k(\varphi)} \Big(I_{g^0}(w_{0,i}) \ \wedge \ \bigvee_{j=0}^{k} \big([\alpha]_k^{[j,i]} \ \wedge \ HK_l(w_{m,n}, w_{j,i}) \big) \Big).
\end{aligned}$$

Intuitively, $[\mathrm{EG}\alpha]^{[m,n]_k}$ is translated to all the $f_k(\varphi)$-symbolic k-paths ($\mathrm{EG}\alpha$ is considered as a subformula of φ) that start at the states encoded by $w_{m,n}$, satisfy α, and are loops. $[\overline{\mathrm{K}}_l \alpha]_k^{[m,n]}$ is translated to all the $f_k(\varphi)$-symbolic k-paths such that each symbolic k-path starts at the initial state g^0, one of its states satisfies α and shares the l-th state with these encoded by $w_{m,n}$. Given the translations above [63], verification of φ over M_k reduces to checking the satisfiability of the propositional formula $[M^{\varphi,g^0}]_k \wedge [\varphi]_{M_k}$, where $[\varphi]_{M_k} = [\varphi]_k^{[0,0]}$.

Several improvements have been suggested to the above encoding of ECTLK such that the length of the formula $[\varphi]_{M_k}$ is reduced. Firstly, the encoding by

Zbrzezny [64] allocates a specific symbolic k-path to each subformula of φ starting with a modality. Moreover, a special structure, called Reduced Boolean Circuit (RCB) [1], is used in [64] for representing the propositional formula $[\varphi]_{M_k}$. A RCB represents subformulas of $[\varphi]_{M_k}$ by fresh propositions such that each two identical subformulas correspond to the same proposition. The above improvements were defined for ECTL only.

Secondly, van der Meyden at al. [29] extended the solution of [64] to ECTLK, but instead of using RCBs, they directly encoded φ such that each subformula ψ of $[\varphi]_{M_k}$ occurring within a scope of a k-element disjunction or conjunction is replaced with a propositional variable p_ψ and the reduced formula $[\varphi]_{M_k}$ is taken in concjuntion with the implication $p_\psi \Rightarrow \psi$.

The above approaches show an improved preformance over the original encoding for some subclasses of ECTLK composed mainly of long and deeply nested formulas.

4.2 Unbounded Model Checking

UMC was originally introduced for verification of CTL [48] as an alternative to BMC and approaches based on BDDs. Then, UMC was extended to $\mathrm{CTL_pK}$ [34] as well as to other more expressive logics.

We begin by extending the syntax and semantics of CTLK to $\mathrm{CTL_pK}$ by adding past operators AY and AH. The operators including *Since* are omitted. A backward path $\pi = (g_0, g_1, \ldots)$ is a maximal sequence of global states such that $(g_{k+1}, g_k) \in T$ for each $k \geq 0$ (if π is finite, then k needs to be restricted accordingly). Let $\overline{\Pi}(g)$ denote the set of all the backward paths starting at $g \in G$.

- $(M, g) \models \mathrm{AY}\varphi$ iff for all $\pi \in \overline{\Pi}(g)$ we have $(M, \pi(1)) \models \varphi$,
- $(M, g) \models \mathrm{AH}\varphi$ iff for all $\pi \in \overline{\Pi}(g)$ and for all $k \geq 0$ we have $(M, \pi(k)) \models \varphi$.

Intuitively, $\mathrm{AY}\varphi$ specifies that for all the backward step states φ holds, whereas $\mathrm{AH}\varphi$ expresses that for all the states in the past φ holds.

Unlike BMC, UMC is capable of handling the whole language of the logic. Our aim is to translate $\mathrm{CTL_pK}$ formulas into propositional formulas in conjunctive normal form, accepted as an input by SAT-solvers.

Specifically, for a given $\mathrm{CTL_pK}$ formula φ, a corresponding propositional formula $[\varphi](w)$ is computed, where w is a global state variable (i.e., a vector of propositional variables for representing global states) encoding these states of the model where φ holds. The translation is not operating directly on temporal-epistemic formulas. Instead, to calculate propositional formulas either the QBF or the fix-point characterisation of $\mathrm{CTL_pK}$ formulas (see Section 3) is used. More specifically, three basic algorithms are exploited. The first one, implemented by the procedure *forall* [48], is used for translating formulas $\mathrm{Z}\alpha$ such that $\mathrm{Z} \in \{\mathrm{AX}, \mathrm{AY}, \mathrm{K}_i, \mathrm{D}_\Gamma, \mathrm{E}_\Gamma\}$. This procedure eliminates the universal quantifiers from a QBF formula characterising 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 $\mathrm{Z}\alpha$ such that $\mathrm{Z} \in \{\mathrm{AG}, \mathrm{AH}, \mathcal{C}_\Gamma\}$. This procedure computes

the greatest fix-point, in the standard way, using Boolean representations of sets rather than sets themselves. For formulas of the form $\mathrm{A}(\alpha \mathrm{U} \beta)$ the third procedure, called lfp_{AU}, computing the least fix-point (in a similar way), is used. In so doing, given a formula φ a propositional formula $[\varphi](w)$ is obtained such that φ is valid in the model M iff the propositional formula $[\varphi](w) \wedge I_{g^0}(w)$ is satisfiable.

The reader is referred to [33] for more details, especially on computing fix-points over propositional representations of sets. In the following section we show how to represent $\mathrm{CTL_pK}$ formulas in QBF and then translate them to propositional formulas in CNF.

From a fragment of QBF to CNF *Quantified Boolean Formulas* (QBF) are an extension of propositional logic by means of quantifiers ranging over propositions. 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 $\alpha(p \leftarrow \mathbf{true}) \vee \alpha(p \leftarrow \mathbf{false})$,
- $\forall p.\alpha$ iff $\alpha(p \leftarrow \mathbf{true}) \wedge \alpha(p \leftarrow \mathbf{false})$,

where $\alpha \in \mathrm{QBF}$, $p \in PV$ and $\alpha(p \leftarrow q)$ denotes substitution with the variable q of every occurrence of the variable p in formula α.
For example, the formula $[\mathrm{AX}\alpha](w)$ is equivalent to the formula $\forall v.(T(w,v) \Rightarrow [\alpha](v))$ in QBF. Similar equivalences are obtained for the formulas $\mathrm{AY}\alpha$, $\mathrm{K}_i\alpha$, $\mathrm{D}_\Gamma\alpha$, and $\mathrm{E}_\Gamma\alpha$ by replacing $T(w,v)$ with suitable encodings of the relations T^{-1}, $\sim_i$, $\sim^D_\Gamma$, and $\sim^E_\Gamma$.

For defining a translation from a fragment of QBF (resulting from the translation of $\mathrm{CTL_pK}$) to propositional logic, one needs to know how to compute a CNF formula which is equivalent to a given propositional formula φ. While the standard algorithm toCNF [48, 54], which transforms a propositional formula to one in CNF, preserving satisfiability only, is of linear complexity, a translation to an equivalent formula is NP-complete. For such a translation, one can use the algorithm *equCNF* - a version of the algorithm toCNF, known as a *cube reduction*. We refer the reader to [9, 22], where alternative solutions can be found. The algorithm *equCNF* is a slight modification of the DPLL algorithm checking satisfiability of a CNF formula (see [54]), but it can be presented in a general way, abstracting away from its specific realisation.

Assume that φ is an input formula. Initially, the algorithm *equCNF* builds a satisfying assignment for the formula $\mathrm{toCNF}(\varphi) \wedge \neg l_\varphi$ (l_φ is a literal used in $\mathrm{toCNF}(\varphi)$), i.e., the assignment which falsifies φ. If one is found, instead of terminating, the algorithm constructs a new clause that is in conflict with the current assignment (i.e., it rules out the satisfying assignment). Each time a satisfying assignment is obtained, a blocking clause is generated by the algorithm `blocking_clause` and added to the working set of clauses. This clause rules out a set of cases where φ is false. Thus, on termination, when there is no satisfying

assignment for the current set of clauses, the conjunction of the blocking clauses generated precisely characterises φ.

A blocking clause could in principle be generated using the conflict-based learning procedure. If we require a blocking clause to contain only input variables, i.e., literals used in φ, then one could either use an (alternative) implication graph [48] in which all the roots are input literals or a method introduced by Szreter [61, 60], which consists in searching a directed acyclic graph representing the formula.

Now, our aim is to compute a propositional formula equivalent to a given QBF formula $\forall p_1 \ldots \forall p_n.\varphi$. The algorithm constructs a formula ψ equivalent to φ and eliminates from ψ the quantified variables on-the-fly, which is correct as ψ is in CNF. The algorithm differs from *equCNF* in one step only, where the procedure `blocking_clause` generates a blocking clause and deprives it of the quantified propositional variables. On termination, the resulting formula is a conjunction of the blocking clauses without the quantified propositions and precisely characterises $\forall p_1 \ldots \forall p_n.\varphi$ (see [33, 54] for more details).

4.3 VerICS

VerICS [17, 35] is a verification tool for real-time systems (RTS) and multi-agent systems (MAS). It offers three complementary methods of model checking: SAT-based Bounded Model Checking (BMC), SAT-based Unbounded Model Checking (UMC), and an on-the-fly verification while constructing abstract models of systems. The theoretical background for its implementation has been presented elsewhere [54, 55].

A network of communicating (timed) automata (together with a valuation function) is the basic VerICS's formalism for modelling a system to be verified. Timed automata are used to specify RTS, whereas timed or untimed automata are applied to model MAS. VerICS translates a network of automata and a temporal-epistemic formula into a propositional formula in CNF and invokes a SAT-solver in order to check for its satisfiability.

Currently, VerICS implements BMC for ECTLKD (ECTLK extended with deontic operators) and TECTLK (see Section 5), and UMC for $\mathrm{CTL_pK}$. VerICS has been implemented in C++; its internal functionalities are available via a interface written in Java [62].

In line with other BMC-based model checkers, the size of the state space, which can be efficiently verified depends on the specific example and the size of the submodel considered. In some scenarios VerICS proved capable to analyse fragments of models as large as 10^{100} states and over.

5 Extensions to Real-Time Epistemic Logic

In this section we briefly discuss some extensions to real-time to the ECTLK framework analysed so far. The timed temporal-epistemic logic TECTLK [42] was introduced to deal with situation where time is best assumed to be dense

and hence modelled by real numbers. The underlying semantics uses networks of *timed automata* [2] to specify the behaviour of the agents. These automata extend standard finite state automata by a set of clocks $\mathcal{X}$ (to measure the flow of time) and time constrains built over $\mathcal{X}$ that can be used for defining guards on the transitions as well invariants on their locations. When moving from a state to another, a timed automaton can either execute action transitions constrained by guards and invariants, or time transitions constrained by invariants only. Crucial for automated verification of timed automata is the definition of an equivalence relation $\equiv \subseteq \mathbb{R}^{|\mathcal{X}|} \times \mathbb{R}^{|\mathcal{X}|}$ on clocks valuations, which identifies two valuations v and v' in which either all the clocks exceed some value c_{max}[4], or two clocks x and y with the same integer part in v and v' and either their fractional parts are equal to 0, or are ordered in the same way, i.e., $fractional(v(x)) \leq fractional(v(y))$ iff $fractional(v'(x)) \leq fractional(v'(y))$. The equivalence classes of $\equiv$ are called *zones*. Since $\equiv$ is of finite index, there is only finitely many zones for each timed automaton.

In addition to the standard epistemic operators, the language of TECTLK contains the temporal operators EG and EU combined with time intervals I on reals in order to specify when precisely formulas are supposed to hold. Note that TECTLK does not include the next step operator EX as this operator is meaningless on dense time models. The formal syntax of TECTLK in BNF is as follows:

$$\varphi, \psi ::= p \in PV \mid \neg p \mid \psi \wedge \varphi \mid \psi \vee \varphi \mid \overline{\mathrm{K}}_i \varphi \mid \overline{\mathrm{E}}_\Gamma \varphi \mid \overline{\mathrm{D}}_\Gamma \varphi \mid \overline{\mathrm{C}}_\Gamma \varphi \mid \mathrm{EG}_I \varphi \mid \mathrm{E}(\varphi \mathrm{U}_I \psi)$$

A (real-time interpreted) model for TECTLK over a timed automaton is defined as a tuple $M = (Q, s^0, T, \sim_1, \ldots, \sim_n, V)$, where Q is the subset of $G \times \mathbb{R}^{|\mathcal{X}|}$ such that G is the set of locations of the timed automaton, all the states in Q are reachable from $s^0 = (g^0, v^0)$ with g^0 being the initial location of the timed automaton and v^0 the valuation in which all the clocks are equal to 0; T is defined by the action and timed transitions of the timed automaton, $\sim_i \subseteq Q \times Q$ is an epistemic relation for agent i defined by $(g, v) \sim_i (g', v)$ iff $g \sim_i g'$ and $v \equiv v'$; and $V : Q \times PV \rightarrow \{true, false\}$ is a valuation function for PV. Intuitively, in the above model two states are in the epistemic relation for agent i if their locations are in this relation according to the standard definition in Section 2 and their clocks valuations belong to the same zone.

In what follows, we give the semantics of $\mathrm{E}(\varphi \mathrm{U}_I \psi)$ and $\mathrm{EG}_I \varphi$ of TECTLK and discuss how BMC is applied to this logic. Differently from the paths of temporal-epistemic models, the paths in real-time models consist of action transitions interleaved with timed transitions. The time distance to a state s from the initial one at a given path can be computed by adding the times of all the timed transitions that has occurred up to this state. Following this intuition the semantics is formulated as follows:

- $(M, s) \models \mathrm{E}(\varphi \mathrm{U}_I \psi)$ iff there is a path in M starting at s which contains a state where ψ holds, reached from s within the time distance of I, and φ holds at all the earlier states,

[4] This constant is computed from a timed automaton and a formula to be verified.

– $(M, s) \models \mathrm{EG}_I\varphi$ iff there is a path in M starting at s such that φ holds at all the states within the time distance of I.

The idea of BMC for $(M, s^0) \models \varphi$, where φ is TECTLK formula, is based on two translations and on the application of BMC for ECTLK. An infinite real-time model M is translated to a finite epistemic model M_d and each formula φ of TECTLK is translated to the formula $cr(\varphi)$ of the logic ECTLK_y, which is a slight modification of ECTLK. The above two translations guarantee that $(M, s^0) \models \varphi$ iff $(M_d, s^0) \models cr(\varphi)$.

Assume we are given a timed automaton A and a TECTLK formula φ. We begin by translating the real-time model M (for A) to M_d. First, the automaton A is extended with one special clock y, an action a_y, and the set of transitions E_y going from each location to itself and resetting the clock y. These transitions are used to start the paths over which sub-formulas of φ are checked. Then, the finite model M_d for the extended timed automaton is built. The model $M_d = (Q_d, q^0, T_d, \sim_1^d, \ldots, \sim_n^d, \mathcal{V}_d)$, where Q_d is a suitably selected (via discretization) finite subset of Q, the relations $T_d, \sim_i^d$ are suitably defined restrictions of the corresponding relations in M, and $\mathcal{V}_d = \mathcal{V}|Q_d$.

The above translation cr of the temporal modalities is non-trivial only. Applying cr to $\mathrm{E}(\alpha \mathrm{U}_I \beta)$ we get the formula $\mathrm{EX}_y \mathrm{E}(cr(\alpha) \mathrm{U} cr((\beta) \wedge p))$, where the operator EX_y is interpreted over the transitions corresponding to the action a_y, and p is a propositional formula characterising zones. A similar translation applies to $\mathrm{EG}_I\alpha$.

After the above two translations have been defined, the model checking of a TECTLK formula φ over M is reduced to model checking of $cr(\varphi)$ over M_d, for which BMC can be used as presented in Section 4.1.

5.1 Example

To exemplify the expressive power of TECTLK we specify a correctness property for an extension of the *Railroad Crossing System* (RCS) [36], a well-known example in the literature of real-time verification. Below, we summarise the description from [42].

The system consists of three agents: Train, Gate, and Controller running in parallel and synchronising through the events: *approach, exit, lower* and *raise*. When a train approaches the crossing, Train sends the signal **approach** to Controller and enters the crossing between 300 and 500 milliseconds (ms) from this event. When Train leaves the crossing, it sends the signal **exit** to Controller. Controller sends the signal **lower** to Gate exactly 100ms after the signal **approach** is received, and sends the signal **raise** signal within 100ms after **exit**. Gate performs the transition **down** within 100ms of receiving the request **lower**, and responds to **raise** by moving **up** between 100ms and 200ms.

Consider the following correctness property: there exists a behaviour of RCS such that agent Train considers possible a situation in which it sends the signal **approach** but agent Gate does not send the signal **down** within 50 ms. This

property can be formalised by the following TECTLK formula:

$$\varphi = \mathrm{EF}_{[0,\infty]}\overline{\mathrm{K}}_{Train}(\mathbf{approach} \wedge \mathrm{EF}_{[0,50]}(\neg\mathbf{down})).$$

By using BMC techniques we can verify the above property for RCS.

6 Conclusions

It has long been argued that epistemic logic provides an intuitive formalism for several specifications of interest in computer science. In this article we have surveyed some of the early contributions by the authors to solving the model checking problem for temporal-epistemic logic in a branching time setting under a discrete and a continuous model of time. The two main approaches here presented, BDD- and SAT-based, are seen as complementary. Indeed, a rather in-depth comparison between VerICS and MCMAS in the context of the dining cryptographers protocol [32] showed each approach to be better suited in some circumstances but not others.

Since the development of these core techniques, several optimisations have been put forward and implemented, including abstraction and symmetry reduction [15, 14], parallel approaches [37], data abstraction [41], partial-order reduction [39], and combinations of BDDs with bounded model checking [31, 49].

The conclusion we can draw from the results above is that temporal-epistemic logic specifications can now be verified effectively with appropriate symbolic model checking techniques.

Afterword

Our joint work on symbolic model checking for multi-agent systems began in 2002 when we met at a meeting of the EU Project Alfebiite. Marek warmly encouraged Alessio, at that time a postdoc under his supervision, to participate in the meeting and to pursue the research direction there identified. Research on the technique summarised in Section 3 began there and continued through a series of further visits encouraged by Marek. The results reported in the other sections of this paper were developed in the following months and years. Following Alessio's move to King's College London later in the same year, our collaboration continued through several joint projects and is still very active today. Marek has regularly provided insightful feedback on these developments. We are indebted to Marek for his advice and encouragement over the past ten years.

Acknowledgements

A. Lomuscio acknowledges support from the UK Engineering and Physical Sciences Research Council (grant no. EP/I00520X/1). W. Penczek is partly supported by the Polish National Science Centre under grant No. DEC-2011/01/B/ST6/01477. The authors are grateful to the anonymous referees for several comments on an earlier draft of this paper.

References

1. P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT-solvers. In *TACAS*, pages 411–425, 2000.
2. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. T. Ball, A. Podelski, and S. K. Rajamani. Boolean and cartesian abstraction for model checking c programs. In *TACAS*, pages 268–283, 2001.
4. A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. In *Highly Dependable Software*, volume 58 of *Advances in Computers*. Academic Press, 2003. Pre-print.
5. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. of the 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 193–207. Springer-Verlag, 1999.
6. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.
7. R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35(8):677–691, 1986.
8. J. R. Burch, E. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1990.
9. P. Chauhan, E. Clarke, and D. Kroening. Using SAT-based image computation for reachability analysis. Technical Report CMU-CS-03-151, Carnegie Mellon University, July 2003.
10. D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.
11. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A new symbolic model verifier. In *Proc. of the 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 495–499. Springer, 1999.
12. E. Clarke, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. In *Proc. of the 5th Int. Conf. on Computer Aided Verification (CAV'93)*, volume 697 of *LNCS*, pages 450–462. Springer-Verlag, 1993.
13. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
14. M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A data symmetry reduction technique for temporal-epistemic logic. In *ATVA*, volume 5799 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2009.
15. M. Cohen, M. Dam, A. Lomuscio, and F. Russo. Abstraction in model checking multi-agent systems. In *AAMAS (2)*, pages 945–952, 2009.
16. D. Dams, R. Gerth, G. Dohmen, R. Herrmann, P. Kelb, and H. Pargmann. Model checking using adaptive state and data abstraction. In *Proc. of the 6th Int. Conf. on Computer Aided Verification (CAV'94)*, volume 818 of *LNCS*, pages 455–467. Springer-Verlag, 1994.
17. 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.
18. E. A. Emerson and C. S. Jutla. Symmetry and model checking. In *Proc. of the 5th Int. Conf. on Computer Aided Verification (CAV'93)*, volume 697 of *LNCS*, pages 463–478. Springer-Verlag, 1993.

19. E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9:105–131, 1995.
20. R. Fagin, J. Y. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.
21. P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proc. of the 16th Int. Conf. on Computer Aided Verification (CAV'04)*, volume 3114 of *LNCS*, pages 479–483. Springer-Verlag, 2004.
22. M. Ganai, A. Gupta, and P. Ashar. Efficient SAT-based unbounded symbolic model checking using circuit cofactoring. In *Proc. of the Int. Conf. on Computer-Aided Design (ICCAD'04)*, pages 510–517, 2004.
23. R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. *Information and Computation*, 150:132–152, 1999.
24. J. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990. A preliminary version appeared in *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, 1984.
25. 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.
26. J. Hintikka. *Knowledge and Belief, An Introduction to the Logic of the Two Notions*. Cornell University Press, Ithaca (NY) and London, 1962.
27. 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.
28. G. J. Holzmann. The model checker SPIN. *IEEE transaction on software engineering*, 23(5):279–295, 1997.
29. X. Huang and C. Luo an R. van der Meyden. Improved bounded model checking for a fair branching-time temporal epistemic logic. In *MoChArt*, pages 95–111, 2010.
30. M. R. A. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, England, 2000.
31. A. V. Jones and A. Lomuscio. Distributed bdd-based bmc for the verification of multi-agent systems. In *AAMAS*, pages 675–682. IFAAMAS, 2010.
32. M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. *Fundamenta Informaticae*, 63(2,3):221–240, 2006.
33. 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.
34. M. Kacprzak, A. Lomuscio, and W. Penczek. From bounded to unbounded model checking for temporal epistemic logic. *Fundamenta Informaticae*, 63(2-3):221–240, 2004.
35. M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Wozna, and A. Zbrzezny. VerICS 2007 - a model checker for knowledge and real-time. *Fundam. Inform.*, 85(1-4):313–328, 2008.
36. I. Kang and I. Lee. An efficient state space generation for the analysis of real-time systems. In *Proc. of Int. Symposium on Software Testing and Analysis*, 1996.

37. M. Z. Kwiatkowska, A. Lomuscio, and H. Qu. Parallel model checking for temporal epistemic logic. In *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 543–548. IOS Press, 2010.
38. W. Lenzen. *Recent work in epistemic logic*, volume 30 of *Acta Philosophica Fennica*. North-Holland, Amsterdam, 1978.
39. A. Lomuscio, W. Penczek, and H. Qu. Partial order reductions for model checking temporal-epistemic logics over interleaved multi-agent systems. *Fundam. Inform.*, 101(1-2):71–90, 2010.
40. A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *CAV*, pages 682–688, 2009.
41. A. Lomuscio, H. Qu, and F. Russo. Automatic data-abstraction in model checking multi-agent systems. In *MoChArt*, volume 6572 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2010.
42. A. Lomuscio, B. Woźna, and W. Penczek. Bounded model checking for knowledge over teal time. *Artificial Intelligence*, 171(16-17):1011–1038, 2007.
43. J. Malinowski and P. Niebert. SAT based bounded model checking with partial order semantics for timed automata. In J. Esparza and R. Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 405–419. Springer, 2010.
44. R. van der Mayden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proc. of the 17th IEEE Computer Security Foundations Workshop (CSFW-17)*, pages 280–291. IEEE Computer Society, June 2004.
45. MCK: Model checking knowledge. http://www.cse.unsw.edu.au/∼mck.
46. MCMAS. http://www-lai.doc.ic.ac.uk/mcmas/.
47. K. McMillan. *Symbolic model checking: An approach to the state explosion problem*. Kluwer Academic Publishers, 1993.
48. 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.
49. A. Meski, W. Penczek, M. Szreter, B. Woźna-Szcześniak, and A. Zbrzezny. Bounded model checking for knowledge and linear time. In *AAMAS*, 2012.
50. R. van der Meyden and N. V. Shilov. Model checking knowledge and time in systems with perfect recall. In *Proc. of the 19th Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'99)*, volume 1738 of *LNCS*, pages 432–445. Springer-Verlag, 1999.
51. R. Parikh and R. Ramanujam. Distributed processes and the logic of knowledge. In *Logic of Programs*, pages 256–268, 1985.
52. D. Peled. All from one, one for all: on model checking using representatives. In *CAV*, pages 409–423, 1993.
53. W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundamenta Informaticae*, 55(2):167–185, 2003.
54. W. Penczek and A. Półrola. *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach*, volume 20 of *Studies in Computational Intelligence*. Springer-Verlag, 2006.
55. W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.
56. F. Raimondi. *Model Checking Multi-Agent Systems*. PhD thesis, University of London, 2006.
57. F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 5, 2007.

58. S. J. Rosenschein. Formal theories of AI in knowledge and robotics. *New generation computing*, 3:345–357, 1985.
59. F. Somenzi. CUDD: CU decision diagram package - release 2.4.0. http://vlsi.colorado.edu/ fabio/CUDD/cuddIntro.html, 2005.
60. M. Szreter. Selective search in bounded model checking of reachability properties. In *Proc. of the 3rd Int. Symp. on Automated Technology for Verification and Analysis (ATVA'05)*, volume 3707 of *LNCS*, pages 159–173. Springer-Verlag, 2005.
61. M. Szreter. Generalized blocking clauses in unbounded model checking. In *Proc. of the 3rd Int. Workshop on Constraints in Formal Verification (CFV'05)*, 2006.
62. VerICS. http://verics.ipipan.waw.pl.
63. B. Woźna, A. Lomuscio, and W. Penczek. Bounded model checking for deontic interpreted systems. In *Proc. of the 2nd Int. Workshop on Logic and Communication in Multi-Agent Systems (LCMAS'04)*, volume 126 of *ENTCS*, pages 93–114. Elsevier, 2005.
64. A. Zbrzezny. Improving the translation from ECTL to SAT. *Fundam. Inform.*, 85(1-4):513–531, 2008.
65. L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proc. of Int. Conf. on Computer-Aided Design (ICCAD'01)*, pages 279–285, 2001.