Chapter 8

Model Checking Temporal Epistemic Logic

Alessio Lomuscio and Wojciech Penczek

Contents

Abstract We survey some of our work on model checking systems against temporal-epistemic specifications, i.e., systems specified in Temporal Epistemic Logic (TEL). We discuss both OBDD-based and SAT-based approaches for verifying systems against TEL specifications. The presentation is grounded on various notions of interpreted systems, including one for modelling real-time systems. We also present a partial-order reduction approach to reduce the models and discuss several alternative and advanced techniques.

Chapter 8 of the *Handbook of Epistemic Logic*, H. van Ditmarsch, J.Y. Halpern, W. van der Hoek and B. Kooi (eds), College Publications, 2015, pp. 397–441.

8.1 Introduction

The study of epistemic logics, or logics 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 the knowledge of the actors (or *agents*) in a system is fundamental not only to the study of the information they have at their disposal, but also to the analysis of their rational actions and, consequently, 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.

At the time considerable attention was given to the adequacy of specific principles, expressed as axioms of modal logic, representing certain properties of knowledge in a rational setting. The standard framework that emerged consisted of the propositional normal modal logic $S5_n$ (see Chapter 1) built on top of the propositional calculus by considering the axioms $K : K_i(p \to q) \to K_i p \to K_i q$, $T : K_i p \to p$, $4 : K_i p \to K_i K_i p$, $5 : \neg K_i p \to K_i \neg K_i p$, together with the rules of necessitation Nec : From φ infer $K_i \varphi$, where K_i denotes the operator for knowledge of agent i, and Modus Ponens. Since then several other formalisms have been introduced accounting for weaker notions of knowledge as well as subtly different informational attitudes such as belief, explicit knowledge, and others.

While in the sixties soundness and completeness of these formalisms were shown, the standard semantics considered was that 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, A is the set of agents, and $V : W \to 2^{PV}$ is an interpretation function for a set of propositional variables PV. Much of the theory of modal logic was developed in this setting up to relatively recent times. However, in the eighties and nineties attention was given to finer grained semantics that explicitly referred to particular states of computation in a system. In terms of epistemic logic, the challenge was to develop a semantics that accounted both for the low-level models of (a-)synchronous actions and protocols, and that at the same time was amenable to simple yet intuitive notions of knowledge. The key formalism put forward at the time satisfying these considerations was the one which became popular with the name of "interpreted system". Originally developed independently by, among others, Parikh and Ramanujam, Halpern and Moses, and Rosenschein, and later popularised in the seminal book "Reasoning about Knowledge" by Fagin, Halpern, Moses, and Vardi, interpreted systems offer a natural yet powerful formalism to represent the temporal evolution of a system as well as the evolution of knowledge of the

agents in a run. The development of this model, succinctly described in the next section, triggered an acceleration in the study of logics for knowledge in the context of Computer Science leading to several results including 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 including their completeness, decidability, and computational complexity. Attempts were made to verify systems automatically by exploring the relation $\Gamma \vdash_L \varphi$, where φ is a specification for the system, L is a logic suitably axiomatised representing the system under analysis, and Γ is 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 logics did not attract significant attention.

At the same time (the early nineties) the area of verification by model checking began acquiring considerable importance with a stream of results being produced for a variety of temporal logics. It was prominently suggested by Halpern and Vardi that model checking, not theorem proving, may provide the suitable verification technique for epistemic concepts. However, it was not before the very end of the nineties that model checking techniques were applied to the verification of multi-agent systems via temporal-epistemic formalisms. To our knowledge the first contribution in the area dates back to 1999 when van der Meyden and Shilov explored the complexity of model checking perfect recall semantics against epistemic specifications. Attention then switched to the use of ad-hoc local propositions for translating the verification of temporal-epistemic into plain temporal logic. Developments of bounded model checking algorithms and BDD-based labelling procedures followed.

The aim of this chapter is to summarise some of the results obtained by the authors in this area. The emphasis is mostly placed on BDD and SAT-based techniques as they constitute the underlying building blocks of many further refinements that followed. The area has grown considerably in recent years and this chapter cannot provide a complete survey of the area. Some additional approaches are discussed in Section 8.7, but others, inevitably, are omitted. In particular, here we only consider approaches where knowledge is treated as a modality interpreted on sets of global states in possible executions and not as a simple predicate as in other approaches.

The rest of the chapter is organised as follows. In Section 8.2 we present syntax and semantics of CTLK, the branching-time combination between

knowledge and time we use throughout the paper, as well as the special class of the interleaved interpreted systems and two known scenarios from the literature. In Section 8.3 we introduce and discuss an OBDD-based approach to the verification of temporal-epistemic logic. In Section 8.4 an alternative yet complementary approach based on bounded and unbounded model checking is discussed. In Section 8.5 extensions to real-time are summarised briefly. In Section 8.6 we present an abstraction technique based on partial orders that is shown to be correct against CTLK. We discuss several alternative and related approaches in Section 8.7.

8.2 Syntax and Semantics

Model checking approaches differ, among other characteristics, by the specification languages they support. In the following we introduce a branching-time version of temporal-epistemic logic.

8.2.1 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 $K_i, i \in A$ and group epistemic modalities E_Γ, D_Γ, and C_Γ ($\Gamma \subseteq A$):

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

where $p \in PV$.

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". 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 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 omit the subscript from the group modalities E, D, and C. As customary we 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 fragment we use overlines to indicate the dual epistemic operators; 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 $\overline{\mathbf{E}_{\mathbf{K}\Gamma}}$, $\overline{\mathbf{C}_{\mathbf{B}\Gamma}}$, and $\overline{\mathrm{C}}_\Gamma$.

Formulas including both temporal and epistemic modalities can represent expressive specifications including the evolution of private and group knowledge over time, knowledge about a changing environment as well as knowledge about other agents' knowledge.

8.2.2 Interpreted systems semantics

In what follows the syntax of the specification language above is interpreted on the multi-agent semantics of interpreted systems. Interpreted systems are a fine-grained semantics often employed to represent the temporal evolution and knowledge in multi-agent systems. Although initially developed for linear time, given the applications of this paper we here present it in its branching time version.

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 $g = (l_1, \ldots, l_n, l_e)$ representing instantaneous snapshots of the system as a whole. The formalism stipulates that each agent i performs one of the enabled actions in a given state according to a *protocol function* $P_i : L_i \to 2^{Act_i}$. P_i maps local states to sets of possible local actions for agent i within its repertoire of local actions Act_i, which includes a special action ϵ known as the *null-action*. Similarly, the environment e is assumed to perform actions following its protocol $P_e : L_e \to 2^{Act_e}$. A *joint action* $a = (act_1, \ldots, act_n, act_e)$ is a tuple of actions performed jointly by all agents and the environment in accordance with their respective protocols. The joint actions form part of the domain of the transition function $T : G \times Act_1 \times \cdots \times Act_n \times Act_e \to G$ 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 for the sets of local states thereby modelling specific classes of systems (perfect recall, no learning, etc.). We do not discuss these options here.

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 $l_i : G \to L_i$ returns the local state of agent i given a global state; and $V : G \times PV \to \{true, false\}$ is an interpretation for the propositional variables PV in the language.

The intuition behind the definition of the relations $\sim_i$ 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.

Let M be a model, $g = (l_1, \ldots, l_n)$ a global state, and φ, ψ formulas in $\mathcal{L}$, the satisfaction relation $\models$ is inductively defined as follows:

- $(M, g) \models p$ 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 $(M, g) \models E^k_\Gamma\varphi$,
- $(M, g) \models AX\varphi$ iff for all $\pi \in \Pi(g)$ we have $(M, \pi(1)) \models \varphi$,
- $(M, g) \models AG\varphi$ 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 usual and are 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., $E^0\varphi = \varphi$; $E^1\varphi = E\varphi$; $E^{k+1}\varphi = EE^k\varphi$.

8.2.3 Interleaved interpreted systems

Interleaved interpreted systems are a restriction of interpreted systems, where all the joint actions are of a special form. More precisely, we assume that if more than one agent is active at a given global state, i.e., executes a non null-action, then all the active agents perform the same (shared) action in that round. Formally, let $Act = \bigcup_{i=1}^{n} Act_i \cup Act_e$ and for each action $a \in Act$ by $Agent(a)$ we mean the set $\{j \in A \cup \{e\} \mid a \in Act_j\}$, i.e., the set of all agents and environment having a in their repertoires. A tuple $(a_1, \ldots, a_n, a_e)$ is a joint action iff there exists a non-null action $a \in Act \setminus \{\epsilon_1, \ldots, \epsilon_n, \epsilon_e\}$ such that $a_j = a$ for all $j \in Agent(a)$, and $a_j = \epsilon_j$

for all $j \in \{1, .., n, e\} \setminus Agent(a)$. By $g \xrightarrow{a} g'$ we denote that there is a transition from g to g' by means of the joint action a.

Similarly to blocking synchronisation in automata, the condition above insists on all agents performing the same non-null action in a global transition; additionally, note that if an agent has the action being performed in its repertoire, it must be performed, for the global transition to be allowed.

8.2.4 Temporal-epistemic specifications

The formalism of interpreted systems has been used successfully to model a variety of scenarios including communication protocols (e.g., the bit transmission problem, message passing systems), coordination protocols (e.g., the attacking generals setting), and cache coherence protocols. We briefly present only two scenarios here and refer the reader to the specialised literature for more details. Our key consideration here is that temporal-epistemic languages seem particularly attractive to express natural and precise specifications involving the information states of the agents in the system.

The dining cryptographers protocol (DCP) for anonymous broadcast is a well-known anonymity protocol in the security literature. We report it in its original wording given by Chaum.

> 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 ((Chaum, 1988, p. 65)).

Temporal-epistemic logic can be used to express the specification of the protocol, which, in turn, can be modelled as an interpreted system. For each agent i in the set of cryptographers A we can consider a local state consisting of the triple (l_i^1, l_i^2, l_i^3), representing, respectively, whether or not the coins observed are the same, 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 tuple $(l_e^1, l_e^2, l_e^3, l_e^4)$ where l_e^1, l_e^2, 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 above and so can the transition function.

The following specifications can be considered for the protocol.

$$AG(\bigwedge_{i \in A}(odd \wedge \neg paid_i) \rightarrow AX(K_i(\bigvee_{j \neq i} paid_j) \bigwedge_{k \neq i} \neg K_i paid_k)) \tag{8.1}$$

$$AG(\bigwedge_{i \in A}(even \rightarrow AX(C_A(\bigwedge_{j \in A} \neg paid_j) \tag{8.2}$$

Specification 8.1 states that when an agent $i \in A$ observes an odd parity in a situation when he did not cover the bill, then in all next states (i.e., when the announcements have been made) he will know that one of the others paid for dinner but without knowing who it was. Specification 8.2 states that when an even parity has been observed, then the cryptographers acquire common knowledge of the fact that none of them paid the bill. Both specifications represent the precise and intuitive requirements of the protocol in question; both formulas can be shown to hold on the protocol.

The DCP is amenable to be scaled to represent any number of cryptographers. The corresponding number of states grows exponentially in the number of cryptographers considered and therefore may only be verified by automated techniques, such as model checking.

The train-gate-controller problem (TGC) is a deadlock protocol in which a number of trains share a tunnel regulated by a traffic signal. While each train runs on its own track, the tunnel can only accommodate one train. A single controller operates a system of traffic lights at both ends of the tunnel, thereby regulating access to the tunnel.

The scenario can be modelled as interpreted system by associating the possible local states *away* from the tunnel, *wait*, and *train-in-tunnel* to each train and the possible local states *green* and *red* to the controller. The local actions for the train consist of *enter the tunnel*, *leave the tunnel*, and *wait* with the obvious effects implemented by the transition function. Assuming the trains work correctly, a train may attempt to enter the tunnel

when it is in the *wait* state and the controller signals the tunnel is free of trains. Protocols can be given for the agents to perform actions following the description above and the transition function can similarly be defined accordingly.

Specifications that can be checked on the TGC include the following.

$$AG(train_1\_in\_tunnel \rightarrow K_1 \neg train_2\_in\_tunnel) \tag{8.3}$$

$$AG(\neg train_1\_in\_tunnel \rightarrow (\neg K_1(train_2\_in\_tunnel) \wedge \neg K_1 \neg train_2\_in\_tunnel)) \tag{8.4}$$

Property 8.3 states that when train 1 is in the tunnel, it knows that train 2 is not in the tunnel. Property 8.4 states that when train 1 is in the tunnel, it does not know whether or not train 2 is in the tunnel. Both specifications can be shown to hold on the TGC model. Variants of the scenario where trains may develop faults have been studied and other specifications have been analysed.

Like DCP the TGC is scalable to any number of trains and may be verified by means of the techniques discussed below.

8.3 OBDD-based Symbolic Model Checking

Given a system S and a specification P to be checked, the model checking approach involves representing S as a logical model M_S, the specification 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 evolutions of the system S; the specification φ_P is a temporal logic formula expressing some property to be checked on the system, e.g., liveness, safety, etc. When the formula φ_P is given in LTL or CTL, checking φ_P on an explicitly given M_S is a tractable problem. It is, however, impractical to present M_S explicitly; instead, M_S is normally given implicitly by means of a program in a dedicated modelling language. This is convenient for the engineer, but the number of states in the resulting model grows exponentially with the number of variables used in the program describing M_S, causing what is commonly referred to as the *state explosion problem*.

Since state-spaces of real systems can be too large to be checked, much of the model checking literature deals with methodologies to limit the impact of the state explosion problem. The most prominent techniques include partial order reduction, symmetry reduction, ordered-binary decision diagrams (OBDDs), bounded and unbounded model checking, and various forms of abstraction. By using partial-order reduction techniques the model M_S

is pruned and provably redundant states are eliminated or collapsed with others depending on the formula to be checked, thereby reducing the size of the state space to be considered. Symmetry reduction techniques are used to reduce the state space of distributed systems composed of many similar processes which can be suitably abstracted. OBDDs are a compact and canonical representation for Boolean formulas, and traditionally offer the underpinnings for the mainstream symbolic approaches to model checking. Bounded and unbounded model checking exploit recent advances in the efficiency of checking satisfiability for appropriate Boolean formulas representing the model and the specification. Abstraction techniques are used to generate smaller models, typically simulations or bisimulations, that can alternatively be checked against the same specification. Predicate abstraction is based on the identification of key predicates, often generated automatically via calls to SMT checkers, which can be used to construct a smaller model for the verification of the formula in question; crucially it is used in verification of infinite-state systems. Several tools have been developed for model checking systems against temporal specifications. These include SPIN, which provides an on-the-fly automata-based approach combined with partial-order reduction for LTL, SMV and NuSMV supporting OBDD-based model checking and bounded model checking for LTL and CTL, POEM supporting partial-order semantic reduction. Several other tools exist for other varieties of temporal logic, e.g., real-time logics and probabilistic temporal logic.

The tools mentioned above are nowadays very sophisticated and support expressive input languages; however they are limited to temporal logics only. In the rest of the chapter we summarise work by the authors towards techniques and tools supporting specifications given in temporal-epistemic logic.

8.3.1 State space representation and labelling

At the heart of the OBDDs approach is 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 $SAT(\varphi)$.

To begin, 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 $m = \lceil log_2 |G| \rceil$ (a tuple of m places can represent

2^m different elements). Any subset $S \subseteq G$ can be represented by using a characteristic function $f_S : (g_1, \ldots, g_m) \to \{0, 1\}$, expressing whether or not the element as encoded is in S. 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 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, denote a Boolean formula.

Given this, Boolean formulas can be used to represent a given interpreted system as follows.

- Sets of local states, global states, actions, and initial global states can be encoded as Boolean formulas for the respective sets.

- Protocols for each agent, the local evolution function for each agent, and the valuation for the atoms can be expressed as Boolean formulas for the respective functions.

- Following this, the global temporal relation and the n epistemic relations for the agents can also be suitably represented as Boolean formulas for the respective relations. The Boolean formula encoding the temporal relation needs to reflect the fact that joint actions are composed of enabled local actions: $f_T(g, g') = \bigvee_{a \in JointAct}(g, a, g') \in T \bigwedge_{i \in A} a_i \in P_i(l_i(g))$, 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.

- The set of reachable global states can be represented by a Boolean formula 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, calculating 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. Observe that every Boolean function we can be associated to 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

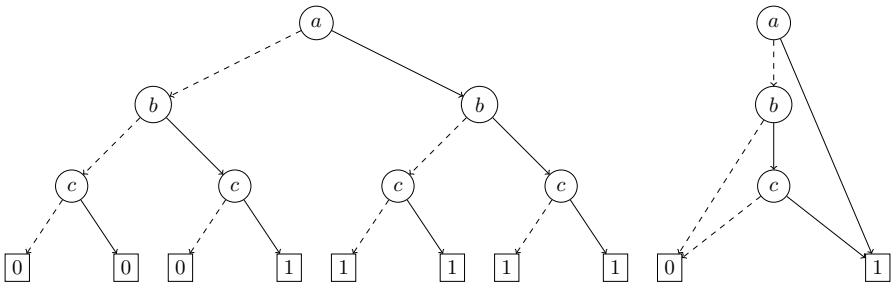

Figure 8.1: A BDT for the Boolean function $a \vee (b \wedge c)$ (left) and its corresponding BDD (right). The dotted lines correspond to assigning the value *false* to the atom whose name the edge leaves from. Conversely the solid lines represent assignments to *true*.

selecting a particular combination of values for the atoms (see Figure 8.1), thereby determining the truth value of the formula.

BDTs are not efficient representations of their corresponding Boolean function. However, a series of operations can be performed on them to reduce them to binary decision diagrams (BDDs). 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 and with the final leaves labelled either as "true" (marked with 1) or "false" (marked with 0) (see Figure 8.1). The order in which operations are applied on the initial BDT affects the resulting BDD and comparing BDDs is also an expensive operation. However, if the ordering of the variables in the BDT is fixed, the resulting reduced BDD is unique (or canonical). This leads to an alternative technique to comparing Boolean functions: compute their canonical BDDs; if they are the same they represent the same Boolean function; if not they represent different functions. The canonical BDDs in which the variables are ordered obtained as above are normally referred to as reduced OBDDs and constitute one of the leading data structures in symbolic model checking.

The reason for this is that operations on Boolean functions and specific set operations such as existential pre-images can be efficiently executed directly on the corresponding OBDDs.

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.

$SAT(\varphi)$ {
  φ is an atomic formula: return $\{g \mid V(g,\varphi) = true\}$;
  φ is $\neg\varphi_1$: return $S \setminus SAT(\varphi_1)$;
  φ is $\varphi_1 \wedge \varphi_2$: return $SAT(\varphi_1) \cap SAT(\varphi_2)$;
  φ is $EX\varphi_1$: return $SAT_{EX}(\varphi_1)$;
  φ is $E(\varphi_1 U \varphi_2)$: return $SAT_{EU}(\varphi_1, \varphi_2)$;
  φ is $EF\varphi_1$: return $SAT_{AF}(\varphi_1)$;
  φ is $K_i\varphi_1$: return $SAT_K(\varphi_1, i)$;
  φ is $E\varphi$: return $SAT_E(\varphi)$;
  φ is $C\varphi$: return $SAT_C(\varphi)$;
}

In the algorithm above, the auxiliary procedures SAT_{EX}, SAT_{EU}, and SAT_{AF} follow the standard algorithms used in temporal logic[1]. For instance the set of global states satisfying $EX\varphi$ is computed as follows (in what follows G is the set of reachable states).

$SAT_{EX}(\varphi)$ {
  $X = SAT(\varphi)$;
  $Y = \{g \in G \mid \exists g' \in X \text{ and } T(g, a, g')\}$
  return Y;
}

Note that the calculation of EX involves computing 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 $\sim_\Gamma^D = \bigcap_{i\in\Gamma} \sim_i$).

$SAT_K(\varphi, i)$ {
  $X = SAT(\neg\varphi)$;
  $Y = \{g \in S \mid \exists g' \in X \text{ and } \sim_i (g, g')\}$
  return $\neg Y \cap G$;
}

$SAT_E(\varphi, \Gamma)$ {
  $X = SAT(\neg\varphi)$;
  $Y = \{g \in G \mid \sim_\Gamma^E (g, g') \text{ and } g' \in X\}$
  return $\neg Y \cap G$;
}

$SAT_D(\varphi, \Gamma)$ {
  $X = SAT(\neg\varphi)$;
  $Y = \{g \in G \mid \sim_\Gamma^D (g, g') \text{ and } g' \in X\}$
  return $\neg Y \cap G$;
}

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

```
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 computed similarly but on $\sim_E^\Gamma$ ($\sim_D^\Gamma$, respectively). For C we use a fix-point construction; note that fix-point constructions already appear in the algorithm to compute the satisfiability of the CTL until operator. All sets operations above are implemented on the corresponding OBDDs thereby producing the OBDD for $SAT(\varphi)$. We can now recast the model checking query $(M, g^0) \models \varphi$ into $g^0 \in SAT(\varphi)$ where g^0 and $SAT(\varphi)$ are suitably encoded as OBDDs.

8.3.2 MCMAS

MCMAS is an open-source toolkit that implements the OBDD-based procedures described above. The model checker takes as input a program describing the evolutions of a multi-agent system and a set of specifications to be checked and returns as output whether or not the specifications are satisfied and witnesses or counterexamples for them. The program is given in ISPL (Interpreted Systems Programming Language), a modelling language suited to represent interpreted systems. An ISPL program consists of a sequence of declarations for the agents in the system, an evaluation for the atomic propositions, and a set of specifications in CTLK (other specification languages including ATL are also supported) to be checked.

In line with interpreted systems an agent is modelled by describing the variables that define it (bounded integers, Boolean, and enumeration types), the set of local actions, protocols, and the local evolution function. This is given in terms of a set of rules governing the value of the target local states when global actions are performed in given sets of local states. A very simple ISPL agent is described in Figure 8.2.

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. MCMAS can be used within Eclipse where

```
1    Agent Sender
2      Vars:
3        bit : {b0, b1};
4        ack : boolean;
5      end Vars
6      Actions = {sb0, sb1, null};
7      Protocol:
8        bit=b0 and ack=false : {sb0};
9        bit=b1 and ack=false : {sb1};
10       ack=true : {null};
11     end Protocol
12     Evolution:
13       (ack=true) if (ack=false) and
14          ( ( (Receiver.Action=sendack) and
15              (Environment.Action=sendSR) )
16           or
17          ( (Receiver.Action=sendack) and
18            (Environment.Action=sendR) )
19          );
20     end Evolution
21   end Agent
```

Figure 8.2: A simple ISPL Agent representing the sender agent in the bit transmission problem.

various functionalities, including counterexample and witness generation, are supported.

Through MCMAS several scenarios from areas like web-services, cache-coherence protocols, diagnosis, and security protocols, have been verified. In line with other BDD-based checkers, the size of the model that can be usefully verified depends on the specific example but is normally around 10^{12} reachable global states. The corresponding number of possible global states can be greater than 10^{30}.

8.4 SAT-based Symbolic Model Checking

SAT-based model checking is the most recent symbolic approach for modal logic. It has been motivated by a dramatic increase in efficiency of SAT-solvers, i.e., algorithms solving the satisfiability problem for propositional formulas. 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 $\mathrm{CTL_pK}$) on the whole model.

8.4.1 Bounded Model Checking

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 at the end of this chapter.

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 start with defining k-models. Let $M = (G, g^0, T, \sim_1, \ldots, \sim_n, V)$ 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 in the unbounded one. To this aim define the function $loop : P_k \to 2^{\mathbb{N}}$ as: $loop(\pi) = \{l \mid 0 \leq l \leq k \text{ and } (\pi(k), \pi(l)) \in T\}$, 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 φ, ψ be ECTLK formulas. $(M_k, g) \models \alpha$ denotes that φ is true at the state g of M_k. The bounded semantics is summarised as follows. $(M_k, g) \models \mathrm{EX}\varphi$ has the same meaning as for unbounded models, for $k > 0$. $(M_k, g) \models \mathrm{E}(\varphi \mathrm{U} \psi)$ has the same meaning as for unbounded models. $(M_k, g) \models \mathrm{EG}\varphi$ states that there is a k-path π, which starts at g, all its states satisfy φ and π is a loop, which means that one of the states of π is a T-successor of the last state of π. $loop(\pi)$ returns the indexes of such states of π. For the epistemic modalities $(\overline{K}_i\varphi, \overline{E_\Gamma}\varphi, \overline{D}_\Gamma\varphi, \overline{C}_\Gamma\varphi)$ the bounded semantics is the same as unbounded, but insisting on reachability of the state satisfying φ on a k-path starting in g^0.

Model checking over models can be reduced to model checking over k-models, called BMC. The main idea of BMC for ECTLK is that checking φ over M_k is replaced by checking the satisfiability of the propositional formula $[M, \varphi]_k := [M^{\varphi, g^0}]_k \wedge [\varphi]_{M_k}$. The formula $[M^{\varphi, g^0}]_k$ represents the k-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 model. If $[M^{\varphi,g^0}]_k \wedge [\varphi]_{M_k}$ is not satisfiable for some k, then we cannot infer that $(M, g^0) \not\models \varphi$ unless k reaches the size of the model M.

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 8.3. 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. Intuitively, each nesting of an epistemic or temporal formula in φ increases the value of $f_k(\varphi)$ by 1, whereas the 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 the transitions corresponding to all the actions 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 the logical equivalence between the global state encodings u and v (i.e., encodes that u and v represent the same global states).
- $HK_i(w,v)$ represents the logical equivalence between the i-local state encodings u and v (i.e., encodes that u and v share the 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, where we give the translation of some selected formulas only. Let w be $w_{m,n}$ and let v denote $w_{0,i}$.

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

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, 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. Given the translations above, 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 $[\varphi]_{M_k} = [\varphi]_k^{[0,0]}$.

Example 8.1
Below we show a part of the model encoding of the TGC protocol for two trains and the controller. Each train can be in 3 different local states, so we need 2 bits for representing its local states: $away_i$ by $(0,0)$, $wait_i$ by $(0,1)$, and $train_i\_in\_tunnel$ by $(1,0)$, where $i \in \{1,2\}$. The controller can be in 2 local states, so the binary representation of these states requires only one bit as follows: $green$ by (0) and red by (1). Thus, a global state, which is composed of a local state for train 1, a local state for the controller, and a local state for train 2, requires 5 bits to be represented, e.g., the initial state $g^0 = (away_1, green, away_2)$ is represented by $(0,0,0,0,0)$. In order to encode global states in the propositional logic we use global state variables being vectors composed of 5 propositional variables. Let

$w = (w[1], ..., w[5])$, $v = (v[1], ..., v[5])$ be two global state variables, and $A = \{1, 2, 3\}$, $D_1 = \{1, 2\}$, $D_2 = \{3\}$, and $D_3 = \{4, 5\}$. The initial state encoding over w is as follows: $I_{g^0}(w) := \bigwedge_{i \in D_1 \cup D_2 \cup D_3} \neg w[i]$.

In order to encode the global transition relation T in propositional logic we need first to encode sets of the global states sharing one local state. To this aim we number all the local states according to their position from 1 to 8 in the sequence $(away_1, wait_1, train_1\_in\_tunnel, green, red, away_2, wait_2, train_2\_in\_tunnel)$.

The formula $p_i(w)$ for $i \in \{1, \ldots, 8\}$ encodes all the global states, containing the local state numbered i, where: $p_1(w) := \neg w[1] \wedge \neg w[2]$ for $away_1$, $p_2(w) := \neg w[1] \wedge w[2]$ for $wait_1$, $p_3(w) := w[1] \wedge \neg w[2]$ for $train_1\_in\_tunnel$, $p_4(w) := w[3]$ for $green$, $p_5(w) := \neg w[3]$ for red, $p_6(w) := \neg w[4] \wedge \neg w[5]$ for $away_2$, $p_7(w) := \neg w[4] \wedge w[5]$ for $wait_2$, $p_8(w) := w[4] \wedge \neg w[5]$ for $train_2\_in\_tunnel$.

Let $agent(t)$ be the set of the numbers of agents whose local states are changed by executing the global transition t. For $t \in T$ let $B_t := \bigcup_{j \in A \setminus agent(t)} D_j$, be the set of the numbers of the bits of a global state that are not changed by t, $pre(t)$ be the set of the numbers of local states from which t has to be executed and $post(t)$ be the set of the numbers of local states reached after executing t.

It is easy to check that for the transition $t = enter_1\_in\_tunnel$, we have $pre(t) = \{2, 4\}$ and $post(t) = \{3, 5\}$, for the transition $t = leave_1\_in\_tunnel$, $pre(t) = \{3, 5\}$ and $post(t) = \{1, 4\}$, and for the transition $t = wait_1$, $pre(t) = \{1\}$ and $post(t) = \{2\}$.

The encoding $T(w, v)$ of the global transition relation T of TGC is as follows:

$$T(w,v) := \bigvee_{t \in T} \Big(\bigwedge_{i \in pre(t)} p_i(w) \wedge \bigwedge_{i \in post(t)} p_i(v) \wedge \bigwedge_{i \in B_t} (w[i] \Leftrightarrow v[i]) \Big).$$

Several improvements have been suggested to the above encoding of ECTLK such that the length of the formula $[\varphi]_{M_k}$ is reduced. They are listed in the final section. These approaches show an improved performance over the original encoding for some subclasses of ECTLK composed mainly of long and deeply nested formulas.

8.4.2 Unbounded Model Checking

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

We begin by extending the syntax and semantics of CTLK to $\mathrm{CTL_pK}$ by adding the 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 predecessor 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 the 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 Quantified Boolean Formula (QBF) or the fix-point characterisation of $\mathrm{CTL_pK}$ formulas (see Section 8.3) is used. More specifically, three basic algorithms are exploited. The first one, implemented by the procedure *forall*, defined by McMillan, is used for translating the 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 the 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 $\mathrm{A}(\varphi \mathrm{U} \psi)$ 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.

In the following section we show how to represent the $\mathrm{CTL_pK}$ formulas in QBF and then translate them to propositional formulas in CNF.

From a fragment of QBF to CNF

The *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 $\alpha \in$ QBF, $p \in PV$ and $\alpha(p \leftarrow q)$ denotes substitution with the variable q of every occurrence of the variable p in the formula α.

For example, the formula $[\mathrm{AX}\alpha](w)$ is equivalent to $\forall v.(T(w,v) \Rightarrow [\alpha](v))$ 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}, $\sim_i$, $\sim_\Gamma^D$, and $\sim_\Gamma^E$.

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, which transforms a propositional formula to one in CNF, preserving satisfiability only, is of linear complexity, a translation to an equivalent propositional formula in CNF is NP-complete. For such a translation, one can use the algorithm *equCNF* - a version of the algorithm toCNF, known as a *cube reduction*. The algorithm *equCNF* is a slight modification of the DPLL algorithm checking satisfiability of a CNF formula, 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 of McMillan, in which all the roots are input literals or a method introduced by Szreter, 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 $\forall p_1 \ldots \forall p_n.\varphi$.

8.4.3 VerICS

VerICS is a verification tool, developed at ICS PAS, that implements the SAT-based BMC and UMC procedures described above as well as in Section 8.5. It offers three complementary methods of model checking: SAT-based BMC, SAT-based UMC, and an on-the-fly verification while constructing abstract models of systems. 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 real time systems, 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. VerICS has been implemented in C++; its internal functionalities are available via a interface written in Java. In line with other SAT-based model checkers, the size of the state space, which can be efficiently verified depends on the specific example and ranges between 10^6 and 10^{50}, which corresponds to encoding and checking $k-$models with k ranging from 10 to 40 approximately.

8.5 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 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* 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}[2], 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:

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \overline{\mathrm{K}}_i\varphi \mid \overline{\mathbf{E}_{\mathbf{K}\Gamma}}\varphi \mid \overline{\mathbf{C}_{\mathbf{B}\Gamma}}\varphi \mid \overline{\mathrm{C}}_\Gamma\varphi \mid \mathrm{EG}_I\varphi \mid \mathrm{E}(\varphi \mathrm{U}_I \varphi)$$

with $p \in PV$. 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 8.2 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 have 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,
- $(M, s) \models \mathrm{EG}_I\varphi$ iff there is a path in M starting at s such that φ holds at all the states within the time distance of I.

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

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 labelled with a_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 discretisation) 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 8.4.1.

8.5.1 Example

To exemplify the expressive power of TECTLK we specify a correctness property for an extension of the *Railroad Crossing System* (RCS), a well-known example in the literature of real-time verification. Below, we give its short description.

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.

8.6 Partial Order Reductions

Several approaches are available to alleviate the difficulty of verifying large state spaces. Partial order reductions have extensively been used in the verification of reactive systems specified against $\mathrm{LTL}_{-\mathrm{X}}$ and $\mathrm{CTL}_{-\mathrm{X}}$ formulas. So far, the only approach which has been implemented and proved efficient for MAS was defined over interleaved interpreted systems. We follow this approach in our presentation.

The main idea behind the partial order reductions is the observation that two paths that differ only in the ordering of independent actions will satisfy the same temporal properties, provided the next step operator is not used. Therefore, rather than dealing with the full model, one can generate a reduced one, which does not contain all the interleavings of the independent actions and still satisfies the same properties. In order to describe partial order reductions used for generating reduced models, we need the following relations and definitions.

Let $i \in A$, $g, g' \in G$, and $J \subseteq A$. The relation $\sim_J = \bigcap_{j\in J} \sim_j$ corresponds to the indistinguishably relation for the epistemic modality of distributed knowledge in group J, whereas the relation $I = \{(a,b) \in Act \times Act \mid Agent(a) \cap Agent(b) = \emptyset\}$ is referred to as the *independence relation* in partial order approaches. Notice that $\sim_\emptyset = G \times G$ while $\sim_{\mathrm{A}} = id_G$. Two actions $a, a' \in Act$ are *dependent* if $(a,a') \notin I$. An action $a \in Act$ is *invisible* in a model M if whenever $g \xrightarrow{a} g'$ for any two states $g, g' \in G$ we have that $V(g) = V(g')$. Given two models $M = (G, g, T, \sim_1, .., \sim_n, V)$ and $M = (G', g', T', \sim'_1, .., \sim'_n, V')$. If $G' \subseteq G$, $g' = g$ and $V' = V|G'$, then we write $M' \subseteq M$ and say that M' is a submodel of M, or that M' is a *reduced* model of M.

As mentioned above, the idea of verification by model checking with partial order reductions is to define an algorithm which generates reduced models which preserve the satisfaction of a class of formulae. In general, the algorithm can be defined for several classes of formulae, which are subsets of CTL*K. In what follows we present an algorithm for CTLK_{-X}, i.e., CTLK without the next step operator. Observe that the formula $EX(executed_a \wedge EXexecuted_b)$, where the proposition $executed_x$ denotes that action $x \in$

$\{a, b\}$ is executed, can distinguish between two paths in which the ordering of the actions a and b is different. This explains why the next step operator is not used in this context.

For $J \subseteq A$, we write CTLK^J_{-X} for the restriction of the logic CTLK_{-X} such that for each subformula $K_i\varphi$ we have $i \in J$. One can define a notion of equivalence on models, called J-stuttering bisimulation, which guarantees the preservation of the formulas in CTLK^J_{-X}. The algorithm presented explores the given model M and returns a reduced one, which is J-stuttering bisimilar to M. Traditionally, in partial order reduction the exploration is carried out by depth-first-search (DFS). In this context DFS is used to compute successor states that will make up the reduced model by exploring systematically the possible computation tree and selecting only some of the possible paths generated. In the following a stack represents the path $\pi = g_0 a_0 g_1 a_1 \cdots g_n$ currently being visited. For the top element of the stack g_n the following three operations are computed in a loop:

1. The set $en(g_n) \subseteq Act$ of enabled actions (not including the ϵ action) is identified and a subset $E(g_n) \subseteq en(g_n)$ of possible actions is heuristically selected (see below).

2. For any action $a \in E(g_n)$ compute the successor state g' such that $g_n \xrightarrow{a} g'$, and add g' to the stack thereby generating the path $\pi' = g_0 a_0 g_1 a_1 \cdots g_n a g'$. Recursively proceed to explore the submodel originating at g' in the same way by means of the present algorithm beginning at step 1.

3. Remove g_n from the stack.

The algorithm begins with a stack comprising of the initial state and terminates when the stack is empty. The model generated by the algorithm is a submodel of the original one. The choice of $E(q)$ is constrained by the class of formulae of CTLK^J_{-X} that must be preserved. Below we give the conditions defining a heuristics for the selection of $E(g)$ (such that $E(g) \neq en(g)$) while visiting state g in the algorithm above.

C1 No action $a \in Act \setminus E(g)$ that is dependent on an action in $E(g)$ can be executed before an action in $E(g)$ is executed.

C2 For every cycle in the constructed state graph there is at least one node g in the cycle for which $E(g) = en(g)$, i.e., for which all the successors of g are expanded.

C3 All actions in $E(g)$ are invisible.

C4 $E(g)$ is a singleton set.

CJ For each action $a \in E(g)$, $Agent(a) \cap J = \emptyset$, i.e., no action in $E(g)$ changes local states of the agents in J.

The conditions $\mathbf{C1} - \mathbf{C3}$ are used for preserving $\mathrm{LTL}_{-\mathrm{X}}$, **C4** for preserving $\mathrm{CTL}_{-\mathrm{X}}$, whereas **CJ** is aimed at preserving the truth value of subformulae of the form $K_i\varphi$ for $i \in J$.

8.6.1 Evaluation

In order to evaluate partial order reductions generated by the implementation of the above method, a prototype tool, based on MCMAS, was implemented. Experimental results have been obtained for three well-knows systems in the literature on MAS: The Train, Gate, and Controller (TGC),the Dining Cryptographers Protocol (DCP) introduced earlier, and the Write-Once cache coherence protocol (WO).

Starting with TGC, the property checked expresses that whenever the train 1 is in the tunnel, it knows that no other train is in the tunnel at the same time:

$$AG(\mathrm{in\_tunnel}_1 \rightarrow K_{\mathrm{train}_1} \bigwedge_{i=2}^{n} \neg \mathrm{in\_tunnel}_i), \tag{8.5}$$

where n is the number of trains in the system, and the proposition in_tunnel$_i$ holds in the states where the train i is in the tunnel. The results showed that the size of the reduced state space $R(n)$ generated by the algorithm is a function of the number of trains n, for $1 \leq n \leq 10$, which turns out to be exponentially smaller when compared to the size of the full state space $F(n)$ below:

- $F(n) = c_n \times 2^{n+1}$, for some $c_n > 1$,
- $R(n) = 3 + 4(n-1)$.

Regarding the DCP scenario and the Write-Once cache coherence protocol, it was found that the algorithm for CTLK^J_{-X} brings negligible benefits for the tested properties. However, in both the cases, substantial reductions are obtained if the properties are expressed in LTLK^J_{-X}. This can be explained by the fact that in order to preserve LTLK^J_{-X} the set $E(g)$ does not need to satisfy the condition **C4**, thereby leading to smaller reduced models

8.7 Notes

Background. Treatments of epistemic logic (see Chapter 1) in terms of the modal system $S5_n$ are normally attributed to the work of Hintikka (1962). In these approaches the semantics is given in terms of Kripke models as introduced by Kripke (1959). The formalism of interpreted systems, introduced in Section 8.2 as a semantics for epistemic logic was put forward independently by Parikh and Ramanujam (1985), Rosenschein (1985), and by Halpern and Moses (1990), and later popularised by Fagin, Halpern, Moses, and Vardi (1995). Interleaved interpreted systems were studied by Lomuscio, Penczek, and Qu (2010) in the context of partial order reductions. They are inspired by standard synchronisation patterns used in network of automata. Other variants of interpreted systems have been discussed in the literature: see the work of Jamroga and Ågotnes (2007), and of Kouvaros and Lomuscio (2013).

Over the years, several scenarios have been modelled as interpreted systems and specified by means of temporal-epistemic properties. Fagin, Halpern, Moses, and Vardi (1995) present an in-depth analysis of a number of scenarios, including communication protocols, against epistemic specifications. The dining cryptographers problem was originally introduced by Chaum (1988) and first analysed in a temporal-epistemic setting by van der Meyden and Su (2004). The formulation presented in Section 8.2 was first discussed by Raimondi and Lomuscio (2007). A reformulation of the protocol including non trustworthy cryptographers was done by Kacprzak, Lomuscio, Niewiadomski, Penczek, Raimondi, and Szreter (2006). The original wording reported in Section 8.2 was first cited by van der Meyden and Su (2004). The train-gate-controller scenario was presented in the context of alternating temporal logic by Alur, Henzinger, Mang, Qadeer, Rajamani, and Tasiran (1998) and recast in terms of epistemic properties by van der Hoek and Wooldridge (2002b). The description adopted here was encoded as an interpreted system in a paper by Jones and Lomuscio (2010). The Write-Once cache coherence protocol described by Baukus and van der Meyden (2004) and by Archibald and Baer (1986), used as a benchmark in Section 8.6, was encoded as an interleaved interpreted system by Lomuscio et al. (2010).

Research on verification of epistemic specifications by model checking was spurred by Halpern and Vardi (1991). At the time theorem proving was the leading verification technology and approaches to verifying epistemic specifications relied on various proof-theoretical techniques for the epistemic logic $S5_n$ enriched by operators for group knowledge. A case in point is the well-known BAN approach put forward by Burrows, Abadi, and Needham (1990) whereby authentication properties could be shown, it

was argued, through automatic proof procedures based on axiomatisations inspired by epistemic logic. Halpern and Vardi (1991) presented the rationale for verifying systems against epistemic properties by model theoretic approaches instead. While in the key example discussed in the paper (the muddy children problem) the temporal evolution is captured through updates of static epistemic models, various remarks on the possible theoretical advantage of model checking procedures over theorem proving are given and used to motivate research into model-theoretical approaches.

Halpern and Vardi (1991) initially spurred considerable research in the area of updates for static epistemic models, like the work by Baltag, Moss, and Solecki (1998), Lomuscio and Ryan (1999), and by Gerbrandy (1999). As most of the updates first studied concerned announcements and information sharing, this in turn lead to considerable research on logics for public announcements, as first suggested by Plaza (1989). As far as we are aware, it was not before the early 2000s that logics of knowledge and time were used as specification languages on which model checking procedures could be applied to. Ten years on from then, while Halpern and Vardi (1991) only expressed the view that "In summary, we do not expect the model checking approach to supplement the theorem proving approach." (p. 19), it appears that model checking (see the volume by Clarke, Grumberg, and Peled (1999)) has now become the de-facto technique for the verification of systems against temporal-epistemic specifications.

This development was profoundly influenced by at least two decades of successful research in model checking for purely temporal specifications where several techniques have been put forward with the aim of mitigating the state-explosion problem. The most prominent methodologies include partial order reductions as proposed by Valmari (1990), Godefroid (1991), and by Peled (1993), symmetry reductions as studied by Clarke, Filkorn, and Jha (1993), Emerson and Jutla (1993), and by Emerson and Sistla (1995), ordered-binary decision diagrams as described by Burch, Clarke, McMillan, Dill, and Hwang (1990) and by McMillan (1993), bounded and unbounded model checking as suggested by Biere, Cimatti, Clarke, and Zhu (1999) and by McMillan (2002), and various forms of abstraction as, among others, discussed by Dams, Gerth, Dohmen, Herrmann, Kelb, and Pargmann (1994) and by Ball, Podelski, and Rajamani (2001). A number of tools have been released implementing these techniques; some of these have reached a high level of maturity and are used in industrial settings. These include SPIN, see the work of Holzmann (1997), which adopts an on-the-fly automata-based approach combined with partial-order reductions for LTL specifications and NuSMV, for which we refer to work by McMillan (1993) and Cimatti, Clarke, Giunchiglia, and Roveri (1999), which used OBDDs and bounded model checking for symbolic model checking against

LTL and CTL specifications. Other implementations are available for either the same or different classes of temporal specifications including real-time logics, probabilistic logic, etc. The work reported in this paper is influenced by the successful methodologies employed against temporal specifications.

To our knowledge the first treatment of model checking against epistemic specifications, in the sense discussed in this chapter, was undertaken by van der Meyden and Shilov (1999a) where the problem is formalised and complexity results for perfect recall semantics are given. In what follows we briefly summarise other notable approaches to the verification of systems against temporal-epistemic specification and relate them to the material presented earlier. Due to the recent growth in the area the discussion is incomplete.

Reduction-based approaches. One of the first approaches to model checking temporal-epistemic logic that we are aware of was put forward by van der Hoek and Wooldridge (2002a, 2003), who proposed a reduction from temporal-epistemic specifications to plain temporal specifications. The approach uses local propositions to identify and fully characterise the local states of the agents. A feature of this approach is that no automatic procedure for the automatic synthesis of local propositions from the model is given. As stated by the authors, the approach is inspired by Engelhardt, van der Meyden, and Moses (1998); in their paper a logic for local proposition is developed, and the basic principles through which epistemic formulas can be rewritten via quantification of local propositions are put forward. It should be noted that, differently from van der Hoek and Wooldridge (2002a), the technique of Engelhardt et al. (1998) focuses on representational issues in static epistemic logic. In this context it is shown that negative results such as non-axiomatisability and undecidability follow in some settings because of reductions to second-order logic even when assuming a weak semantics.

More recently, temporal-epistemic logic on discrete time was reduced to a special case of action-restricted CTL (ARCTL for short) by Lomuscio, Pecheur, and Raimondi (2007b). ARCTL as proposed by Pecheur and Raimondi (2007) is an extension of CTL whereby path quantifiers are labelled with actions. The work of Lomuscio, Pecheur, and Raimondi (2007a) concerns a reduction of CTLK to ARCTL and a compiler into NuSMV implementing the translation. In this work the relations corresponding to the epistemic modalities are effectively recast as special actions in the corresponding action-based transition system and special labels are used for the temporal relations. Experimental results on the dining cryptographers showed that the approach was as efficient as the other toolkits available at the time.

OBDD-based approaches. The MCMAS toolkit developed by Raimondi and Lomuscio (2007), Lomuscio and Raimondi (2006b), and by Lo-

muscio, Qu, and Raimondi (2009) described in Section 8.3 implements interpreted systems semantics and supports the verification of systems against not only temporal-epistemic specifications but also properties based on deontic concepts as done by Raimondi and Lomuscio (2004a) and by Woźna, Lomuscio, and Penczek (2005a), explicit knowledge as discussed by Lomuscio, Raimondi, and Woźna (2007) and ATL cooperation primitives as put forward by Lomuscio and Raimondi (2006c). It is implemented in C++ and relies on the latest BDD package of Somenzi (2005). An in-depth description of the tool is given by Raimondi (2006). More details on OBDDs and related techniques can be found in papers by Bryant (1986) and by Huth and Ryan (2000).

Van der Meyden and colleagues were the first to propose and implement the use of OBDD-based model checking in a temporal-epistemic setting, see the paper by Gammie and van der Meyden (2004). These were preceded by theoretical studies on the computational complexity of the model checking problem by, e.g., van der Meyden and Shilov (1999b); further results on this can be found in work by Lomuscio and Raimondi (2006a), by Engelhardt, Gammie, and van der Meyden (2007), and by Huang and van der Meyden (2010). Like MCMAS, MCK has undergone several versions since its original version. At the time of writing it supports CTL*K specifications and a variant of probabilistic knowledge, i.e., epistemic modalities defined on probabilistic systems as by Huang, Luo, and Meyden (2011). MCK is released in binary form and is implemented in Haskell by using using Long's BDD library. A difference with respect to MCMAS and VerICS is that MCK includes specialised implementations for different evolution semantics including perfect recall and clock semantics. While these can also be encoded on specific examples with MCMAS and VerICS it is possible that specialised semantics can provide the user with efficiency gains; no comparison has been made in this respect so far.

A further BDD-based checker, MCTK, has recently been released as described by Luo (2009). MCTK follows the local propositions approach discussed above to reduce the verification of epistemic specifications to temporal formulas only. MCTK uses NuSMV as the underlying temporal checker. Differently from the approach of van der Hoek and Wooldridge (2002a), a technique for the automatic calculation of local propositions is here given by Su (2004). Experimental results against other checkers are not available at the time of writing.

SAT-based approaches. The SAT-based BMC approach presented in Section 8.4.1 predates the OBDD-based approaches here described. While techniques using OBDDs became prevalent in the ten years up to 2010, there has recently been an increased activity on SAT-based methods. The VerICS tool of Kacprzak, Nabialek, Niewiadomski, Penczek, Pólrola, Szreter,

Woźna, and Zbrzezny (2008) and by Penczek and Półrola (2006), described in Section 8.4, implements timed automata in the sense of Alur and Dill (1994) and interleaved interpreted systems semantics as proposed in Lomuscio, Penczek, and Qu (2010) to support the SAT-based verification of systems against not only temporal-epistemic specifications but also properties based on deontic concepts, see the work of Woźna, Lomuscio, and Penczek (2005a), and of Woźna, Lomuscio, and Penczek (2005b), and real time systems as described by Lomuscio, Woźna, and Penczek (2007).

SAT-based BMC was originally introduced for the verification of LTL specifications by Biere et al. (1999) and by Biere, Cimatti, Clarke, Strichman, and Zhu (2003) as an alternative to approaches based on OBDDs. Then, BMC was defined for ECTL - the existential fragment of CTL, first by Penczek, Woźna, and Zbrzezny (2002) and then refined by Zbrzezny (2008) such that a specific symbolic k-path is allocated to each subformula of the tested formula starting with a modality. Moreover, reduced Boolean circuits as described by Abdulla, Bjesse, and Eén (2000) are used in the encoding of Zbrzezny (2008). A reduced Boolean circuit represents subformulas of the encoding by fresh propositions such that each two identical subformulas correspond to the same proposition. BMC for ECTL was extended to ECTLK by Penczek and Lomuscio (2003) and further to ECTLKD by Woźna, Lomuscio, and Penczek (2005a). The solution of Zbrzezny (2008) for ECTL was extended by Huang, Luo, and van der Meyden (2010) to ECTLK, but without using reduced Boolean circuits.

The approach based on UMC, discussed in Section 8.4.2, was originally introduced by McMillan (2002) for the verification of CTL as an alternative to BMC and the approaches based on BDDs. Then, UMC was extended to $\mathrm{CTL_pK}$ by Kacprzak, Lomuscio, and Penczek (2004). The reader is referred to (Kacprzak, Lomuscio, and Penczek, 2003) for more details on UMC, especially on computing the fix-points over propositional representations of sets. The standard algorithm toCNF transforming a propositional formula to one in CNF, preserving satisfiability only, was presented by McMillan (2002) and by Penczek and Półrola (2006). The translation *equCNF* to an equivalent propositional formula in CNF was given by Penczek and Półrola (2006). We refer the reader to work by Chauhan, Clarke, and Kroening (2003) and by Ganai, Gupta, and Ashar (2004), where alternative solutions can be found. Blocking clauses used in the algorithm *equCNF* can be computed using the methods discussed by McMillan (2002) and by Szreter (2006, 2005).

In addition to the BMC approaches for extensions of CTLK discussed in this chapter, BMC approaches for LTLK (Meşki, Penczek, Szreter, Woźna-Szcześniak, and Zbrzezny, 2014) have been put forward, like for instance by Penczek, Woźna-Szcześniak, and Zbrzezny (2012), using a translation to

SAT and also by Meşki, Penczek, and Szreter (2012), using operations on BDDs. Moreover, the latest release of the MCK checker reported above, now supports BMC through SAT. Experimental results tend to show that OBDDs and SAT-based BMC methods are complementary to one another with BMC working better for reachability and checking small epistemic specifications on very large models, and OBDDs outperforming BMC with complex specifications and models with reachable state spaces in the region of 10^6 to 10^{10}.

Abstraction. While OBDDs and SAT-based approaches can be very effective for representing large state spaces, the state-space grows exponentially with the number of variables in the system. To alleviate this difficulty various forms of abstraction have been put forward to verify systems against temporal-epistemic specifications. The first abstraction technique that we are aware of in this context was developed by Enea and Dima (2007), where a number of abstractions for Kripke models with epistemic relations are defined. The methodology is defined for a specification language based on branching-time temporal (both past and future) and epistemic modalities, interpreted on a Kleene's three valued semantics. This enables the authors to give weak-preservation and error-preservation results for temporal-epistemic specifications with respect to the three-valued semantics given. Somewhat related to this is the approach by Dechesne, Orzan, and Wang (2008) where a notion of refinement is developed in the context of public announcement logic, thereby enabling the authors to give abstractions for which a preservation theorem can be shown. In this case, however, two-valued interpretations are used.

While Enea and Dima (2007) and Dechesne, Orzan, and Wang (2008) developed their work for Kripke models, other techniques have adopted a modular, agent-based view and used interpreted systems as the underlying semantics for Kripke models. The first approach following this line appears to be that of Cohen, Dam, Lomuscio, and Russo (2009),where an existential abstraction technique is developed and a preservation theorem shown. The approach involves taking a quotient of an interpreted system by defining abstract local states, actions, protocols and the transition relation on the abstract model. This enables the authors to show that if a specification in the universal fragment of CTLK holds in the abstract model, it also holds in the concrete one. The approach was later extended by Lomuscio, Qu, and Russo (2010), where a data abstraction methodology for interpreted systems specified against CTLK formulas was put forward and implemented in conjunction with MCMAS. More recently Al-Bataineh and van der Meyden (2011), presented abstraction results applicable to the verification of the dining cryptographers scenario and applied them to knowledge-based programs.

Symmetry reduction is a well-established form of abstraction whereby symmetry considerations are exploited to produce abstract models preserving a given specification. Cohen, Dam, Lomuscio, and Qu (2009b) use a counterpart semantics to interpret epistemic modalities on abstract models by means of symmetry considerations. Experimental results presented by Cohen, Dam, Lomuscio, and Qu (2009b) show a linear reduction in the memory requirements for BDD-based verification. While they relied on manual identification of symmetries, Cohen, Dam, Lomuscio, and Qu (2009a) presented an automatic technique for the reduction and application to data symmetry as well. The technique was implemented on an ad-hoc, extended version of ISPL system descriptions; the benchmarks reported showed an exponential reduction in the time and memory footprint in some scenarios amenable to symmetry reduction.

Partial order reductions have extensively been used for the verification of reactive systems specified against $\mathrm{LTL}_{-\mathrm{X}}$ by e.g., Peled (1994) and against $\mathrm{CTL}_{-\mathrm{X}}$ formulas by Gerth, Kuiper, Peled, and Penczek (1999) and by Penczek, Szreter, Gerth, and Kuiper (2000). Lomuscio, Penczek, and Qu (2009) present theoretical results in the context of interpreted systems and temporal-epistemic logic. In Section 8.6 we summarised the work of Lomuscio, Penczek, and Qu (2010), as this is the only approach we are aware of which has been implemented and shown to be efficient. Traditionally, in partial order reductions the exploration is carried out by depth-first-search (DFS), as done by e.g., Gerth, Kuiper, Peled, and Penczek (1999), or by double-depth-first-search, as done by e.g., Courcoubetis, Vardi, Wolper, and Yannakakis (1992). The conditions $C1-C3$, used in the algorithm, are inspired by Peled (1993) and by Gerth, Kuiper, Peled, and Penczek (1999).

Optimised algorithms. Some results in the literature have focused on novel, optimised algorithms for the verification of temporal-epistemic specifications. Sometimes these algorithms are distributed or parallel. For example, Kwiatkowska, Lomuscio, and Qu (2010) present parallel versions of the labelling algorithms for the automatic verification of temporal-epistemic properties. The results point to a significant speed-up in the labelling of formulas although the performance is strongly dependent on the number of cores available and the type of specification to be checked. The work by Jones and Lomuscio (2010), discussed above in the context of the combination between BMC and OBDDs, also includes a distributed algorithm for bounded satisfaction based on the notion of seed states for state-space partitioning. In a different context Cohen and Lomuscio (2010) put forward an algorithm for the non-elementary speed-up of model checking synchronous systems with perfect recall. An improved encoding for the BMC problem via SAT, which was shown to generate a polynomially smaller number of propositions in the encodings thereby allowing faster verification times, was

presented by Huang, Luo, and Meyden (2010). An approach to synthesising groups of agents satisfying an epistemic specification on a given system was explored by Jones, Knapik, Lomuscio, and Penczek (2012).

Extensions to other agent-based specifications. Model checking approaches have also been investigated for specifications richer than the temporal-epistemic logics discussed here. Raimondi and Lomuscio (2004b) presented an OBDD-based approach to the verification of deontic interpreted systems (Lomuscio and Sergot, 2003); the BMC case was analysed by Woźna, Lomuscio, and Penczek (2005a). Deontic interpreted systems are a formalism for the representation of correct functioning behaviour of the agents in a system. Local states are partitioned into correct and incorrect ones, and a further agent-index modality representing "at all the correct states for agent i" is introduced. The modality is interpreted by considering the global states in which the agent in question is operating correctly. By means of this formalism, one can analyse scenarios where some agents may display faulty behaviour. For example, the properties of a variant of the dining cryptographers scenario where some cryptographers are intruders saying the opposite of what they should was verified through this formalism by Kacprzak, Lomuscio, Niewiadomski, Penczek, Raimondi, and Szreter (2006).

Extensions to epistemic logic to include explicit knowledge have also been discussed and implemented, e.g., by Lomuscio and Woźna (2006) and by Lomuscio, Raimondi, and Woźna (2007). Both VerICS and MCMAS support these features.

Model checking of systems against alternating-temporal logic (ATL) specifications has been pursued by Alur, Henzinger, and Kupferman (2002). ATL extends CTL by adding strategies in the semantics and explicit constructs for representing what groups of agents can enforce. Its model checking problem is considerably harder under partial observability. MOCHA, see (Alur, de Alfaro, Henzinger, Krishnan, Mang, Qadeer, Rajamani, and Tasiran, 2000) is an explicit state model checker supporting ATL modalities. Even if strategies and knowledge can interact in subtle ways as argued by Jamroga and van der Hoek (2004)[3], progress has been made both in the definition of combinations between ATL with knowledge and their verification. Specifically, Lomuscio and Raimondi (2006c) put forward a symbolic, OBDD-based model checking algorithm for the verification of ATLK specifications and discussed experimental results. An alternative approach using MOCHA is discussed by Wooldridge, Agotnes, Dunne, and van der Hoek (2007) in combination with the local propositions construction referenced above.

[3]For a detailed discussion on this interaction, see Chapter 11.

Epistemic concepts have been used in a broader context to reason about multi-agent systems modelled by other attitudes (such as norms, beliefs, desires, goals, or intentions). These properties are often treated simply as propositions in a temporal language and not as first-citizens like the modalities discussed above. Given this, they are technically very different and not discussed here.

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.

Acknowledgements Much of the work described in this chapter is based on joint research with Magdalena Kacprzak, Hongyang Qu, Franco Raimondi, Maciej Szreter, Boèna Woźna Szcześniak, and Andrzej Zbrzezny.

We would like to thank Catalin Dima, Masoud Koleini and Ji Ruan for valuable feedback on a preliminary version of this chapter.