

# Bounded Model Checking Linear Time and Knowledge Using Decision Diagrams <sup>★</sup>

Artur Męski<sup>1,2</sup>, Wojciech Penczek<sup>2,3</sup>, and Maciej Szreter<sup>2</sup>

<sup>1</sup> University of Łódź, FMCS, Banacha 22, 90-238 Łódź, Poland

<sup>2</sup> Institute of Computer Science, PAS, Ordona 21, 01-237 Warsaw, Poland

<sup>3</sup> University of Natural Sciences and Humanities, Institute of Informatics,  
3 Maja 54, 08-110 Siedlce, Poland

{meski,penczek,mszreter}@ipipan.waw.pl

**Abstract.** We present a novel approach to verification of multi-agent systems by bounded model checking for LTLK, i.e., Linear Time Temporal Logic extended with the epistemic component, which is interpreted over interleaved interpreted systems. Our method is based on binary decision diagrams. We describe the algorithm and provide its experimental evaluation together with the comparison with an existing tool.

## 1 Introduction

It is often crucial to ensure that multi-agent systems (MAS) conform to their specifications and exhibit some desired behaviour. This can be checked in a fully automatic manner using model checking [5], which is one of the rapidly developing verification techniques. Model checking has been studied by various researchers in the context of MAS and different modal logics for specifying MAS properties [2, 8, 9, 14, 16, 17, 21, 22].

When the verification is performed by searching directly through the state space of the MAS, its size is likely to grow exponentially with the number of agents. Therefore, several approaches alleviating this so called state-space explosion problem have been proposed. One of them is *bounded model checking* (BMC) [1], in which only a portion of the original model truncated up to some specific depth is considered. This approach can be combined with a technique which involves the translation of the verification problem to the boolean satisfiability problem (SAT) [19, 14], or with symbolic techniques based on *binary decision diagrams* (BDDs) [12]. A SAT-based BMC for the verification the LTLK properties of MAS, i.e., LTL augmented with the epistemic component (also called CKL<sub>n</sub> [9]) has been defined ([20]) independently to our solution and submitted to CS&P'11 as well. In this paper we propose an alternative and fully novel approach. We define a BDD-based BMC method for LTLK interpreted over interleaved interpreted systems (IIS) [15], which to our knowledge is the first such attempt.

---

<sup>★</sup> Partly supported by the Polish Ministry of Science and Higher Education under the grant No. N N206 258035.

The rest of the paper is organised as follows. Sect. 2 provides the basic definitions and notations for LTLK and IIS. Sect. 3 describes our BDD-based BMC method. The last two sections contain an experimental evaluation of the approach together with its discussion as well as the final remarks.

## 2 Preliminaries

In this section we introduce the basic definitions used in the paper. In particular, we define the semantics of interpreted systems, as well as the syntax and the semantics of LTLK.

### 2.1 Interleaved interpreted systems

The semantics of *interpreted systems* provides a setting to reason about MAS by means of specifications based on knowledge and linear or branching time. We report here the basic setting as popularised in [7], restricted to interleaved interpreted systems [15]. Therefore, we assume that if more than one agent is active at a given state, all the active agents perform the same (shared) action in the round. Note that it is still possible for agents to communicate by means of shared actions.

We begin by assuming a MAS to be composed of  $n$  agents<sup>1</sup>  $\mathcal{A}$ . We associate a set of *possible local states*  $L_i$  and *actions*  $Act_i$  to each agent  $i \in \mathcal{A}$ . We assume that the special action  $\epsilon_i$ , called “null”, or “silent” action of agent  $i$  belongs to  $Act_i$ ; as it will be clear below the local state of agent  $i$  remains the same if the null action is performed. Also note that we do not assume that the sets of actions of the agents to be disjoint. We call  $Act = \bigcup_{i \in \mathcal{A}} Act_i$  the union of all the sets  $Act_i$ . For each action  $a$  by  $Agent(a) \subseteq \mathcal{A}$  we mean all the agents  $i$  such that  $a \in Act_i$ , i.e., the set of agents potentially able to perform  $a$ . Following closely the interpreted system model, we consider a *local protocol* modelling the program the agent is executing. Formally, for any agent  $i$ , the actions of the agents are selected according to a *local protocol*  $P_i : L_i \rightarrow 2^{Act_i}$ ; we assume that  $\epsilon_i \in P_i(l)$ , for any  $l \in L_i$ , i.e., we insist on the null action to be enabled at every local state. For each agent  $i$ , we define an evolution (partial) function  $t_i : L_i \times Act_i \rightarrow L_i$ , where  $t_i(l, \epsilon_i) = l$  for each  $l \in L_i$ . The local transition function considered here differs from the standard treatment in interpreted systems by having the local action as the only parameter. A *global state*  $g = (l_1, \dots, l_n)$  is a tuple of local states for all the agents in the MAS corresponding to an instantaneous snapshot of the system at a given time. Given a global state  $g = (l_1, \dots, l_n)$ , we denote by  $g^i = l_i$  the local component of agent  $i \in \mathcal{A}$  in  $g$ . Let  $G$  be a set of global states. The *global interleaved evolution* function  $t : G \times \prod_{i=1}^n Act_i \rightarrow G$  is defined as follows:  $t(g, a_1, \dots, a_n) = g'$  iff there exists an action  $a \in Act \setminus \{\epsilon_1, \dots, \epsilon_n\}$  such that for all  $i \in Agent(a)$ ,  $a_i = a$  and  $t_i(g^i, a) = g'^i$ , and for all  $i \in \mathcal{A} \setminus Agent(a)$ ,  $a_i = \epsilon_i$  and  $t_i(g^i, a_i) = g'^i$ . In brief we write the above as  $g \xrightarrow{a} g'$ .

<sup>1</sup> Note in the present study we do not consider the environment component. This may be added with no technical difficulty at the price of heavier notation.

Similar to blocking synchronisation in automata, the above insists on all agents performing the same non-epsilon 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. This assumes local protocols are defined in such a way to permit this; if a local protocol does not allow this, the local action cannot be performed and therefore the global transition does not comply with the definition of interleaving above. As we formally clarify below, we only consider interleaved transitions here.

We assume that the global transition relation is total, i.e., that for any  $g \in G$  there exists an  $a \in Act$  such that  $g \xrightarrow{a} g'$  for some  $g' \in G$ . An **infinite** sequence of global states and actions  $\rho = g_0 a_0 g_1 a_1 g_2 \dots$  is called an *interleaved path* (or simply a *path*) originating at  $g_0$  if there is a sequence of interleaved transitions from  $g_0$  onwards, i.e.,  $g_i \xrightarrow{a_i} g_{i+1}$  for every  $i \geq 0$ . Any **finite** prefix of a path is called an *(interleaved) run*. By  $length(\rho)$  we mean the number of the states of  $\rho$  if  $\rho$  is a run, and  $\omega$  if  $\rho$  is a path. The set of all the interleaved paths and runs originating from  $g$  is denoted by  $\Pi(g)$ . The set of all the interleaved paths originating from  $g$  is denoted by  $\Pi^\omega(g)$ . A state  $g$  is said to be *reachable* from  $g_0$  if there is a path or a run  $\rho = g_0 a_0 g_1 a_1 g_2 \dots$  such that  $g = g_i$  for some  $i \geq 0$ .

For each agent  $i \in \mathcal{A}$ ,  $\sim_i \subseteq G \times G$  is an *epistemic indistinguishability* relation over global states defined by  $g \sim_i r$  if  $g^i = r^i$ . Further, let  $\Gamma \subseteq \mathcal{A}$ . The union of  $\Gamma$ 's accessibility relations is defined as  $\sim_\Gamma^E = \bigcup_{i \in \Gamma} \sim_i$ . By  $\sim_\Gamma^C$  we denote the transitive closure of  $\sim_\Gamma^E$ , whereas  $\sim_\Gamma^D = \bigcap_{i \in \Gamma} \sim_i$ .

**Definition 1 (Interleaved Interpreted Systems).** *Given a set of propositions  $\mathcal{PV}$  such that  $true, false \in \mathcal{PV}$ , an interleaved interpreted system (IIS), also referred to as a model, is a tuple  $M = (G, \iota, \Pi, \{\sim_i\}_{i \in \mathcal{A}}, \mathcal{V})$ , where  $G$  is a set of global states,  $\iota \in G$  is an initial (global) state such that each state in  $G$  is reachable from  $\iota$ ,  $\Pi = \bigcup_{g \in G} \Pi(g)$  is the set of all the interleaved paths and runs originating from all states in  $G$ , and  $V : G \rightarrow 2^{\mathcal{PV}}$  is a valuation function.*

By  $\Pi^\omega$  we denote the set of all the interleaved paths of  $\Pi$ .

## 2.2 Syntax and Semantics of LTLK

Combinations of linear time with knowledge have long been used in the analysis of temporal epistemic properties of systems [7]. We now recall the basic definitions here and adapt them to our purposes when needed.

**Definition 2 (Syntax).** *Let  $\mathcal{PV}$  be a set of atomic propositions to be interpreted over the global states of a system,  $p \in \mathcal{PV}$ , and  $\Gamma \subseteq \mathcal{A}$ . Then, the syntax of LTLK is defined by the following BNF grammar:*

$$\begin{aligned} \varphi ::= & p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi \mid \varphi R \varphi \mid \\ & K_i \varphi \mid \bar{K}_i \varphi \mid E_\Gamma \varphi \mid \bar{E}_\Gamma \varphi \mid D_\Gamma \varphi \mid \bar{D}_\Gamma \varphi \mid C_\Gamma \varphi \mid \bar{C}_\Gamma \varphi. \end{aligned}$$

The temporal operators U and R are named as usual *until* and *release* respectively, X is the next step operator. The operator  $K_i$  represents “agent  $i$  knows”

and  $\bar{K}_i$  is the corresponding dual representing “agent  $i$  does not know whether or not something holds”. The epistemic operators  $D_\Gamma, E_\Gamma$ , and  $C_\Gamma$  represent distributed knowledge in the group  $\Gamma$ , “everyone in  $\Gamma$  knows”, and common knowledge among agents in  $\Gamma$ .  $\bar{D}_\Gamma, \bar{E}_\Gamma$ , and  $\bar{C}_\Gamma$  are the corresponding dual ones.

Typically, the semantics of LTLK is defined over paths of a model  $M$  only, whereas our semantics exploits paths and runs. This semantics can be conveniently applied also to submodels (to be defined later) in order to verify efficiently the existential fragment of LTLK over paths and runs.

**Definition 3 (Semantics).** *Given a model  $M = (G, \iota, \Pi, \{\sim_i\}_{i \in \mathcal{A}}, \mathcal{V})$ , where  $\mathcal{V}(s)$  is the set of propositions that hold at  $s$ . Let  $\rho(i)$  denote the  $i$ -th state of a path or run  $\rho \in \Pi$ , and  $\rho[i]$  denote the path or run  $\rho$  with a designated formula evaluation position  $i$ , where  $i \leq \text{length}(\rho)$ . Note that  $\rho[0] = \rho$ . The formal semantics of LTLK is defined recursively as follows:*

- $M, \rho[i] \models p$  iff  $p \in \mathcal{V}(\rho(i))$ ;
  - $M, \rho[i] \models \neg\varphi$  iff  $M, \rho[i] \not\models \varphi$ ;
  - $M, \rho[i] \models \varphi_1 \wedge \varphi_2$  iff  $M, \rho[i] \models \varphi_1$  and  $M, \rho[i] \models \varphi_2$ ;
  - $M, \rho[i] \models \varphi_1 \vee \varphi_2$  iff  $M, \rho[i] \models \varphi_1$  or  $M, \rho[i] \models \varphi_2$ ;
  - $M, \rho[i] \models X\varphi$  iff  $\text{length}(\rho) > i$  and  $M, \rho[i+1] \models \varphi$ ;
  - $M, \rho[i] \models \varphi_1 U \varphi_2$  iff  $(\exists k \geq i)[M, \rho[k] \models \varphi_2$  and  $(\forall i \leq j < k) M, \rho[j] \models \varphi_1]$ ;
  - $M, \rho[i] \models \varphi_1 R \varphi_2$  iff  $[(\rho \in \Pi^\omega(\iota)$  and  $(\forall k \geq i) M, \rho[k] \models \varphi_2)$  or  $(\exists k \geq i)[M, \rho[k] \models \varphi_1$  and  $(\forall i \leq j \leq k) M, \rho[j] \models \varphi_2]$ ;
  - $M, \rho[i] \models K_i \varphi$  iff  $(\forall \rho' \in \Pi^\omega(\iota))(\forall k \geq 0)[\rho'(k) \sim_i \rho(i)$  implies  $M, \rho'[k] \models \varphi]$ ;
  - $M, \rho[i] \models \bar{K}_i \varphi$  iff  $(\exists \rho' \in \Pi(\iota))(\exists k \geq 0)[\rho'(k) \sim_i \rho(i)$  and  $M, \rho'[k] \models \varphi]$ ;
  - $M, \rho[i] \models Y \varphi$  iff  $(\forall \rho' \in \Pi^\omega(\iota))(\forall k \geq 0)[\rho'(k) \sim_Y^Y \rho(i)$  implies  $M, \rho'[k] \models \varphi]$ ,
  - $M, \rho[i] \models \bar{Y} \varphi$  iff  $(\exists \rho' \in \Pi(\iota))(\exists k \geq 0)[\rho'(k) \sim_Y^Y \rho(i)$  and  $M, \rho'[k] \models \varphi]$ ,
- where  $Y \in \{D, E, C\}$ .

Let  $g \in G$  and  $\varphi$  be an LTLK formula. We use the following notations:

- $M, g \models \varphi$  iff  $M, \rho[0] \models \varphi$  for all the paths  $\rho \in \Pi^\omega(g)$ ;
- $M \models \varphi$  iff  $M, \iota \models \varphi$ ;
- $\text{Props}(\varphi)$  is the set of atomic propositions appearing in  $\varphi$ .

LTL is the sublogic of LTLK which consists only of the formulae built without the epistemic operators. ELTLK is the existential fragment of LTLK, defined by the following grammar:

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi \mid \varphi R \varphi \mid \bar{K}_i \varphi \mid \bar{E}_\Gamma \varphi \mid \bar{D}_\Gamma \varphi \mid \bar{C}_\Gamma \varphi.$$

Moreover, an ELTLK formula  $\varphi$  holds in the model  $M$ , denoted  $M \models_{\exists} \varphi$ , iff  $M, \rho[0] \models \varphi$  for some path or run  $\rho \in \Pi(\iota)$ . The intuition behind this definition is that ELTLK is obtained only by restricting the syntax of the epistemic operators while the temporal ones remain the same. We get the existential version of these operators by the change from the universal quantification over the paths ( $\models$ ) to the existential quantification ( $\models_{\exists}$ ) over the paths and the runs in the definition of the validity in the model  $M$ . Notice that this change is only necessary when  $\varphi$  contains a temporal operator, which is not nested in an epistemic operator.

Our semantics meets two important properties. Firstly, for LTL the definition of validity in a model  $M$  uses paths only. Secondly, if we replace each  $\Pi$  with  $\Pi^\omega$ , the semantics does not change as our models have total transition relations (each run is a prefix of some path). The semantics applied to submodels of  $M$  does not have the above property, but it preserves ELTLK over  $M$ , which is shown in Lemma 2.

### 3 BDD-based BMC for ELTLK

In this section we show how to perform BMC of ELTLK using BDDs [5] by combining the standard approach for ELTL [4] with the method for the epistemic operators [21] in a similar manner to the solution for CTL\* of [5].

Let  $\mathcal{PV}$  be a set of propositional variables. For an ELTLK formula  $\varphi$  we define inductively the *number  $\gamma(\varphi)$  of nested epistemic operators* in the formula:

- if  $\varphi = p$ , where  $p \in \mathcal{PV}$ , then  $\gamma(\varphi) = 0$ ,
- if  $\varphi = \odot\varphi'$  and  $\odot \in \{\neg, X\}$ , then  $\gamma(\varphi) = \gamma(\varphi')$ ,
- if  $\varphi = \varphi' \odot \varphi''$  and  $\odot \in \{\wedge, \vee, U, R\}$ , then  $\gamma(\varphi) = \gamma(\varphi') + \gamma(\varphi'')$ ,
- if  $\varphi = Y\varphi'$  and  $Y \in \{\bar{K}_i, \bar{E}_\Gamma, \bar{D}_\Gamma, \bar{C}_\Gamma\}$ , then  $\gamma(\varphi) = \gamma(\varphi') + 1$ .

Let  $Y$  be some epistemic ELTLK operator, i.e.,  $Y \in \{\bar{K}_i, \bar{E}_\Gamma, \bar{D}_\Gamma, \bar{C}_\Gamma\}$ . If  $\varphi = Y\psi$  is an ELTLK formula, by *sub*( $\varphi$ ) we denote the formula  $\psi$  nested in the epistemic operator  $Y$ . Moreover, for an arbitrary ELTLK formula  $\varphi$  we define inductively the set  $\mathcal{Y}(\varphi)$  of its subformulae in the form  $Y\psi$ :

- if  $\varphi = p$ , where  $p \in \mathcal{PV}$ , then  $\mathcal{Y}(\varphi) = \emptyset$ ,
- if  $\varphi = \odot\varphi'$  and  $\odot \in \{\neg, X\}$ , then  $\mathcal{Y}(\varphi) = \mathcal{Y}(\varphi')$ ,
- if  $\varphi = \varphi' \odot \varphi''$  and  $\odot \in \{\wedge, \vee, U, R\}$ , then  $\mathcal{Y}(\varphi) = \mathcal{Y}(\varphi') \cup \mathcal{Y}(\varphi'')$ ,
- if  $\varphi = Y\varphi'$  and  $Y \in \{\bar{K}_i, \bar{E}_\Gamma, \bar{D}_\Gamma, \bar{C}_\Gamma\}$ , then  $\mathcal{Y}(\varphi) = \mathcal{Y}(\varphi') \cup \{\varphi\}$ .

**Definition 4 (Submodel).** Let  $M = (G, \iota, \Pi, \{\sim_i\}_{i \in \mathcal{A}}, \mathcal{V})$  and  $U \subseteq G$  with  $\iota \in U$ . The submodel generated by  $U$  is a tuple  $M|_U = (U, \iota, \Pi', \{\sim'_i\}_{i \in \mathcal{A}}, \mathcal{V}')$ , where:  $\sim'_i = \sim_i \cap U^2$  for each  $i \in \mathcal{A}$ ,  $\mathcal{V}' = \mathcal{V} \cap U^2$ , and  $\Pi'$  is the set of the paths and runs of  $M$  having all the states in  $U$ , formally,  $\Pi' = \{\rho \in \Pi \mid (\forall 0 \leq i \leq \text{length}(\rho)) \rho(i) \in U\}$ .

For ELTLK formulae  $\varphi, \psi$ , and  $\psi'$ , by  $\varphi[\psi \leftarrow \psi']$  we denote the formula  $\varphi$  in which every occurrence of  $\psi$  is replaced with  $\psi'$ . Let  $M = (G, \iota, \Pi, \{\sim_i\}_{i \in \mathcal{A}}, \mathcal{V})$  be a model, then by  $\mathcal{V}_M$  we understand the valuation function  $\mathcal{V}$  of the model  $M$ , and by  $G_R \subseteq G$  the set of its reachable states. Moreover, we define  $\llbracket M, \varphi \rrbracket = \{g \in G_R \mid M, g \models \exists \varphi\}$ .

Given a model  $M = (G, \iota, \Pi, \{\sim_i\}_{i \in \mathcal{A}}, \mathcal{V})$ , and an ELTLK formula  $\varphi$ , Algorithm 1 is used to compute the set  $\llbracket M, \varphi \rrbracket$ , under the assumption that we have the algorithms for computing this set for each  $\varphi$  being an ELTL formula or in the form  $Yp$ , where  $p \in \mathcal{PV}$ , and  $Y \in \{\bar{K}_i, \bar{E}_\Gamma, \bar{D}_\Gamma, \bar{C}_\Gamma\}$  (we use the algorithms from [4] and [21], respectively). In order to obtain this set, we construct a new model  $M_c$  together with an ELTL formula  $\varphi_c$ , as described in Algorithm 1, and

compute the set  $\llbracket M_c, \varphi_c \rrbracket$ , which is equal to  $\llbracket M, \varphi \rrbracket$  (see Lemma 1). Initially  $\varphi_c$  equals  $\varphi$ , which is an ETLTK formula, and we process the formula in stages to reduce it to an ETLTL formula by replacing with propositional variables all its subformulae containing epistemic operators. We begin by choosing some epistemic subformula  $\psi$  of  $\varphi_c$ , which consists of exactly one epistemic operator, and process it in two stages. First, we modify the valuation function of  $M_c$  such that every state initialising some path or run along which  $sub(\psi)$  holds is labelled with the new propositional variable  $p_{sub(\psi)}$ , and we replace with the variable  $p_{sub(\psi)}$  every occurrence of  $sub(\psi)$  in  $\psi$ . In the second stage, we deal with the epistemic operators having in their scopes propositional variables only. By modifying the valuation function of  $M_c$  we label every state initialising some path or run along which the modified simple epistemic formula  $\psi$  holds with a new variable  $p_\psi$ . Similarly to the previous stage, we replace every occurrence of  $\psi$  in  $\varphi_c$  with  $p_\psi$ . In the subsequent iterations, we process every remaining epistemic subformulae of  $\varphi_c$  in the same way until there are no more nested epistemic operators in  $\varphi_c$ , i.e., we obtain an ETLTL formula  $\varphi_c$ , and the model  $M_c$  with the appropriately modified valuation function. Finally, we compute the set of all reachable states of  $M_c$  that initialise at least one path or run along which  $\varphi_c$  holds (line 13).

**Lemma 1.** *Let  $M = (G, \iota, \Pi, \{\sim_i\}_{i \in \mathcal{A}}, \mathcal{V})$  be a model,  $\varphi$  an ETLTK formula, and  $g \in G$ . Then,  $M, g \models_{\exists} \varphi$  iff  $M_c, g \models_{\exists} \varphi_c$ .*

*Proof (Sketch).* The proof is by induction on the number of nested epistemic operators, where we show that  $M, g \models_{\exists} \varphi$  iff  $M', g \models_{\exists} \varphi'$  (where  $M'$  and  $\varphi'$  are obtained by the lines 4-7 of the algorithm) iff  $M_c, g \models_{\exists} \varphi_c$  (where  $M_c$  and  $\varphi_c$  are obtained by the lines 8-11 of the algorithm).

---

**Algorithm 1** The algorithm for computing  $\llbracket M, \varphi \rrbracket$

---

```

1:  $M_c := M, \varphi_c := \varphi$ 
2: while  $\gamma(\varphi_c) \neq 0$  do
3:   pick  $\psi \in \mathcal{Y}(\varphi_c)$  such that  $\gamma(\psi) = 1$ 
4:   for all  $g \in \llbracket M_c, sub(\psi) \rrbracket$  do
5:      $\mathcal{V}_{M_c}(g) := \mathcal{V}_{M_c}(g) \cup \{p_{sub(\psi)}\}$ 
6:   end for
7:    $\psi := \psi[sub(\psi) \leftarrow p_{sub(\psi)}]$ 
8:   for all  $g \in \llbracket M_c, \psi \rrbracket$  do
9:      $\mathcal{V}_{M_c}(g) := \mathcal{V}_{M_c}(g) \cup \{p_\psi\}$ 
10:  end for
11:   $\varphi_c := \varphi_c[\psi \leftarrow p_\psi]$ 
12: end while
13: return  $\llbracket M_c, \varphi_c \rrbracket$ 

```

---

To perform bounded model checking of an ETLTK formula, we use Algorithm 2. Given a model  $M$  and an ETLTK formula  $\varphi$ , the algorithm checks if

---

**Algorithm 2** The algorithm for verifying ELTLK formula  $\varphi$  in the model  $M$

---

```

1:  $Reach := \{\iota\}, New := \{\iota\}$ 
2: while  $New \neq \emptyset$  do
3:    $Next := New_{\sim}$ 
4:   if  $\iota \in \llbracket M|_{Reach}, \varphi \rrbracket$  then
5:     return  $true$ 
6:   end if
7:    $New := Next \setminus Reach$ 
8:    $Reach := Reach \cup New$ 
9: end while
10: return  $false$ 

```

---

there exists a path or run initialised in  $\iota$  on which  $\varphi$  holds, i.e., if  $M, \iota \models_{\exists} \varphi$ . For any  $X \subseteq G$  by  $X_{\sim} \stackrel{def}{=} \{g' \in G \mid (\exists g \in X)(\exists \rho \in \Pi(g)) g' = \rho(1)\}$  we define the set of the immediate successors of all the states in  $X$ . The algorithm starts with the set  $Reach$  of reachable states that initially contains only the state  $\iota$ . With each iteration the verified formula is checked (line 4), and the set  $Reach$  is extended with new states (line 8). The algorithm operates on submodels  $M|_{Reach}$  generated by the set  $Reach$  to check if the initial state  $\iota$  is in the set of states from which there is a path or run on which  $\varphi$  holds. The loop terminates if there is such a path or run in the obtained submodel, and the algorithm returns  $true$  (line 4). The search continues until no new states can be reached from the states in  $Reach$ . When we obtain the set the of reachable states, and a path or run from the initial state on which  $\varphi$  holds could not be found in any of the obtained submodels, the algorithm terminates with  $false$ .

The correctness of the results obtained by the bounded model checking algorithm is stated by the following lemma.

**Lemma 2.** *Let  $M = (G, \iota, \Pi, \{\sim_i\}_{i \in \mathcal{A}}, \mathcal{V})$  be a model, and  $\varphi$  an ELTLK formula. Then,  $M, g \models_{\exists} \varphi$  iff exists  $G' \subseteq G$  such that  $g \in G'$  and  $M|_{G'}, g \models_{\exists} \varphi$ .*

*Proof (Sketch).* “ $\Rightarrow$ ” This way the proof is obvious as we simply take  $G' = G$ . “ $\Leftarrow$ ” This way the proof is more involved. It is by induction on the number of the nested epistemic operators in a formula  $\varphi$ . Whereas the base case is straightforward ( $\varphi$  is then an LTL formula), in the induction step we use Def. 4, the semantics of ELTLK, and exploit the fact that if  $M|_{G'}, g \models_{\exists} \varphi$ , then, as  $\varphi$  is an existential formula, it holds as well in any extension of  $M|_{G'}$ , so we have  $M, g \models_{\exists} \varphi$ .

**ELTL.** To define the sets of states corresponding to the ELTL formulae that are needed in Algorithm 1, we use the method described in [4] that is based on the idea of checking the non-emptiness of Büchi automata. The method proceeds as follows. First, let  $M$  be a model, and  $\varphi$  an ELTL formula. We begin with constructing the tableau for  $\varphi$  (as described in [4]), that is then combined with the model  $M$  to obtain their product, which contains these paths of  $M$  where the formula  $\varphi$  potentially holds. Next, the product is verified in terms of the

CTL model checking of  $EG\textit{true}$  formula under fairness constraints. The fairness constraints, corresponding to sets of states, allow to choose only the paths of the model, along which at least one state in each set representing fairness constraints appears in a cycle. In case of ELTL model checking, fairness is applied to guarantee that  $\varphi U \psi$  really holds, i.e., to eliminate paths where  $\varphi$  holds continuously, but  $\psi$  never holds. Finally, we choose only these reachable states of the product that belong to some particular set of states computed for the formula. The corresponding states of the verified system that are in this set, comprise the set  $\llbracket M, \varphi \rrbracket$ , i.e., the reachable states where the verified formula holds. As we are unable to include more details (due to the page limit), we refer the reader to [4].

The method we use for ELTL has some limitations when used for BMC, where it is preferable to detect counterexamples using not only the paths but also the runs of the submodel. As the method assumes totality of the transition relation of the verified model, it allows finding counterexamples only along the paths of the model. However, this does not change the correctness of the results if the final submodel has the total transition relation: in the worst case the detection of the counterexample is delayed to the last iteration, i.e., when all the reachable states are computed. Nonetheless this should not keep us from assessing the potential efficiency of the approach, as if the method used for computing the set of states for an ELTL formula would be able to detect counterexamples sooner.

## 4 Experimental Results

In this section we consider three scalable systems which we use to evaluate performance of our BMC approach and the one of MCK.

At the time of writing this paper, we have received a new version of the tool MCK<sup>2</sup> from its authors. The manual for MCK states that the tool supports SAT-based BMC for CTL\*K. Unfortunately, no theory behind this implementation has ever been published. We are aware of the paper [11], which describes SAT-based BMC for CTLK, but it does not discuss how this approach can be extended to CTL\*K. It should be also noted that MCK implements different semantics of MAS, in which agents can perform independent actions simultaneously in a single step of the protocol, what may result in different counterexamples and their lengths than given by IIS. To allow the comparison, we ensured that for each considered benchmark, the counterexamples found by the tools are of similar size, i.e., either they are constant or their complexity is the same with respect to the number of the processes. All the benchmarks can be found at the webpage of Verics – <http://verics.ipipan.waw.pl/>, together with an instruction how to repeat our experiments. Our method is implemented with reordering, and with the fixed interleaving order of the BDD variables. The reordering is performed by the Rudell’s sifting algorithm available in the implementation of CUDD library, that we use for manipulating BDDs.

<sup>2</sup> <http://cgi.cse.unsw.edu.au/~mck/mcks/docDownload/manual>

The tests have been performed on a computer fitted with Intel Xeon 2 GHz processor and 4 GB of RAM, running Linux 2.6. The specifications for the described benchmarks are given in the universal form, for which we verify the corresponding counterexample formula, i.e., the formula which is negated and interpreted existentially. Moreover, for every specification given, there exists a counterexample. With  $i(n)$  we denote the number of iterations needed by our algorithm to find the counterexample, where  $n$  is the scaling parameter.

#### 4.1 Benchmarks

**Faulty Generic Pipeline Paradigm (FGPP)** (adapted from [18]) consists of Producer, Consumer, and a chain of  $n$  intermediate Nodes transmitting data, together with a chain of  $n$  Alarms enabled when some error occurs. We consider the following specifications:

$\varphi_1 = G(\text{ProdSend} \rightarrow K_C K_P \text{ConsReady})$ ,  $\varphi_2 = G(\text{Problem}_n \rightarrow (F(\text{Repair}_n) \vee G(\text{Alarm}_n \text{Send})))$ ,  $\varphi_3 = \bigwedge_{i=1}^n G(\text{Problem}_i \rightarrow (F(\text{Repair}_i) \vee G(\text{Alarm}_i \text{Send})))$ , and  $\varphi_4 = \bigwedge_{i=1}^n G(K_P(\text{Problem}_i \rightarrow (F(\text{Repair}_i) \vee G(\text{Alarm}_i \text{Send}))))$ . The formula  $\varphi_1$  ( $i(n) = 2n + 3$ ) states that if Producer produces a commodity, then Consumer knows that Producer does not know that Consumer has the commodity. The formula  $\varphi_2$  ( $i(n) = 2n + 4$ ) expresses that each time a problem occurs at node  $n$ , then either it is repaired or the alarm of node  $n$  rings. The formula  $\varphi_3$  ( $i(n) = 6$ ) expresses that each time a problem occurs on a node, then either it is repaired or the alarm rings. The formula,  $\varphi_4$  ( $i(n) = 6$ ) expresses that Producer knows that each time a problem occurs on a node, then either it is repaired or the alarm rings.

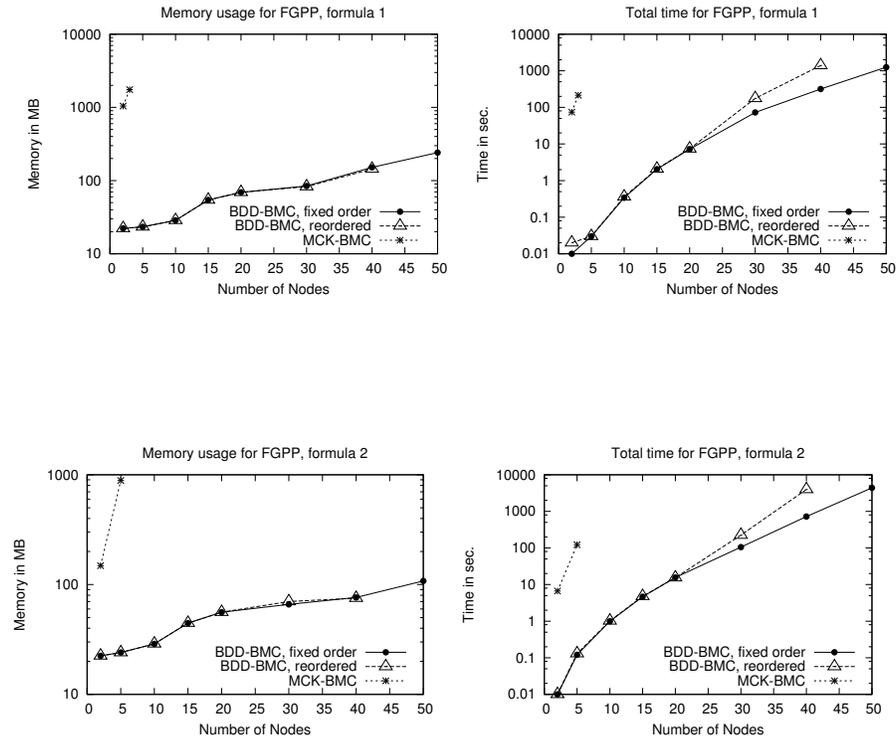
**A faulty train controller system (FTC)** (adapted from [10]) consists of a controller, and  $n$  trains (for  $n \geq 2$ ), one of which is dysfunctional. We consider the following specifications:  $\varphi_1 = G(\text{InTunnel}_1 \rightarrow K_{\text{Train}_1}(\bigwedge_{i=2}^n \neg \text{InTunnel}_i))$ , and  $\varphi_2 = G(K_{\text{Train}_1} \bigwedge_{i=1, j=2, i < j}^n \neg (\text{InTunnel}_i \wedge \text{InTunnel}_j))$ . The formula  $\varphi_1$  ( $i(n) = 5$ ) expresses that whenever a train is in the tunnel, it knows that the other train is not. The formula  $\varphi_2$  ( $i(n) = 5$ ) represents that trains are aware of the fact that they have exclusive access to the tunnel.

**Dining Cryptographers (DC)** [3] is a scalable anonymity protocol, which has been formalised and analysed in many works, e.g., [13, 16]. In this paper we assume the formalisation of DC in terms of a network of automata [13], and we consider the following specifications:  $\varphi_1 = G(\text{odd} \wedge \neg \text{paid}_1 \rightarrow \bigvee_{i=2}^n K_1(\text{paid}_i))$ ,  $\varphi_2 = G(\neg \text{paid}_1 \rightarrow K_1(\bigvee_{i=2}^n \text{paid}_i))$ ,  $\varphi_3 = G(\text{odd} \rightarrow C_{1, \dots, n} \neg (\bigvee_{i=1}^n \text{paid}_i))$ . The formula  $\varphi_1$  ( $i(n) = 4n + 2$ ) expresses that always when the number of uttered differences is odd and the first cryptographer has not paid for dinner, then he knows the cryptographer who paid for dinner. The formula  $\varphi_2$  ( $i(n) = 2$ ) states that it is always true that if the first cryptographer has not paid for dinner, then he knows that some other cryptographer pays. The formula  $\varphi_3$  ( $i(n) = 4n + 2$ ) states that always when the number of uttered differences is odd, than it is

common knowledge of all the cryptographers that none of the cryptographers has paid for dinner.

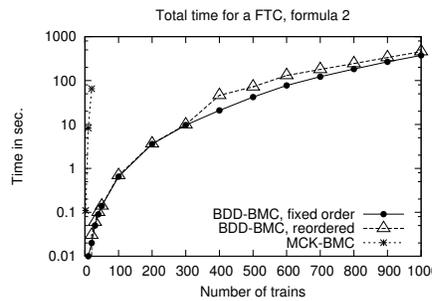
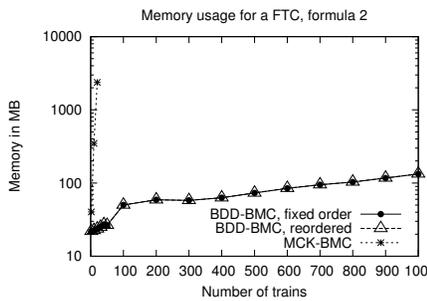
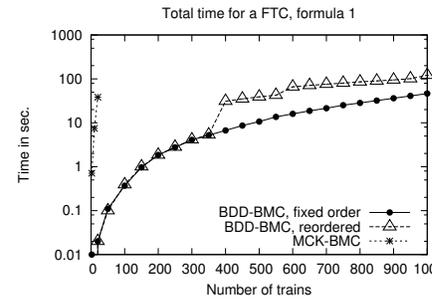
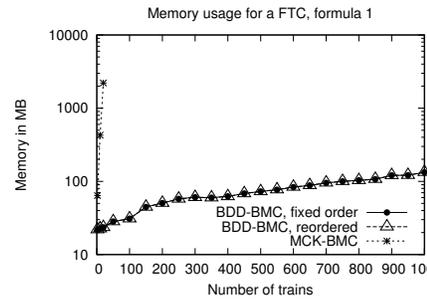
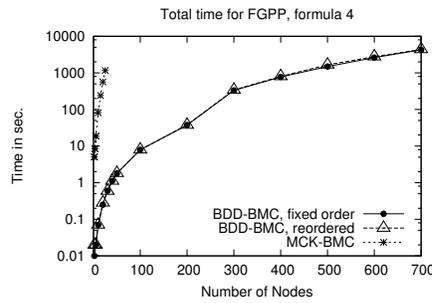
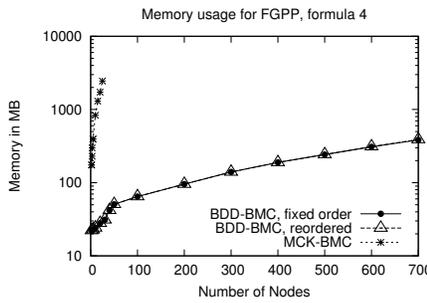
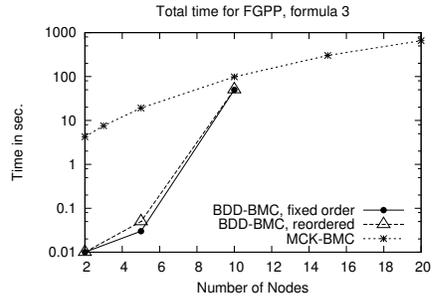
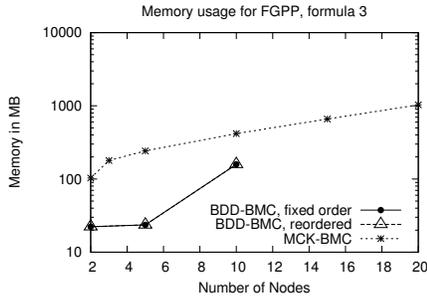
The local states and their protocols for all our benchmarks can be found at the webpage of VerICS.

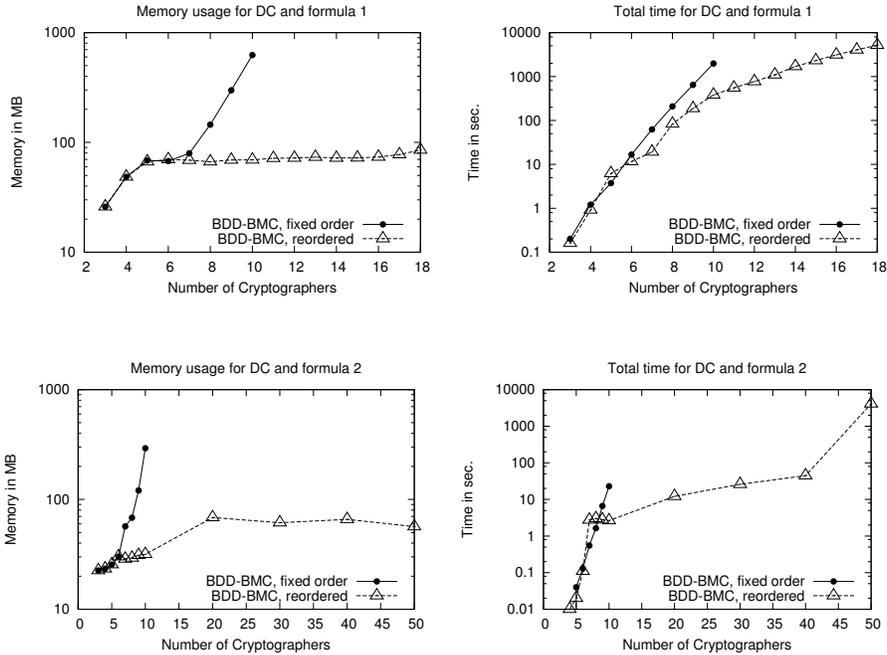
## 4.2 Performance Evaluation



In most of the considered benchmarks, our BDD-based method is superior to MCK, sometimes even by several orders of magnitude. However, the performance of MCK is much better in the case of DC and  $\varphi_3$  of FGPP. For DC, contrary to our IIS-based benchmarks, the length of the counterexamples is constant for MCK independently on the number of the cryptographers. We decided not to include these results because it would not be fair to compare verification methods working on the models of significantly different sizes.

The reordering of the BDD variables does not cause any significant change of the performance in the case of FGPP and FTC, but for DC it reduces the memory consumption. Therefore the fixed interleaving order we used can often be considered optimal, but the loss in the verification time to reorder the variables,





in favour of reducing memory consumption, is also not significant and is often worth the tradeoff.

In most cases, BDD-BMC spends a considerable amount of time on encoding the system. Therefore, BDD-BMC may provide additional time gains when verifying multiple specifications of the same system using the prepared encoding, that does not depend on a verified property.

## 5 Final Remarks

We have proposed, implemented, and experimentally evaluated our bounded model checking method based on BDDs for the combination of linear time with knowledge properties. The experimental results show that the approach is very promising. Moreover, our BDD-based BMC is a complete method, i.e., it can also be used to check that the verified existential properties are false in the considered model.

In the future we are going to compare our method with the SAT-based BMC method for LTLK described in [20]. We are also going to extend the presented algorithm to handle CTL\*K properties. Because our implementation is in its preliminary stage, we also need to improve it in many areas, e.g. the encoding of the transition relation, is not as effective as it could be, because it does not take advantage of any kind of partitioning methods.

## References

1. A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. In *Highly Dependable Software*, volume 58 of *Advances in Computers*. Academic Press, 2003.
2. R. Bordini, M. Fisher, C. Pardavila, W. Visser, and M. Wooldridge. Model checking multi-agent programs with CASP. In *Proc. of CAV'03*, volume 2725 of *LNCS*, pp. 110–113. Springer-Verlag, 2003.
3. D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.
4. E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In *Proc. of CAV'94*, volume 818 of *LNCS*, pp. 415–427. Springer-Verlag, 1994.
5. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
6. K. Engelhardt, R. van der Meyden, and Y. Moses. Knowledge and the logic of local propositions. In *Proc. of TARK'98*, pp. 29–41, 1998.
7. R. Fagin, J. Y. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.
8. P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proc. of CAV'04*, volume 3114 of *LNCS*, pp. 479–483. Springer-Verlag, 2004.
9. W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In *Proc. of SPIN'02*, volume 2318 of *LNCS*, pp. 95–111. Springer-Verlag, 2002.
10. W. van der Hoek and M. Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.
11. X. Huang, C. Luo, and R. van der Meyden. Improved bounded model checking for a fair branching-time temporal epistemic logic. In *Proc. of MoChArt'2010*, LNAI. Springer, 2011.
12. A. Jones and A. Lomuscio. A BDD-based BMC approach for the verification of multi-agent systems. In *Proc. of CS&P'09*, volume 1, pp. 253–264. Warsaw University, 2009.
13. M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. *Fundam. Inform.*, 72(1-2):215–234, 2006.
14. M. Kacprzak, A. Lomuscio, and W. Penczek. From bounded to unbounded model checking for temporal epistemic logic. *Fundam. Inform.*, 63(2-3):221–240, 2004.
15. A. Lomuscio, W. Penczek, and H. Qu. Partial order reduction for model checking interleaved multi-agent systems. In *AAMAS, IFAAMAS Press.*, pp. 659–666, 2010.
16. R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proc. of CSFW-17*, pp. 280–291. IEEE Computer Society, 2004.
17. R. van der Meyden and N. V. Shilov. Model checking knowledge and time in systems with perfect recall. In *Proc. of FSTTCS'99*, *LNCS* 1738, pp. 432–445. Springer-Verlag, 1999.
18. D. Peled. All From One, One For All: On Model Checking Using Representatives. In *Proc. of CAV'93*, *LNCS* 697, pp. 409–423. Springer-Verlag, 1993.
19. W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundam. Inform.*, 55(2):167–185, 2003.
20. W. Penczek, B. Woźna, and A. Zbrzezny. Towards SAT-based BMC for LTLK over Interleaved Interpreted Systems. *Submitted to CS&P'2011*.
21. F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 5(2):235–251, 2007.
22. K. Su, A. Sattar, and X. Luo. Model checking temporal logics of knowledge via OBDDs. *The Computer Journal*, 50(4):403–420, 2007.